IsaSAT: Heuristics and Code Generation

Mathias Fleury, Jasmin Blanchette, Peter Lammich

August 2, 2023

Contents

1 Refinement of Literals **5**
- 1.1 Literals as Natural Numbers . . . 5
  - 1.1.1 Definition . . . 5
  - 1.1.2 Lifting to annotated literals . . . 6
- 1.2 Conflict Clause . . . 6
- 1.3 Atoms with bound . . . 6
- 1.4 Operations with set of atoms. . . . 7
- 1.5 Set of atoms with bound . . . 8
- 1.6 Instantion for code generation . . . 13
  - 1.6.1 Literals as Natural Numbers . . . 13
  - 1.6.2 State Conversion . . . 14
  - 1.6.3 Code Generation . . . 14
- 1.7 Polarities . . . 16
  - 1.7.1 Atom-Of . . . 19

2 The memory representation: Arenas **31**
- 2.1 Status of a clause . . . 32
- 2.2 Definition . . . 33
- 2.3 Separation properties . . . 36
- 2.4 MOP versions of operations . . . 62
  - 2.4.1 Access to literals . . . 62
  - 2.4.2 Swapping of literals . . . 63
  - 2.4.3 Position Saving . . . 63
  - 2.4.4 Clause length . . . 64
- 2.5 Virtual Domain . . . 73
- 2.6 Virtual domain . . . 73
- 2.7 Code Generation . . . 75

3 The memory representation: Manipulation of all clauses **95**

4 Efficient Trail **105**
- 4.1 Types . . . 105
- 4.2 Control Stack . . . 106
- 4.3 Encoding of the reasons . . . 109
- 4.4 Definition of the full trail . . . 110
- 4.5 Code generation . . . 111
  - 4.5.1 Conversion between incomplete and complete mode . . . 111
  - 4.5.2 Level of a literal . . . 111
  - 4.5.3 Current level . . . 112

4.5.4 Polarity . . . 112
4.5.5 Length of the trail . . . 113
4.5.6 Consing elements . . . 113
4.5.7 Setting a new literal . . . 119
4.5.8 Polarity: Defined or Undefined . . . 120
4.5.9 Reasons . . . 120
4.6 Direct access to elements in the trail . . . 122
4.7 Options . . . 128
4.7.1 Definition . . . 128
4.7.2 Refinement . . . 129
4.8 Moving averages . . . 131
4.8.1 Phase saving . . . 132

5 Phase Saving **133**
5.1 Rephasing . . . 133
5.2 Statistics . . . 147
5.3 Information related to restarts . . . 155
5.4 Heuristics . . . 156
5.4.1 Number of clauses . . . 165
5.4.2 Lifting to heuristic level . . . 172
5.4.3 Variable-Move-to-Front . . . 244
5.4.4 Hash for lists . . . 296

6 LBD **297**
6.1 Types and relations . . . 297
6.2 Testing if a level is marked . . . 297
6.3 Marking more levels . . . 298
6.4 Cleaning the marked levels . . . 298
6.5 Extracting the LBD . . . 301

7 Refinement of the Watched Function **305**
7.1 Definition . . . 305
7.2 Operations . . . 305
7.3 Rewatch . . . 306

8 Clauses Encoded as Positions **319**

9 Profiling **331**
9.1 Occurrence lists . . . 412
9.1.1 Abstract Occurrence Lists . . . 412
9.1.2 Concrete Occurrence lists . . . 414
9.2 Clause Marking . . . 418
9.2.1 Abstract Representation . . . 418
9.2.2 Concrete Representation . . . 421
9.3 ACIDS . . . 424
9.3.1 Access Function . . . 433

10 Complete state **437**
10.1 VMTF . . . 437
10.1.1 Conflict . . . 437
10.2 Full state . . . 438
10.3 Lift Operations to State . . . 445
10.4 More theorems . . . 446
10.5 Shared Code Equations . . . 449
10.6 Fast to slow conversion . . . 451
10.6.1 Lifting of Options . . . 464

11 Sorting of clauses **473**

12 Printing information about progress **489**
12.0.1 Print Information for IsaSAT . . . 489

13 VMTF Decision Heuristic **631**
13.1 Code generation for the VMTF decision heuristic and the trail . . . 631
13.2 Bumping . . . 642
13.3 Backtrack level for Restarts . . . 651

14 Propagation: Inner Loop **723**
14.1 Find replacement . . . 723
14.2 Updates . . . 731
14.3 Full inner loop . . . 738
14.4 DRAT proof generation . . . 768

15 Backtrack **773**
15.1 Backtrack with direct extraction of literal if highest level . . . 773

16 Backtrack **781**
16.1 Backtrack with direct extraction of literal if highest level . . . 781
16.2 Backtrack with direct extraction of literal if highest level . . . 833
16.2.1 Access Function . . . 843

17 Initialisation **853**
17.1 Code for the initialisation of the Data Structure . . . 853
17.1.1 Initialisation of the state . . . 853
17.1.2 Parsing . . . 859
17.1.3 Extractions of the atoms in the state . . . 874
17.1.4 Parsing . . . 882
17.1.5 Conversion to normal state . . . 885

18 Propagation Loop And Conflict **985**
18.1 Unit Propagation, Inner Loop . . . 985
18.2 Unit propagation, Outer Loop . . . 985

19 Decide **991**

20 Combining Together: the Other Rules **1007**

21 Combining Together: the Other Rules **1009**

22 Restarts **1029**
22.1 Simplification of binary clauses 1168
22.2 Forward subsumption 1214
22.2.1 Algorithm 1214
22.2.2 Refinement to isasat. 1247

23 Full CDCL with Restarts **1345**

24 Full IsaSAT **1367**
24.1 Correctness Relation 1369
24.2 Refinements of the Whole SAT Solver 1373
24.3 Refinements of the Whole Bounded SAT Solver 1413

25 Code of Full IsaSAT **1443**

theory *WB-More-Word*
imports *Word-Lib.Many-More Isabelle-LLVM.Bits-Natural*
begin

lemma *nat-uint-XOR*: ‹*nat* (*uint* (*a XOR b*)) = *nat* (*uint a*) *XOR nat* (*uint b*)›
if *len*: ‹*LENGTH*('*a*) > *0*›
for *a b* :: ‹'*a* ::*len Word.word*›
proof −
have *1*: ‹*uint* ((*word-of-int*:: *int* ⇒ '*a Word.word*)(*uint a*)) = *uint a*›
by (*subst* (*2*) *word-of-int-uint*[*of a*, *symmetric*]) (*rule refl*)
have *H*: ‹*nat* (*bintrunc n* (*a XOR b*)) = *nat* (*bintrunc n a XOR bintrunc n b*)›
if ‹*n*> *0*› **for** *n* **and** *a* :: *int* **and** *b* :: *int*
using *that*
proof (*induction n arbitrary*: *a b*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*) **note** *IH* = *this*(*1*) **and** *Suc* = *this*(*2*)
then show *?case*
by (*cases n*) *simp-all*
qed
have ‹*nat* (*bintrunc LENGTH*('*a*) (*a XOR b*)) = *nat* (*bintrunc LENGTH*('*a*) *a XOR bintrunc LENGTH*('*a*) *b*)› **for** *a b*
using *len H*[*of* ‹*LENGTH*('*a*)› *a b*] **by** *auto*
then have ‹*nat* (*uint* (*a XOR b*)) = *nat* (*uint a XOR uint b*)›
by *transfer*
then show *?thesis*
unfolding *xor-nat-def* **by** *auto*
qed

lemma *bitXOR-1-if-mod-2-int*: ‹*L OR 1* = (*if L mod 2* = *0 then L* + *1 else L*)› **for** *L* :: *int*
apply (*rule bin-rl-eqI*)
unfolding *bin-rest-OR bin-last-OR*
by (*auto simp*:)

lemma *bitOR-1-if-mod-2-nat*:
‹*L OR 1* = (*if L mod 2* = *0 then L* + *1 else L*)›
‹*L OR* (*Suc 0*) = (*if L mod 2* = *0 then L* + *1 else L*)› **for** *L* :: *nat*
proof −
have *H*: ‹*L OR 1* = *L* + (*if bin-last* (*int L*) *then 0 else 1*)›
unfolding *or-nat-def*

```
    apply (auto simp: or-nat-def bin-last-def xor-one-eq
      bitXOR-1-if-mod-2-int)
      apply presburger
    done
  show ‹L OR 1 = (if L mod 2 = 0 then L + 1 else L)›
    unfolding H
    apply (auto simp: or-nat-def bin-last-def)
    apply presburger+
    done
  then show ‹L OR (Suc 0) = (if L mod 2 = 0 then L + 1 else L)›
    by simp
qed

lemma bin-pos-same-XOR3:
  ‹a XOR a XOR c = c›
  ‹a XOR c XOR a = c› for a c :: int
  by (metis bin-ops-same(3) int-xor-assoc int-xor-zero)+

lemma bin-pos-same-XOR3-nat:
  ‹a XOR a XOR c = c›
  ‹a XOR c XOR a = c› for a c :: nat
 unfolding xor-nat-def by (auto simp: bin-pos-same-XOR3)

end
theory IsaSAT-Literals
  imports More-Sepref.WB-More-Refinement Word-Lib.Many-More
    Watched-Literals.Watched-Literals-Watch-List
    Entailment-Definition.Partial-Herbrand-Interpretation
    Isabelle-LLVM.Bits-Natural
begin
```

Chapter 1

Refinement of Literals

1.1 Literals as Natural Numbers

1.1.1 Definition

lemma *Pos-div2-iff*:
‹*Pos* ((*bb* :: *nat*) *div* 2) = *b* ⟷ *is-pos b* ∧ (*bb* = 2 ∗ *atm-of b* ∨ *bb* = 2 ∗ *atm-of b* + 1)›
by (*cases b*) *auto*
lemma *Neg-div2-iff*:
‹*Neg* ((*bb* :: *nat*) *div* 2) = *b* ⟷ *is-neg b* ∧ (*bb* = 2 ∗ *atm-of b* ∨ *bb* = 2 ∗ *atm-of b* + 1)›
by (*cases b*) *auto*

Modeling *nat literal* via the transformation associating $(2::'a) * n$ or $(2::'a) * n + (1::'a)$ has some advantages over the transformation to positive or negative integers: 0 is not an issue. It is also a bit faster according to Armin Biere.

fun *nat-of-lit* :: ‹*nat literal* ⇒ *nat*› **where**
‹*nat-of-lit* (*Pos L*) = 2∗*L*›
| ‹*nat-of-lit* (*Neg L*) = 2∗*L* + 1›

lemma *nat-of-lit-def*: ‹*nat-of-lit L* = (*if is-pos L then* 2 ∗ *atm-of L else* 2 ∗ *atm-of L* + 1)›
by (*cases L*) *auto*

fun *literal-of-nat* :: ‹*nat* ⇒ *nat literal*› **where**
‹*literal-of-nat n* = (*if even n then Pos* (*n div* 2) *else Neg* (*n div* 2))›

lemma *lit-of-nat-nat-of-lit*[*simp*]: ‹*literal-of-nat* (*nat-of-lit L*) = *L*›
by (*cases L*) *auto*

lemma *nat-of-lit-lit-of-nat*[*simp*]: ‹*nat-of-lit* (*literal-of-nat n*) = *n*›
by *auto*

lemma *atm-of-lit-of-nat*: ‹*atm-of* (*literal-of-nat n*) = *n div* 2›
by *auto*

There is probably a more "closed" form from the following theorem, but it is unclear if that is useful or not.

lemma *uminus-lit-of-nat*:
‹− (*literal-of-nat n*) = (*if even n then literal-of-nat* (*n*+1) *else literal-of-nat* (*n*−1))›
by (*auto elim*!: *oddE*)

lemma *literal-of-nat-literal-of-nat-eq*[*iff*]: ‹*literal-of-nat x* = *literal-of-nat xa* ⟷ *x* = *xa*›

by *auto presburger+*

definition *nat-lit-rel* :: ‹(*nat* × *nat literal*) *set*› **where**
‹*nat-lit-rel* = *br literal-of-nat* (λ-. *True*)›

lemma *ex-literal-of-nat*: ‹∃ *bb*. *b* = *literal-of-nat bb*›
by (*cases b*)
(*auto simp*: *nat-of-lit-def split*: *if-splits*; *presburger*; *fail*)+

1.1.2 Lifting to annotated literals

fun *pair-of-ann-lit* :: ‹('*a*, '*b*) *ann-lit* ⇒ '*a literal* × '*b option*› **where**
‹*pair-of-ann-lit* (*Propagated L D*) = (*L*, *Some D*)›
| ‹*pair-of-ann-lit* (*Decided L*) = (*L*, *None*)›

fun *ann-lit-of-pair* :: ‹'*a literal* × '*b option* ⇒ ('*a*, '*b*) *ann-lit*› **where**
‹*ann-lit-of-pair* (*L*, *Some D*) = *Propagated L D*›
| ‹*ann-lit-of-pair* (*L*, *None*) = *Decided L*›

lemma *ann-lit-of-pair-alt-def*:
‹*ann-lit-of-pair* (*L*, *D*) = (*if D* = *None then Decided L else Propagated L* (*the D*))›
by (*cases D*) *auto*

lemma *ann-lit-of-pair-pair-of-ann-lit*: ‹*ann-lit-of-pair* (*pair-of-ann-lit L*) = *L*›
by (*cases L*) *auto*

lemma *pair-of-ann-lit-ann-lit-of-pair*: ‹*pair-of-ann-lit* (*ann-lit-of-pair L*) = *L*›
by (*cases L*; *cases* ‹*snd L*›) *auto*

lemma *literal-of-neq-eq-nat-of-lit-eq-iff*: ‹*literal-of-nat b* = *L* ⟷ *b* = *nat-of-lit L*›
by (*auto simp del*: *literal-of-nat.simps*)

lemma *nat-of-lit-eq-iff*[*iff*]: ‹*nat-of-lit xa* = *nat-of-lit x* ⟷ *x* = *xa*›
apply (*cases x*; *cases xa*) **by** *auto presburger+*

definition *ann-lit-rel*:: ‹('*a* × *nat*) *set* ⇒ ('*b* × *nat option*) *set* ⇒
(('*a* × '*b*) × (*nat*, *nat*) *ann-lit*) *set*› **where**
ann-lit-rel-internal-def:
‹*ann-lit-rel R R'* = {(*a*, *b*). ∃ *c d*. (*fst a*, *c*) ∈ *R* ∧ (*snd a*, *d*) ∈ *R'* ∧
b = *ann-lit-of-pair* (*literal-of-nat c*, *d*)}›

1.2 Conflict Clause

definition *the-is-empty* **where**
‹*the-is-empty D* = *Multiset.is-empty* (*the D*)›

1.3 Atoms with bound

definition *unat32-max* :: *nat* **where**
‹*unat32-max* ≡ *2^32−1*›

definition *unat64-max* :: *nat* **where**
‹*unat64-max* ≡ *2^64−1*›

definition *snat32-max* :: *nat* **where**

‹*snat32-max* ≡ *2^31−1*›

definition *snat64-max* :: *nat* **where**
‹*snat64-max* ≡ *2^63−1*›

lemma *li-unat32-maxdiv2-le-unit32-max*: ‹*a* ≤ *unat32-max div 2* + *1* ⟹ *a* ≤ *unat32-max*›
by (*auto simp*: *unat32-max-def*)

lemma *unat64-max-uint-def*: ‹*unat* (−*1* :: *64 Word.word*) = *unat64-max*›
proof −
have ‹*unat* (−*1* :: *64 Word.word*) = *unat* (− *Numeral1* :: *64 Word.word*)›
unfolding *numeral.numeral-One* **..**
also have ‹... = *unat64-max*›
unfolding *unat-bintrunc-neg*
apply (*simp add*: *unat64-max-def*)
apply (*subst numeral-eq-Suc*; *subst semiring-bit-operations-class.mask-Suc-exp*; *simp*)+
done
finally show *?thesis* **.**
qed

1.4 Operations with set of atoms.

context
fixes $\mathcal{A}_{in}$ :: ‹*nat multiset*›
begin

abbreviation D_0 :: ‹(*nat* × *nat literal*) *set*› **where**
‹D_0 ≡ (λ*L*. (*nat-of-lit L*, *L*)) ‘ *set-mset* ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$)›

The following lemma was necessary at some point to prove the existence of a watch list.

lemma *ex-list-watched*:
fixes *W* :: ‹*nat literal* ⇒ *'a list*›
shows ‹∃ *aa*. ∀ *x*∈#$\mathcal{L}_{all}$ $\mathcal{A}_{in}$. *nat-of-lit x* < *length aa* ∧ *aa* ! *nat-of-lit x* = *W x*›
(**is** ‹∃ *aa*. *?P aa*›)
proof −
define D' **where** ‹D' = D_0›
define $\mathcal{L}_{all}'$ **where** ‹$\mathcal{L}_{all}'$ = $\mathcal{L}_{all}$›
define D'' **where** ‹D'' = *mset-set* (*snd* ‘ D')›
let *?f* = ‹(λ*L a*. *a*[*nat-of-lit L*:= *W L*])›
interpret *comp-fun-commute ?f*
apply *standard*
apply (*case-tac* ‹*y* = *x*›)
apply (*solves simp*)
apply (*intro ext*)
apply (*subst* (*asm*) *lit-of-nat-nat-of-lit*[*symmetric*])
apply (*subst* (*asm*)(*3*) *lit-of-nat-nat-of-lit*[*symmetric*])
apply (*clarsimp simp only*: *comp-def intro*!: *list-update-swap*)
done
define *aa* **where**
‹*aa* ≡ *fold-mset ?f* (*replicate* (*1*+*Max* (*nat-of-lit* ‘ *snd* ‘ D')) []) (*mset-set* (*snd* ‘ D'))›
have *length-fold*: ‹*length* (*fold-mset* (λ*L a*. *a*[*nat-of-lit L* := *W L*]) *l M*) = *length l*› **for** *l M*
by (*induction M*) *auto*
have *length-aa*: ‹*length aa* = *Suc* (*Max* (*nat-of-lit* ‘ *snd* ‘ D'))›
unfolding *aa-def* D''*-def*[*symmetric*] **by** (*simp add*: *length-fold*)
have *H*: ‹*x* ∈# $\mathcal{L}_{all}'$ ⟹

```
    length l ≥ Suc (Max (nat-of-lit ‘ set-mset (𝓛all ′))) ⟹
    fold-mset (λL a. a[nat-of-lit L := W L]) l (remdups-mset (𝓛all ′)) ! nat-of-lit x = W x›
  for x l 𝓛all ′
  unfolding 𝓛all ′-def[symmetric]
  apply (induction 𝓛all ′ arbitrary: l)
  subgoal by simp
  subgoal for xa Ls l
    apply (case-tac ‹(nat-of-lit ‘ set-mset Ls) = {}›)
     apply (solves simp)
    apply (auto simp: less-Suc-eq-le length-fold)
    done
  done
 have H′: ‹aa ! nat-of-lit x = W x› if ‹x ∈# 𝓛all 𝒜in› for x
  using that unfolding aa-def D′-def
  by (auto simp: D′-def image-image remdups-mset-def[symmetric]
      less-Suc-eq-le intro!: H)
 have ‹?P aa›
  by (auto simp: D′-def image-image remdups-mset-def[symmetric]
      less-Suc-eq-le length-aa H′)
 then show ?thesis
  by blast
qed
```

The following two definitions are very important in practise for the invariants for the SAT solver.

```
definition isasat-input-bounded where
 [simp]: ‹isasat-input-bounded = (∀ L ∈# 𝓛all 𝒜in. nat-of-lit L ≤ unat32-max)›

definition isasat-input-nempty where
 [simp]: ‹isasat-input-nempty = (set-mset 𝒜in ≠ {})›

definition isasat-input-bounded-nempty where
 ‹isasat-input-bounded-nempty = (isasat-input-bounded ∧ isasat-input-nempty)›
```

1.5 Set of atoms with bound

```
context
 assumes in-𝓛all-less-unat32-max: ‹isasat-input-bounded›
begin

lemma in-𝓛all-less-unat32-max′: ‹L ∈# 𝓛all 𝒜in ⟹ nat-of-lit L ≤ unat32-max›
 using in-𝓛all-less-unat32-max by auto

lemma in-𝒜in-less-than-unat32-max-div-2:
 ‹L ∈# 𝒜in ⟹ L ≤ unat32-max div 2›
 using in-𝓛all-less-unat32-max′[of ‹Neg L›]
 unfolding Ball-def atms-of-𝓛all-𝒜in in-𝓛all-atm-of-in-atms-of-iff
 by (auto simp: unat32-max-def)

lemma simple-clss-size-upper-div2′:
 assumes
   lits: ‹literals-are-in-𝓛in 𝒜in C› and
   dist: ‹distinct-mset C› and
   tauto: ‹¬tautology C› and
   in-𝓛all-less-unat32-max: ‹∀ L ∈# 𝓛all 𝒜in. nat-of-lit L < unat32-max − 1›
 shows ‹size C ≤ unat32-max div 2›
```

proof −
let *?C* = ‹*atm-of* ‘# *C*›
have ‹*distinct-mset ?C*›
proof (*rule ccontr*)
assume ‹¬ *?thesis*›
then obtain *K* **where** ‹¬*count* (*atm-of* ‘# *C*) *K* ≤ *Suc 0*›
unfolding *distinct-mset-count-less-1*
by *auto*
then have ‹*count* (*atm-of* ‘# *C*) *K* ≥ *2*›
by *auto*
then obtain *L L′ C′* **where**
C: ‹*C* = {#*L*, *L′*#} + *C′*› **and** *L-L′*: ‹*atm-of L* = *atm-of L′*›
by (*auto dest*!: *count-image-mset-multi-member-split-2*)
then show *False*
using *dist tauto* **by** (*auto simp*: *atm-of-eq-atm-of tautology-add-mset*)
qed
then have *card*: ‹*size ?C* = *card* (*set-mset ?C*)›
using *distinct-mset-size-eq-card* **by** *blast*
have *size*: ‹*size ?C* = *size C*›
using *dist tauto*
by (*induction C*) (*auto simp*: *tautology-add-mset*)
have *m*: ‹*set-mset ?C* ⊆ {*0..<unat32-max div 2*}›
proof
fix *L*
assume ‹*L* ∈ *set-mset ?C*›
then have ‹*L* ∈ *atms-of* (*$\mathcal{L}_{all}$* *$\mathcal{A}_{in}$*)›
using *lits* **by** (*auto simp*: *literals-are-in-$\mathcal{L}_{in}$-def atm-of-lit-in-atms-of*
in-all-lits-of-m-ain-atms-of-iff subset-iff)
then have ‹*Pos L* ∈# (*$\mathcal{L}_{all}$* *$\mathcal{A}_{in}$*)›
using *lits* **by** (*auto simp*: *in-$\mathcal{L}_{all}$-atm-of-in-atms-of-iff*)
then have ‹*nat-of-lit* (*Pos L*) < *unat32-max* − *1*›
using *in-$\mathcal{L}_{all}$-less-unat32-max* **by** (*auto simp*: *atm-of-lit-in-atms-of*
in-all-lits-of-m-ain-atms-of-iff subset-iff)
then have ‹*L* < *unat32-max div 2*›
by (*auto simp*: *atm-of-lit-in-atms-of*
in-all-lits-of-m-ain-atms-of-iff subset-iff unat32-max-def)
then show ‹*L* ∈ {*0..<unat32-max div 2*}›
by (*auto simp*: *atm-of-lit-in-atms-of unat32-max-def*
in-all-lits-of-m-ain-atms-of-iff subset-iff)
qed
moreover have ‹*card* ... = *unat32-max div 2*›
by *auto*
ultimately have ‹*card* (*set-mset ?C*) ≤ *unat32-max div 2*›
using *card-mono*[*OF* - *m*] **by** *auto*
then show *?thesis*
unfolding *card*[*symmetric*] *size* **.**
qed

lemma *simple-clss-size-upper-div2*:
assumes
lits: ‹*literals-are-in-$\mathcal{L}_{in}$* *$\mathcal{A}_{in}$* *C*› **and**
dist: ‹*distinct-mset C*› **and**
tauto: ‹¬*tautology C*›
shows ‹*size C* ≤ *1* + *unat32-max div 2*›
proof −

```
  let ?C = ‹atm-of `# C›
  have ‹distinct-mset ?C›
  proof (rule ccontr)
    assume ‹¬ ?thesis›
    then obtain K where ‹¬count (atm-of `# C) K ≤ Suc 0›
      unfolding distinct-mset-count-less-1
      by auto
    then have ‹count (atm-of `# C) K ≥ 2›
      by auto
    then obtain L L' C' where
      C: ‹C = {#L, L'#} + C'› and L-L': ‹atm-of L = atm-of L'›
      by (auto dest!: count-image-mset-multi-member-split-2)
    then show False
      using dist tauto by (auto simp: atm-of-eq-atm-of tautology-add-mset)
  qed
  then have card: ‹size ?C = card (set-mset ?C)›
    using distinct-mset-size-eq-card by blast
  have size: ‹size ?C = size C›
    using dist tauto
    by (induction C) (auto simp: tautology-add-mset)
  have m: ‹set-mset ?C ⊆ {0..unat32-max div 2}›
  proof
    fix L
    assume ‹L ∈ set-mset ?C›
    then have ‹L ∈ atms-of (\mathcal{L}_{all}  \mathcal{A}_{in})›
    using lits by (auto simp: literals-are-in-\mathcal{L}_{in}-def atm-of-lit-in-atms-of
        in-all-lits-of-m-ain-atms-of-iff subset-iff)
    then have ‹Neg L ∈# (\mathcal{L}_{all}  \mathcal{A}_{in})›
      using lits by (auto simp: in-\mathcal{L}_{all}-atm-of-in-atms-of-iff)
    then have ‹nat-of-lit (Neg L) ≤ unat32-max›
      using in-\mathcal{L}_{all}-less-unat32-max by (auto simp: atm-of-lit-in-atms-of
        in-all-lits-of-m-ain-atms-of-iff subset-iff)
    then have ‹L ≤ unat32-max div 2›
       by (auto simp: atm-of-lit-in-atms-of
        in-all-lits-of-m-ain-atms-of-iff subset-iff unat32-max-def)
    then show ‹L ∈ {0 .. unat32-max div 2}›
       by (auto simp: atm-of-lit-in-atms-of unat32-max-def
        in-all-lits-of-m-ain-atms-of-iff subset-iff)
  qed
  moreover have ‹card ... =  1 + unat32-max div 2›
    by auto
  ultimately have ‹card (set-mset ?C) ≤ 1 + unat32-max div 2›
    using card-mono[OF - m] by auto
  then show ?thesis
    unfolding card[symmetric] size .
qed

lemma clss-size-unat32-max:
  assumes
   lits: ‹literals-are-in-\mathcal{L}_{in} \mathcal{A}_{in} C› and
   dist: ‹distinct-mset C›
  shows ‹size C ≤ unat32-max + 2›
proof −
  let ?posC = ‹filter-mset is-pos C›
  let ?negC = ‹filter-mset is-neg C›
  have C: ‹C = ?posC + ?negC›
```

```
    apply (subst multiset-partition[of - is-pos])
    by auto
  have ‹literals-are-in-\mathcal{L}_in \mathcal{A}_in ?posC›
    by (rule literals-are-in-\mathcal{L}_in-mono[OF lits]) auto
  moreover have ‹distinct-mset ?posC›
    by (rule distinct-mset-mono[OF -dist]) auto
  ultimately have pos: ‹size ?posC ≤ 1 + unat32-max div 2›
    by (rule simple-clss-size-upper-div2) (auto simp: tautology-decomp)

  have ‹literals-are-in-\mathcal{L}_in \mathcal{A}_in ?negC›
    by (rule literals-are-in-\mathcal{L}_in-mono[OF lits]) auto
  moreover have ‹distinct-mset ?negC›
    by (rule distinct-mset-mono[OF -dist]) auto
  ultimately have neg: ‹size ?negC ≤ 1 + unat32-max div 2›
    by (rule simple-clss-size-upper-div2) (auto simp: tautology-decomp)

  show ?thesis
    apply (subst C)
    apply (subst size-union)
    using pos neg by linarith
qed

lemma clss-size-upper:
  assumes
   lits: ‹literals-are-in-\mathcal{L}_in \mathcal{A}_in C› and
   dist: ‹distinct-mset C› and
   in-\mathcal{L}_all-less-unat32-max: ‹∀ L ∈# \mathcal{L}_all \mathcal{A}_in. nat-of-lit L < unat32-max − 1›
 shows ‹size C ≤ unat32-max›
proof −
  let ?A = ‹remdups-mset (atm-of `# C)›
  have [simp]: ‹distinct-mset (poss ?A)› ‹distinct-mset (negs ?A)›
    by (simp-all add: distinct-image-mset-inj inj-on-def)

  have ‹C ⊆# poss ?A + negs ?A›
    apply (rule distinct-subseteq-iff[THEN iffD1])
    subgoal by (auto simp: dist distinct-mset-add disjunct-not-in)
    subgoal
      apply rule
      using literal.exhaust-sel by (auto simp: image-iff )
    done
  have [simp]: ‹literals-are-in-\mathcal{L}_in \mathcal{A}_in (poss ?A)› ‹literals-are-in-\mathcal{L}_in \mathcal{A}_in (negs ?A)›
    using lits
    by (auto simp: literals-are-in-\mathcal{L}_in-negs-remdups-mset literals-are-in-\mathcal{L}_in-poss-remdups-mset)

  have ‹¬ tautology (poss ?A)› ‹¬ tautology (negs ?A)›
    by (auto simp: tautology-decomp)
  then have ‹size (poss ?A) ≤ unat32-max div 2› and ‹size (negs ?A) ≤ unat32-max div 2›
    using simple-clss-size-upper-div2'[of ‹poss ?A›]
      simple-clss-size-upper-div2'[of ‹negs ?A›] in-\mathcal{L}_all-less-unat32-max
    by auto
  then have ‹size C ≤ unat32-max div 2 + unat32-max div 2›
    using ‹C ⊆# poss (remdups-mset (atm-of `# C)) + negs (remdups-mset (atm-of `# C))›
      size-mset-mono by fastforce
  then show ?thesis by (auto simp: unat32-max-def)
qed
```

lemma
  assumes
    lits: ‹*literals-are-in-$\mathcal{L}_{in}$-trail $\mathcal{A}_{in}$ M*› **and**
    n-d: ‹*no-dup M*›
  shows
    literals-are-in-$\mathcal{L}_{in}$-trail-length-le-unat32-max:
      ‹*length M ≤ Suc (unat32-max div 2)*› **and**
    literals-are-in-$\mathcal{L}_{in}$-trail-count-decided-unat32-max:
      ‹*count-decided M ≤ Suc (unat32-max div 2)*› **and**
    literals-are-in-$\mathcal{L}_{in}$-trail-get-level-unat32-max:
      ‹*get-level M L ≤ Suc (unat32-max div 2)*›
proof −
  have ‹*length M = card (atm-of ‘ lits-of-l M)*›
    using *no-dup-length-eq-card-atm-of-lits-of-l*[*OF n-d*] **.**
  moreover have ‹*atm-of ‘ lits-of-l M ⊆ set-mset $\mathcal{A}_{in}$*›
    using *lits* **unfolding** *literals-are-in-$\mathcal{L}_{in}$-trail-atm-of* **by** *auto*
  ultimately have ‹*length M ≤ card (set-mset $\mathcal{A}_{in}$)*›
    by (*simp add*: *card-mono*)
  moreover {
    have ‹*set-mset $\mathcal{A}_{in}$ ⊆ {0 ..< (unat32-max div 2) + 1}*›
      using *in-$\mathcal{A}_{in}$-less-than-unat32-max-div-2* **by** (*fastforce simp*: *in-$\mathcal{L}_{all}$-atm-of-in-atms-of-iff*
          Ball-def atms-of-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$ unat32-max-def)
    from *subset-eq-atLeast0-lessThan-card*[*OF this*] **have** ‹*card (set-mset $\mathcal{A}_{in}$) ≤ unat32-max div 2 + 1*›
      .
  }
  ultimately show ‹*length M ≤ Suc (unat32-max div 2)*›
    by *linarith*
  moreover have ‹*count-decided M ≤ length M*›
    unfolding *count-decided-def* **by** *auto*
  ultimately show ‹*count-decided M ≤ Suc (unat32-max div 2)*› **by** *simp*
  then show ‹*get-level M L ≤ Suc (unat32-max div 2)*›
    using *count-decided-ge-get-level*[*of M L*]
    by *simp*
qed

lemma *length-trail-unat32-max-div2*:
  fixes *M* :: ‹(*nat*, *'b*) *ann-lits*›
  assumes
    M-$\mathcal{L}_{all}$: ‹*∀ L∈set M. lit-of L ∈# $\mathcal{L}_{all}$ $\mathcal{A}_{in}$*› **and**
    n-d: ‹*no-dup M*›
  shows ‹*length M ≤ unat32-max div 2 + 1*›
proof −
  have *dist-atm-M*: ‹*distinct-mset {#atm-of (lit-of x). x ∈# mset M#}*›
    using *n-d* **by** (*metis distinct-mset-mset-distinct mset-map no-dup-def*)
  have *incl*: ‹*atm-of ‘# lit-of ‘# mset M ⊆# remdups-mset (atm-of ‘# $\mathcal{L}_{all}$ $\mathcal{A}_{in}$)*›
    apply (*subst distinct-subseteq-iff*[*THEN iffD1*])
    using *assms dist-atm-M*
    by (*auto 5 5 simp*: *Decided-Propagated-in-iff-in-lits-of-l lits-of-def no-dup-distinct*
        atm-of-eq-atm-of)
  have *inj-on*: ‹*inj-on nat-of-lit (set-mset (remdups-mset ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$)))*›
    by (*auto simp*: *inj-on-def*)
  have *H*: ‹*xa ∈# $\mathcal{L}_{all}$ $\mathcal{A}_{in}$ ⟹ atm-of xa ≤ unat32-max div 2*› **for** *xa*
    using *in-$\mathcal{L}_{all}$-less-unat32-max*
    by (*cases xa*) (*auto simp*: *unat32-max-def*)
  have ‹*remdups-mset (atm-of ‘# $\mathcal{L}_{all}$ $\mathcal{A}_{in}$) ⊆# mset [0..< 1 + (unat32-max div 2)]*›

apply (*subst distinct-subseteq-iff*[*THEN iffD1*])
using *H distinct-image-mset-inj*[*OF inj-on*]
by (*force simp del*: *literal-of-nat.simps simp*: *distinct-mset-mset-set*
dest: *le-neq-implies-less*)+
note - = *size-mset-mono*[*OF this*]
moreover have ‹*size* (*nat-of-lit* '# *remdups-mset* ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$)) = *size* (*remdups-mset* ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$))›
by *simp*
ultimately have *2*: ‹*size* (*remdups-mset* (*atm-of* '# ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$))) ≤ *1* + *unat32-max div 2*›
by *auto*
from *size-mset-mono*[*OF incl*] **have** *1*: ‹*length M* ≤ *size* (*remdups-mset* (*atm-of* '# ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$)))›
unfolding *unat32-max-def count-decided-def*
by (*auto simp del*: *length-filter-le*)
with *2* **show** *?thesis*
by (*auto simp*: *unat32-max-def*)
qed

end

end

1.6 Instantion for code generation

instantiation *literal* :: (*default*) *default*
begin

definition *default-literal* **where**
‹*default-literal* = *Pos default*›
instance by *standard*

end

instantiation *fmap* :: (*type*, *type*) *default*
begin

definition *default-fmap* **where**
‹*default-fmap* = *fmempty*›
instance by *standard*

end

1.6.1 Literals as Natural Numbers

definition *propagated* **where**
‹*propagated L C* = (*L*, *Some C*)›

definition *decided* **where**
‹*decided L* = (*L*, *None*)›

definition *uminus-lit-imp* :: ‹*nat* ⇒ *nat*› **where**
‹*uminus-lit-imp L* = *L XOR 1*›

lemma *uminus-lit-imp-uminus*:
‹(*RETURN o uminus-lit-imp*, *RETURN o uminus*) ∈
nat-lit-rel →$_f$ ⟨*nat-lit-rel*⟩*nres-rel*›
unfolding *bitXOR-1-if-mod-2 uminus-lit-imp-def*

by (*intro frefI nres-relI*) (*auto simp*: *uminus-lit-imp-def case-prod-beta p2rel-def br-def nat-lit-rel-def split*: *option.splits*, *presburger*)

1.6.2 State Conversion

Functions and Types:

More Operations

1.6.3 Code Generation

More Operations

definition *literals-to-update-wl-empty* :: ‹*nat twl-st-wl* ⇒ *bool*› **where**
‹*literals-to-update-wl-empty* = (λ(*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*). *Q* = {#})›

lemma *in-nat-list-rel-list-all2-in-set-iff*:
‹(*a*, *aa*) ∈ *nat-lit-rel* ⟹
list-all2 (λ*x x'*. (*x*, *x'*) ∈ *nat-lit-rel*) *b ba* ⟹
a ∈ *set b* ⟷ *aa* ∈ *set ba*›
apply (*subgoal-tac* ‹*length b* = *length ba*›)
subgoal
apply (*rotate-tac 2*)
apply (*induction b ba rule*: *list-induct2*)
apply (*solves simp*)
apply (*auto simp*: *p2rel-def nat-lit-rel-def br-def*, *presburger*)[]
done
subgoal using *list-all2-lengthD* **by** *auto*
done

definition *is-decided-wl* **where**
‹*is-decided-wl L* ⟷ *snd L* = *None*›

definition *get-maximum-level-remove* **where**
‹*get-maximum-level-remove M D L* = *get-maximum-level M* (*remove1-mset L D*)›

lemma *in-list-all2-ex-in*: ‹*a* ∈ *set xs* ⟹ *list-all2 R xs ys* ⟹ ∃ *b* ∈ *set ys*. *R a b*›
apply (*subgoal-tac* ‹*length xs* = *length ys*›)
apply (*rotate-tac 2*)
apply (*induction xs ys rule*: *list-induct2*)
apply ((*solves auto*)+)[2]
using *list-all2-lengthD* **by** *blast*

definition *find-decomp-wl-imp* :: ‹(*nat*, *nat*) *ann-lits* ⇒ *nat clause* ⇒ *nat literal* ⇒ (*nat*, *nat*) *ann-lits nres*› **where**
‹*find-decomp-wl-imp* = (λ*M*$_0$ *D L*. *do* {
let lev = *get-maximum-level M*$_0$ (*remove1-mset* (−*L*) *D*);
let k = *count-decided M*$_0$;
(-, *M*) ←
WHILE$_T$λ(*j*, *M*). *j* = *count-decided M* ∧ *j* ≥ *lev* ∧ (*M* = [] ⟶ *j* = *lev*) ∧ (∃ *M'*. *M*$_0$ = *M'* @ *M* ∧ (*j* =
(λ(*j*, *M*). *j* > *lev*)
(λ(*j*, *M*). *do* {
ASSERT(*M* ≠ []);
if is-decided (*hd M*)
then RETURN (*j*−*1*, *tl M*)
else RETURN (*j*, *tl M*)}
)

```
      (k, M_0);
    RETURN M
  })›

lemma ex-decomp-get-ann-decomposition-iff:
  ‹(∃ M2. (Decided K # M1, M2) ∈ set (get-all-ann-decomposition M)) ⟷
    (∃ M2. M = M2 @ Decided K # M1)›
  using get-all-ann-decomposition-ex by fastforce

lemma count-decided-tl-if:
  ‹M ≠ [] ⟹ count-decided (tl M) = (if is-decided (hd M) then count-decided M − 1 else count-decided
M)›
  by (cases M) auto

lemma count-decided-butlast:
  ‹count-decided (butlast xs) = (if is-decided (last xs) then count-decided xs − 1 else count-decided xs)›
  by (cases xs rule: rev-cases) (auto simp: count-decided-def)

definition find-decomp-wl' where
  ‹find-decomp-wl' =
    (λ(M::(nat, nat) ann-lits) (D::nat clause) (L::nat literal).
      SPEC(λM1. ∃ K M2. (Decided K # M1, M2) ∈ set (get-all-ann-decomposition M) ∧
        get-level M K = get-maximum-level M (D − {#−L#}) + 1))›

definition get-conflict-wl-is-None :: ‹nat twl-st-wl ⇒ bool› where
  ‹get-conflict-wl-is-None = (λ(M, N, D, NE, UE, Q, W). is-None D)›

lemma get-conflict-wl-is-None: ‹get-conflict-wl S = None ⟷ get-conflict-wl-is-None S›
  by (cases S) (auto simp: get-conflict-wl-is-None-def split: option.splits)

lemma hd-decided-count-decided-ge-1:
  ‹x ≠ [] ⟹ is-decided (hd x) ⟹ Suc 0 ≤ count-decided x›
  by (cases x) auto

definition (in −) find-decomp-wl-imp' :: ‹(nat, nat) ann-lits ⇒ nat clause-l list ⇒ nat ⇒
    nat clause ⇒ nat clauses ⇒ nat clauses ⇒ nat lit-queue-wl ⇒
    (nat literal ⇒ nat watched) ⇒ - ⇒ (nat, nat) ann-lits nres› where
  ‹find-decomp-wl-imp' = (λM N U D NE UE W Q L. find-decomp-wl-imp M D L)›

definition is-decided-hd-trail-wl where
  ‹is-decided-hd-trail-wl S = is-decided (hd (get-trail-wl S))›

definition is-decided-hd-trail-wll :: ‹nat twl-st-wl ⇒ bool nres› where
  ‹is-decided-hd-trail-wll = (λ(M, N, D, NE, UE, Q, W).
    RETURN (is-decided (hd M))
  )›

lemma Propagated-eq-ann-lit-of-pair-iff:
  ‹Propagated x21 x22 = ann-lit-of-pair (a, b) ⟷ x21 = a ∧ b = Some x22›
  by (cases b) auto

lemma set-mset-all-lits-of-mm-atms-of-ms-iff:
  ‹set-mset (all-lits-of-mm A) = set-mset (𝓛_all 𝒜) ⟷ atms-of-ms (set-mset A) = atms-of (𝓛_all 𝒜)›
  by (force simp add:  atms-of-s-def in-all-lits-of-mm-ain-atms-of-iff atms-of-ms-def
     atms-of-𝓛_all-𝒜_in atms-of-def atm-of-eq-atm-of uminus-𝒜_in-iff
      eq-commute[of ‹set-mset (all-lits-of-mm -)› ‹set-mset (𝓛_all -)›]
```

dest: *multi-member-split*)

1.7 Polarities

type-synonym *tri-bool* = ‹*bool option*›

definition *UNSET* :: ‹*tri-bool*› **where**
[*simp*]: ‹*UNSET* = *None*›

definition *SET-FALSE* :: ‹*tri-bool*› **where**
[*simp*]: ‹*SET-FALSE* = *Some False*›

definition *SET-TRUE* :: ‹*tri-bool*› **where**
[*simp*]: ‹*SET-TRUE* = *Some True*›

definition (**in** −) *tri-bool-eq* :: ‹*tri-bool* ⇒ *tri-bool* ⇒ *bool*› **where**
‹*tri-bool-eq* = (=)›
end
theory *IsaSAT-Literals-LLVM*
imports *WB-More-Word IsaSAT-Literals Watched-Literals.WB-More-IICF-LLVM*
More-Sepref.WB-More-Sepref-LLVM
begin

hide-const (**open**) *NEMonad.RETURN*

lemma *aal-assn-boundsD'*:
assumes *A*: ‹*rdomp* (*aal-assn'* *TYPE*('*l*::*len2*) *TYPE*('*ll*::*len2*) *A*) *xss*› **and** ‹*i* < *length xss*›
shows ‹*length* (*xss* ! *i*) < *max-snat LENGTH*('*ll*)›
using *aal-assn-boundsD-aux1*[*OF A*] *assms*
by *auto*

abbreviation ‹*word32-rel* ≡ *word-rel* :: (*32 word* × -) *set*›
abbreviation ‹*word64-rel* ≡ *word-rel* :: (*64 word* × -) *set*›
abbreviation ‹*word32-assn* ≡ *word-assn* :: *32 word* ⇒ -›
abbreviation ‹*word64-assn* ≡ *word-assn* :: *64 word* ⇒ -›

abbreviation *snat64-assn* :: ‹*nat* ⇒ *64 word* ⇒ -› **where** ‹*snat64-assn* ≡ *snat-assn*›
abbreviation *snat32-assn* :: ‹*nat* ⇒ *32 word* ⇒ -› **where** ‹*snat32-assn* ≡ *snat-assn*›
abbreviation *unat64-assn* :: ‹*nat* ⇒ *64 word* ⇒ -› **where** ‹*unat64-assn* ≡ *unat-assn*›

abbreviation *unat32-assn* :: ‹*nat* ⇒ *32 word* ⇒ -› **where** ‹*unat32-assn* ≡ *unat-assn*›

lemma *RETURN-comp-5-10-hnr-post*[*to-hnr-post*]:
‹(*RETURN ooooo f5*)abcd$e = *RETURN*$(*f5*abcd$e)›
‹(*RETURN oooooo f6*)abcdef = *RETURN*$(*f6*$abcde$f)›
‹(*RETURN ooooooo f7*)abcdef$g = *RETURN*$(*f7*abcdef$g)›
‹(*RETURN oooooooo f8*)abcdefgh = *RETURN*$(*f8*$abcdefg$h)›
‹(*RETURN ooooooooo f9*)abcdefgh$i = *RETURN*$(*f9*abcdefgh$i)›
‹(*RETURN oooooooooo f10*)abcdefghij = *RETURN*$(*f10*$abcdefghi$j)›
‹(*RETURN* o_{11} *f11*)abcdefghij$k = *RETURN*$(*f11*abcdefghij$k)›
‹(*RETURN* o_{12} *f12*)abcdefghijkl = *RETURN*$(*f12*$abcdefghijk$l)›
‹(*RETURN* o_{13} *f13*)abcdefghijkl$m = *RETURN*$(*f13*abcdefghijkl$m)›
‹(*RETURN* o_{14} *f14*)abcdefghijklmn = *RETURN*$(*f14*$abcdefghijklm$n)›

‹(RETURN o_{15} f15)abcdefghijklmn$xo = RETURN$(f15abcdefghijklmn$xo)›
‹(RETURN o_{16} f16)abcdefghijklmnxop = RETURN$(f16$abcdefghijklm$n$xo$p)›
‹(RETURN o_{17} f17)abcdefghijklmnxop$q = RETURN$(f17abcdefghijklmnxop$
‹(RETURN o_{18} f18)abcdefghijklmnxopqr = RETURN$(f18$abcdefghijklm$n$xo$
‹(RETURN o_{19} f19)abcdefghijklmnxopqr$s = RETURN$(f19abcdefghijklmn$x
by *simp-all*

method *synthesize-free* =
(*rule free-thms sepref-frame-free-rules*)+

definition [*simp,llvm-inline*]: ‹*case-prod-open* ≡ *case-prod*›
lemmas *fold-case-prod-open* = *case-prod-open-def*[*symmetric*]

lemma *case-prod-open-arity*[*sepref-monadify-arity*]:
‹*case-prod-open* ≡ λ_2*fp p. SP case-prod-open*$($\lambda_2$*a b. fp*$*a*$*b*)$*p*›
by (*simp-all only*: *SP-def APP-def PROTECT2-def RCALL-def*)

lemma *case-prod-open-comb*[*sepref-monadify-comb*]:
‹$\bigwedge$*fp p. case-prod-open*$*fp*$*p* ≡ *Refine-Basic.bind*$(*EVAL*$*p*)$($\lambda_2$*p.* (*SP case-prod-open*$*fp*$*p*))›
by (*simp-all*)

lemma *case-prod-open-plain-comb*[*sepref-monadify-comb*]:
EVAL$(*case-prod-open*$(λ_2*a b. fp a b*)$*p*) ≡
Refine-Basic.bind$(*EVAL*$*p*)$($\lambda_2$*p. case-prod-open*$(λ_2*a b. EVAL*$(*fp a b*))$*p*)
apply (*rule eq-reflection*, *simp split*: *list.split prod.split option.split*)+
done

lemma *hn-case-prod-open'*[*sepref-comb-rules*]:
assumes *FR*: ‹Γ ⊢ *hn-ctxt* (*prod-assn P1 P2*) *p' p* ** Γ *1*›
assumes *Pair*: $\bigwedge$*a1 a2 a1' a2'*. ⟦*p'*=(*a1'*,*a2'*)⟧
⟹ *hn-refine* (*hn-ctxt P1 a1' a1* ∧* *hn-ctxt P2 a2' a2* ∧* Γ *1*) (*f a1 a2*)
(Γ *2 a1 a2 a1' a2'*) *R* (*CP a1 a2*) (*f' a1' a2'*)
assumes *FR2*: ‹$\bigwedge$*a1 a2 a1' a2'*. Γ *2 a1 a2 a1' a2'* ⊢ *hn-ctxt P1' a1' a1* ** *hn-ctxt P2' a2' a2* ** Γ *1'*›
shows ‹*hn-refine* Γ (*case-prod-open f p*) (*hn-ctxt* (*prod-assn P1' P2'*) *p' p* ** Γ *1'*)
R (*CP-SPLIT CP p*) (*case-prod-open*$($\lambda_2$*a b. f' a b*)$*p'*)› (**is** ‹*?G* Γ›)
unfolding *case-prod-open-def*
unfolding *autoref-tag-defs PROTECT2-def*
apply1 (*rule hn-refine-cons-pre*[*OF FR*])
apply1 (*cases p*; *cases p'*; *simp add*: *prod-assn-pair-conv*[*THEN prod-assn-ctxt*])
unfolding *CP-SPLIT-def prod.simps*
apply (*rule hn-refine-cons*[*OF - Pair - entails-refl*])
applyS (*simp add*: *hn-ctxt-def*)
applyS *simp* **using** *FR2*
by (*simp add*: *hn-ctxt-def*)

lemma *ho-prod-open-move*[*sepref-preproc*]: ‹*case-prod-open* (λ*a b x. f x a b*) = (λ*p x. case-prod-open* (*f x*) *p*)›
by (*auto*)

lemma *op-list-list-upd-alt-def*: ‹*op-list-list-upd xss i j x* = *xss*[*i* := (*xss* ! *i*)[*j* := *x*]]›
unfolding *op-list-list-upd-def* **by** *auto*

definition ‹*tuple4 a b c d* ≡ (*a*,*b*,*c*,*d*)›

definition ‹*tuple7 a b c d e f g* ≡ *tuple4 a b c* (*tuple4 d e f g*)›
definition ‹*tuple13 a b c d e f g h i j k l m* ≡ (*tuple7 a b c d e f* (*tuple7 g h i j k l m*))›

lemmas *fold-tuples* = *tuple4-def*[*symmetric*] *tuple7-def*[*symmetric*] *tuple13-def*[*symmetric*]

sepref-register *tuple4 tuple7 tuple13*

sepref-def *tuple4-impl* [*llvm-inline*] **is** ‹*uncurry3* (*RETURN oooo tuple4*)› ::
‹$A1^d *_a A2^d *_a A3^d *_a A4^d \rightarrow_a A1 \times_a A2 \times_a A3 \times_a A4$›
unfolding *tuple4-def* **by** *sepref*

sepref-def *tuple7-impl* [*llvm-inline*] **is** ‹*uncurry6* (*RETURN ooooooo tuple7*)› ::
‹$A1^d *_a A2^d *_a A3^d *_a A4^d *_a A5^d *_a A6^d *_a A7^d \rightarrow_a A1 \times_a A2 \times_a A3 \times_a A4 \times_a A5 \times_a A6 \times_a A7$›
unfolding *tuple7-def* **by** *sepref*

sepref-def *tuple13-impl* [*llvm-inline*] **is** ‹*uncurry12* (*RETURN* o_{13} *tuple13*)› ::
$A1^d *_a A2^d *_a A3^d *_a A4^d *_a A5^d *_a A6^d *_a A7^d *_a A8^d *_a A9^d *_a A10^d *_a A11^d *_a A12^d *_a A13^d$
$\rightarrow_a A1 \times_a A2 \times_a A3 \times_a A4 \times_a A5 \times_a A6 \times_a A7 \times_a A8 \times_a A9 \times_a A10 \times_a A11 \times_a A12 \times_a A13$
unfolding *tuple13-def* **by** *sepref*

lemmas *fold-tuple-optimizations* = *fold-tuples fold-case-prod-open*

lemma *snat64-max-refine*[*sepref-import-param*]: ‹(*0x7FFFFFFFFFFFFFFF*, *snat64-max*)∈*snat-rel*′ *TYPE*(*64*)›
apply (*auto simp*: *snat-rel-def snat.rel-def in-br-conv snat64-max-def snat-invar-def*)
apply (*auto simp*: *snat-def*)
done

lemma *snat32-max-refine*[*sepref-import-param*]: ‹(*0x7FFFFFFF*, *snat32-max*)∈*snat-rel*′ *TYPE*(*32*)›
apply (*auto simp*: *snat-rel-def snat.rel-def in-br-conv snat32-max-def snat-invar-def*)
apply (*auto simp*: *snat-def*)
done

lemma *unat64-max-refine*[*sepref-import-param*]: ‹(*0xFFFFFFFFFFFFFFFF*, *unat64-max*)∈*unat-rel*′ *TYPE*(*64*)›
apply (*auto simp*: *unat-rel-def unat.rel-def in-br-conv unat64-max-def*)
done

lemma *unat32-max-refine*[*sepref-import-param*]: ‹(*0xFFFFFFFF*, *unat32-max*)∈*unat-rel*′ *TYPE*(*32*)›
apply (*auto simp*: *unat-rel-def unat.rel-def in-br-conv unat32-max-def*)
done

abbreviation ‹*uint32-nat-assn* ≡ *unat-assn*′ *TYPE*(*32*)›
abbreviation ‹*uint64-nat-assn* ≡ *unat-assn*′ *TYPE*(*64*)›

abbreviation ‹*sint32-nat-assn* ≡ *snat-assn*′ *TYPE*(*32*)›
abbreviation ‹*sint64-nat-assn* ≡ *snat-assn*′ *TYPE*(*64*)›

It is critical for performance of auto to calculate the power instead of letting auto do it every time.

lemmas [*simplified*, *sepref-bounds-simps*] =
unat32-max-def snat32-max-def
unat64-max-def snat64-max-def

lemma *is-up'-32-64*[*simp*,*intro*!]: ‹*is-up' UCAST*(*32* → *64*)› **by** (*simp add*: *is-up'*)
lemma *is-down'-64-32*[*simp*,*intro*!]: ‹*is-down' UCAST*(*64* → *32*)› **by** (*simp add*: *is-down'*)

lemma *ins-idx-upcast64*:
‹*l*[*i*:=*y*] = *op-list-set l* (*op-unat-snat-upcast TYPE*(*64*) *i*) *y*›
‹*l*!*i* = *op-list-get l* (*op-unat-snat-upcast TYPE*(*64*) *i*)›
by *simp-all*

type-synonym *'a array-list32* = ‹(*'a*,*32*)*array-list*›
type-synonym *'a array-list64* = ‹(*'a*,*64*)*array-list*›

abbreviation ‹*arl32-assn* ≡ *al-assn' TYPE*(*32*)›
abbreviation ‹*arl64-assn* ≡ *al-assn' TYPE*(*64*)›

type-synonym *'a larray32* = ‹(*'a*,*32*) *larray*›
type-synonym *'a larray64* = ‹(*'a*,*64*) *larray*›

abbreviation ‹*larray32-assn* ≡ *larray-assn' TYPE*(*32*)›
abbreviation ‹*larray64-assn* ≡ *larray-assn' TYPE*(*64*)›

definition ‹*unat-lit-rel* == *unat-rel' TYPE*(*32*) *O nat-lit-rel*›
lemmas [*fcomp-norm-unfold*] = *unat-lit-rel-def*[*symmetric*]

abbreviation *unat-lit-assn* :: ‹*nat literal* ⇒ *32 word* ⇒ *assn*› **where**
‹*unat-lit-assn* ≡ *pure unat-lit-rel*›

1.7.1 Atom-Of

type-synonym *atom-assn* = ‹*32 word*›

definition ‹*atom-rel* ≡ *b-rel* (*unat-rel' TYPE*(*32*)) (λ*x*. *x*<*2^31*)›
abbreviation ‹*atom-assn* ≡ *pure atom-rel*›

lemma *atom-rel-alt*: ‹*atom-rel* = *unat-rel' TYPE*(*32*) *O nbn-rel* (*2^31*)›
by (*auto simp*: *atom-rel-def*)

interpretation *atom*: *dflt-pure-option-private* ‹*2^32*−*1*› *atom-assn* ‹*ll-icmp-eq* (*2^32*−*1*)›
apply *unfold-locales*
subgoal
unfolding *atom-rel-def*
apply (*simp add*: *pure-def fun-eq-iff pred-lift-extract-simps*)
apply (*auto simp*: *unat-rel-def unat.rel-def in-br-conv unat-minus-one-word*)
done
subgoal proof *goal-cases*
case *1*
interpret *llvm-prim-arith-setup* .
show *?case* **unfolding** *bool.assn-def* **by** *vcg'*
qed

subgoal by *simp*
done

lemma *atm-of-refine*: ‹(λx. x div 2 , atm-of) ∈ nat-lit-rel → nat-rel›
by (*auto simp*: *nat-lit-rel-def in-br-conv*)

sepref-def *atm-of-impl* **is** [] ‹*RETURN o* (λx::*nat*. x div 2)›
:: ‹*uint32-nat-assn*^{k} →$_a$ *atom-assn*›
unfolding *atom-rel-def b-assn-pure-conv*[*symmetric*]
apply (*rule hfref-bassn-resI*)
subgoal by *sepref-bounds*
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

lemmas [*sepref-fr-rules*] = *atm-of-impl.refine*[*FCOMP atm-of-refine*]

definition *Pos-rel* :: ‹*nat* ⇒ *nat*› **where**
[*simp*]: ‹*Pos-rel n* = *2* ∗ *n*›

lemma *Pos-refine-aux*: ‹(*Pos-rel*,*Pos*)∈*nat-rel* → *nat-lit-rel*›
by (*auto simp*: *nat-lit-rel-def in-br-conv split*: *if-splits*)

lemma *Neg-refine-aux*: ‹(λx. *2*∗x + *1*,*Neg*)∈*nat-rel* → *nat-lit-rel*›
by (*auto simp*: *nat-lit-rel-def in-br-conv split*: *if-splits*)

sepref-def *Pos-impl* **is** [] ‹*RETURN o Pos-rel*› :: ‹*atom-assn*^{d} →$_a$ *uint32-nat-assn*›
unfolding *atom-rel-def Pos-rel-def*
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

sepref-def *Neg-impl* **is** [] ‹*RETURN o* (λx. *2*∗x+*1*)› :: ‹*atom-assn*^{d} →$_a$ *uint32-nat-assn*›
unfolding *atom-rel-def*
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

lemmas [*sepref-fr-rules*] =
Pos-impl.refine[*FCOMP Pos-refine-aux*]
Neg-impl.refine[*FCOMP Neg-refine-aux*]

sepref-def *atom-eq-impl* **is** ‹*uncurry* (*RETURN oo* (=))› :: ‹*atom-assn*^{d} ∗$_a$ *atom-assn*^{d} →$_a$ *bool1-assn*›
unfolding *atom-rel-def*
by *sepref*

definition *value-of-atm* :: ‹*nat* ⇒ *nat*› **where**
[*simp*]: ‹*value-of-atm A* = *A*›

lemma *value-of-atm-rel*: ‹(λx. x, *value-of-atm*) ∈ *nat-rel* → *nat-rel*›
by (*auto*)

sepref-def *value-of-atm-impl*
is [] ‹*RETURN o* (λx. x)›
:: ‹*atom-assn*^{d} →$_a$ *unat-assn*′ *TYPE*(*32*)›

unfolding *value-of-atm-def atom-rel-def*
by *sepref*

lemmas [*sepref-fr-rules*] = *value-of-atm-impl.refine*[*FCOMP value-of-atm-rel*]

definition *index-of-atm* :: ‹*nat* ⇒ *nat*› **where**
[*simp*]: ‹*index-of-atm A* = *value-of-atm A*›

lemma *index-of-atm-rel*: ‹(λ*x*. *value-of-atm x*, *index-of-atm*) ∈ *nat-rel* → *nat-rel*›
by (*auto*)

sepref-def *index-of-atm-impl*
is [] ‹*RETURN o* (λ*x*. *value-of-atm x*)›
:: ‹*atom-assn*^{d} →$_a$ *snat-assn'* *TYPE*(*64*)›
unfolding *index-of-atm-def*
apply (*rewrite at* ‹-› *eta-expand*)
apply (*subst annot-unat-snat-upcast*[**where** *'l*=*64*])
by *sepref*

lemmas [*sepref-fr-rules*] = *index-of-atm-impl.refine*[*FCOMP index-of-atm-rel*]

lemma *annot-index-of-atm*: ‹*xs* ! *x* = *xs* ! *index-of-atm x*›
‹*xs* [*x* := *a*] = *xs* [*index-of-atm x* := *a*]›
by *auto*

definition *index-atm-of* **where**
[*simp*]: ‹*index-atm-of L* = *index-of-atm* (*atm-of L*)›

context fixes *x y* :: *nat* **assumes** ‹*NO-MATCH* (*index-of-atm y*) *x*› **begin**
lemmas *annot-index-of-atm'* = *annot-index-of-atm*[**where** *x*=*x*]
end

method-setup *annot-all-atm-idxs* = ‹*Scan.succeed* (*fn ctxt* => *SIMPLE-METHOD'*
let
val ctxt = *put-simpset HOL-basic-ss ctxt*
val ctxt = *ctxt addsimps* @{*thms annot-index-of-atm'*}
val ctxt = *ctxt addsimprocs* [@{*simproc NO-MATCH*}]
in
simp-tac ctxt
end
)›

lemma *annot-index-atm-of*[*def-pat-rules*]:
‹*nth*$*xs*$(*atm-of*$*x*) ≡ *nth*$*xs*$(*index-atm-of*$*x*)›
‹*list-update*$*xs*$(*atm-of*$*x*)$*a* ≡ *list-update*$*xs*$(*index-atm-of*$*x*)$*a*›
by *auto*

sepref-def *index-atm-of-impl*
is ‹*RETURN o index-atm-of*›
:: ‹*unat-lit-assn*^{d} →$_a$ *snat-assn'* *TYPE*(*64*)›
unfolding *index-atm-of-def*
by *sepref*

lemma *nat-of-lit-refine-aux*: ‹((λx. x), *nat-of-lit*) ∈ *nat-lit-rel* → *nat-rel*›
by (*auto simp*: *nat-lit-rel-def in-br-conv*)

sepref-def *nat-of-lit-rel-impl* **is** [] ‹*RETURN o* (λx::*nat*. x)› :: ‹*uint32-nat-assn*^{k} →$_a$ *sint64-nat-assn*›
apply (*rewrite annot-unat-snat-upcast*[**where** *'l=64*])
by *sepref*
lemmas [*sepref-fr-rules*] = *nat-of-lit-rel-impl.refine*[*FCOMP nat-of-lit-refine-aux*]

lemma *uminus-refine-aux*: ‹(λx. x *XOR* 1, *uminus*) ∈ *nat-lit-rel* → *nat-lit-rel*›
apply (*auto simp*: *nat-lit-rel-def in-br-conv bitXOR-1-if-mod-2*[*simplified*])
subgoal by *linarith*
subgoal by (*metis dvd-minus-mod even-Suc-div-two odd-Suc-minus-one*)
done

sepref-def *uminus-impl* **is** [] ‹*RETURN o* (λx::*nat*. x *XOR* 1)› :: ‹*uint32-nat-assn*^{k} →$_a$ *uint32-nat-assn*›
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

lemmas [*sepref-fr-rules*] = *uminus-impl.refine*[*FCOMP uminus-refine-aux*]

lemma *lit-eq-refine-aux*: ‹((=), (=)) ∈ *nat-lit-rel* → *nat-lit-rel* → *bool-rel*›
by (*auto simp*: *nat-lit-rel-def in-br-conv split*: *if-splits*; *auto?*; *presburger*)

sepref-def *lit-eq-impl* **is** [] ‹*uncurry* (*RETURN oo* (=))› :: ‹*uint32-nat-assn*^{k} ×$_a$ *uint32-nat-assn*^{k} →$_a$ *bool1-assn*›
by *sepref*

lemmas [*sepref-fr-rules*] = *lit-eq-impl.refine*[*FCOMP lit-eq-refine-aux*]

lemma *is-pos-refine-aux*: ‹(λx. x *AND* 1 = 0, *is-pos*) ∈ *nat-lit-rel* → *bool-rel*›
by (*auto simp*: *nat-lit-rel-def in-br-conv bitAND-1-mod-2*[*simplified*] *split*: *if-splits*)

sepref-def *is-pos-impl* **is** [] ‹*RETURN o* (λx. x *AND* 1 = 0)› :: ‹*uint32-nat-assn*^{k} →$_a$ *bool1-assn*›
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

lemmas [*sepref-fr-rules*] = *is-pos-impl.refine*[*FCOMP is-pos-refine-aux*]

sepref-decl-op *nat-lit-eq*: ‹(=) :: *nat literal* ⇒ - ⇒ -› ::
‹(*Id* :: (*nat literal* × -) *set*) → (*Id* :: (*nat literal* × -) *set*) → *bool-rel*› .

sepref-def *nat-lit-eq-impl*
is [] ‹*uncurry* (*RETURN oo* (λx y. x = y))›
:: ‹*uint32-nat-assn*^{k} ×$_a$ *uint32-nat-assn*^{k} →$_a$ *bool1-assn*›
by *sepref*

lemma *nat-lit-rel*: ‹((=), *op-nat-lit-eq*) ∈ *nat-lit-rel* → *nat-lit-rel* → *bool-rel*›
by (*auto simp*: *nat-lit-rel-def br-def split*: *if-splits*; *presburger*)

sepref-register ‹(=) :: *nat literal* ⇒ - ⇒ -›
declare *nat-lit-eq-impl.refine*[*FCOMP nat-lit-rel*, *sepref-fr-rules*]

context

fixes *l-dummy* :: ‹*'l*::*len2 itself*›
fixes *ll-dummy* :: ‹*'ll*::*len2 itself*›
fixes *L LL AA*
defines [*simp*]: ‹*L* ≡ (*LENGTH* (*'l*))›
defines [*simp*]: ‹*LL* ≡ (*LENGTH* (*'ll*))›
defines [*simp*]: ‹*AA* ≡ *raw-aal-assn TYPE*(*'l*::*len2*) *TYPE*(*'ll*::*len2*)›
begin
private lemma *n-unf*: ‹*hr-comp AA* (⟨⟨*the-pure A*⟩*list-rel*⟩*list-rel*) = *aal-assn A*› **unfolding** *aal-assn-def AA-def* **..**

context
notes [*fcomp-norm-unfold*] = *n-unf*
begin

lemma *aal-assn-free*[*sepref-frame-free-rules*]: ‹*MK-FREE AA aal-free*›
apply *rule* **by** *vcg*
sepref-decl-op *list-list-free*: ‹λ-::- *list list*. ()› :: ‹⟨⟨*A*⟩*list-rel*⟩*list-rel* → *unit-rel*› **.**

lemma *hn-aal-free-raw*: ‹(*aal-free*,*RETURN o op-list-list-free*) ∈ *AA*^{d} →$_a$ *unit-assn*›
by *sepref-to-hoare vcg*

sepref-decl-impl *aal-free*: *hn-aal-free-raw*
.

lemmas *array-mk-free*[*sepref-frame-free-rules*] = *hn-MK-FREEI*[*OF aal-free-hnr*]
end
end

lemma *of-nat-snat*:
‹(*id*,*of-nat*) ∈ *snat-rel'* *TYPE*(*'a*::*len2*) → *word-rel*›
by (*auto simp*: *snat-rel-def snat.rel-def in-br-conv snat-eq-unat*)

lemma *of-nat-unat*:
‹(*id*,*of-nat*) ∈ *unat-rel'* *TYPE*(*'a*::*len2*) → *word-rel*›
by (*auto simp*: *unat-rel-def unat.rel-def in-br-conv snat-eq-unat*)

type-synonym *tri-bool-assn* = ‹*8 word*›

definition ‹*tri-bool-rel-aux* ≡ { (*0*::*nat*,*None*), (*2*,*Some True*), (*3*,*Some False*) }›
definition ‹*tri-bool-rel* ≡ *unat-rel'* *TYPE*(*8*) *O tri-bool-rel-aux*›
abbreviation ‹*tri-bool-assn* ≡ *pure tri-bool-rel*›
lemmas [*fcomp-norm-unfold*] = *tri-bool-rel-def*[*symmetric*]

lemma *tri-bool-UNSET-refine-aux*: ‹(*0*,*UNSET*)∈*tri-bool-rel-aux*›
and *tri-bool-SET-TRUE-refine-aux*: ‹(*2*,*SET-TRUE*)∈*tri-bool-rel-aux*›
and *tri-bool-SET-FALSE-refine-aux*: ‹(*3*,*SET-FALSE*)∈*tri-bool-rel-aux*›
and *tri-bool-eq-refine-aux*: ‹((=),*tri-bool-eq*) ∈ *tri-bool-rel-aux*→*tri-bool-rel-aux*→*bool-rel*›
by (*auto simp*: *tri-bool-rel-aux-def tri-bool-eq-def*)

sepref-def *tri-bool-UNSET-impl* **is** [] ‹*uncurry0* (*RETURN 0*)› :: ‹*unit-assn*^{k} →$_a$ *unat-assn'* *TYPE*(*8*)›
apply (*annot-unat-const* ‹*TYPE*(*8*)›)
by *sepref*

sepref-def *tri-bool-SET-TRUE-impl* **is** [] ‹*uncurry0* (*RETURN 2*)› :: ‹*unit-assn*^{k} →$_a$ *unat-assn'* *TYPE*(*8*)›

apply (*annot-unat-const* ‹*TYPE*(*8*)›)
by *sepref*

sepref-def *tri-bool-SET-FALSE-impl* **is** [] ‹*uncurry0* (*RETURN 3*)› :: ‹*unit-assn*^{k} →$_a$ *unat-assn*′ *TYPE*(*8*)›
apply (*annot-unat-const* ‹*TYPE*(*8*)›)
by *sepref*

sepref-def *tri-bool-eq-impl* [*llvm-inline*] **is** [] ‹*uncurry* (*RETURN oo* (=))› :: ‹(*unat-assn*′ *TYPE*(*8*))^k *$_a$ (*unat-assn*′ *TYPE*(*8*))^k →$_a$ *bool1-assn*›
by *sepref*

lemmas [*sepref-fr-rules*] =
tri-bool-UNSET-impl.refine[*FCOMP tri-bool-UNSET-refine-aux*]
tri-bool-SET-TRUE-impl.refine[*FCOMP tri-bool-SET-TRUE-refine-aux*]
tri-bool-SET-FALSE-impl.refine[*FCOMP tri-bool-SET-FALSE-refine-aux*]
tri-bool-eq-impl.refine[*FCOMP tri-bool-eq-refine-aux*]
hide-const (**open**) *tuple4 tuple7*
end
theory *Pairing-Heaps-Impl-LLVM*
imports *Pairing-Heap-LLVM.Pairing-Heaps-Impl IsaSAT-Literals-LLVM*
begin

type-synonym *hp-assn* = ‹*32 word ptr* × *32 word ptr* × *32 word ptr* × *32 word ptr* × *64 word ptr* × *32 word*›

definition *hp-assn* :: ‹- ⇒ *hp-assn* ⇒ *assn*› **where**
‹*hp-assn* ≡ (*IICF-Array.array-assn atom.option-assn* ×$_a$
IICF-Array.array-assn atom.option-assn ×$_a$
IICF-Array.array-assn atom.option-assn ×$_a$
IICF-Array.array-assn uint64-nat-assn ×$_a$ *atom.option-assn*)›

sepref-def *mop-hp-read-prev-imp-code*
is ‹*uncurry mop-hp-read-prev-imp*›
:: ‹*atom-assn*^{k} *$_a$ *hp-assn*^{k} →$_a$ *atom.option-assn*›
unfolding *mop-hp-read-prev-imp-def hp-assn-def*
apply (*rewrite at* ‹- ! ⌑› *value-of-atm-def*[*symmetric*])
apply (*rewrite* **in** ‹- ! ⌑› *annot-unat-snat-upcast*[**where** *′l*=‹*64*›])
by *sepref*

sepref-def *mop-hp-read-nxt-imp-code*
is ‹*uncurry mop-hp-read-nxt-imp*›
:: ‹*atom-assn*^{k} *$_a$ *hp-assn*^{k} →$_a$ *atom.option-assn*›
unfolding *mop-hp-read-nxt-imp-def hp-assn-def*
apply (*rewrite at* ‹- ! ⌑› *value-of-atm-def*[*symmetric*])
apply (*rewrite* **in** ‹- ! ⌑› *annot-unat-snat-upcast*[**where** *′l*=‹*64*›])
by *sepref*

sepref-def *mop-hp-read-parent-imp-code*
is ‹*uncurry mop-hp-read-parent-imp*›
:: ‹*atom-assn*^{k} *$_a$ *hp-assn*^{k} →$_a$ *atom.option-assn*›
unfolding *mop-hp-read-parent-imp-def hp-assn-def*
apply (*rewrite at* ‹- ! ⌑› *value-of-atm-def*[*symmetric*])
apply (*rewrite* **in** ‹- ! ⌑› *annot-unat-snat-upcast*[**where** *′l*=‹*64*›])
by *sepref*

sepref-def *mop-hp-read-child-imp-code*
is ‹*uncurry mop-hp-read-child-imp*›
:: ‹$atom\text{-}assn^k *_a hp\text{-}assn^k \rightarrow_a atom.option\text{-}assn$›
unfolding *mop-hp-read-child-imp-def hp-assn-def*
apply (*rewrite at* ‹-! ⌑› *value-of-atm-def*[*symmetric*])
apply (*rewrite* **in** ‹- ! ⌑› *annot-unat-snat-upcast*[**where** *'l*=‹*64*›])
by *sepref*

sepref-def *mop-hp-read-score-imp-code*
is ‹*uncurry mop-hp-read-score-imp*›
:: ‹$atom\text{-}assn^k *_a hp\text{-}assn^k \rightarrow_a uint64\text{-}nat\text{-}assn$›
unfolding *mop-hp-read-score-imp-def hp-assn-def*
apply (*rewrite at* ‹-! ⌑› *value-of-atm-def*[*symmetric*])
apply (*rewrite* **in** ‹- ! ⌑› *annot-unat-snat-upcast*[**where** *'l*=‹*64*›])
by *sepref*

lemma *source-node-impl-alt-def*:
‹*source-node-impl* = (λ(*prevs*, *nxts*, *parents*, *children*, *scores*,*i*). *i*)›
by (*auto intro*!: *ext*)

sepref-def *source-node-impl-code*
is ‹(*RETURN o source-node-impl*)›
:: ‹$hp\text{-}assn^k \rightarrow_a atom.option\text{-}assn$›
unfolding *source-node-impl-alt-def hp-assn-def*
by *sepref*

lemma *update-source-node-impl-alt-def*:
‹*update-source-node-impl* = (λ*i* (*prevs*, *nxts*, *parents*, *children*, *scores*,-). (*prevs*, *nxts*, *parents*, *children*, *scores*, *i*))›
by (*auto intro*!: *ext*)

sepref-def *update-source-node-impl-code*
is ‹*uncurry* (*RETURN oo update-source-node-impl*)›
:: ‹$atom.option\text{-}assn^k *_a hp\text{-}assn^d \rightarrow_a hp\text{-}assn$›
unfolding *update-source-node-impl-alt-def hp-assn-def*
by *sepref*

sepref-def *mop-hp-update-prev'-imp-code*
is ‹*uncurry2 mop-hp-update-prev'-imp*›
:: ‹$atom\text{-}assn^k *_a atom.option\text{-}assn^k *_a hp\text{-}assn^d \rightarrow_a hp\text{-}assn$›
unfolding *mop-hp-update-prev'-imp-def hp-assn-def*
apply (*rewrite at* ‹-[⌑:=-]› *value-of-atm-def*[*symmetric*])
apply (*rewrite* **in** ‹- [⌑:=-]› *annot-unat-snat-upcast*[**where** *'l*=‹*64*›])
by *sepref*

sepref-def *mop-hp-update-child'-imp-code*
is ‹*uncurry2 mop-hp-update-child'-imp*›
:: ‹$atom\text{-}assn^k *_a atom.option\text{-}assn^k *_a hp\text{-}assn^d \rightarrow_a hp\text{-}assn$›
unfolding *mop-hp-update-child'-imp-def hp-assn-def*
apply (*rewrite at* ‹-[⌑:=-]› *value-of-atm-def*[*symmetric*])
apply (*rewrite* **in** ‹- [⌑:=-]› *annot-unat-snat-upcast*[**where** *'l*=‹*64*›])
by *sepref*

sepref-def *mop-hp-update-nxt'-imp-code*
is ‹*uncurry2 mop-hp-update-nxt'-imp*›
:: ‹$atom\text{-}assn^k *_a atom.option\text{-}assn^k *_a hp\text{-}assn^d \rightarrow_a hp\text{-}assn$›

unfolding *mop-hp-update-nxt'-imp-def hp-assn-def*
apply (*rewrite at* ‹-[⌑:=-]› *value-of-atm-def*[*symmetric*])
apply (*rewrite* **in** ‹- [⌑:=-]› *annot-unat-snat-upcast*[**where** *'l*=‹*64*›])
by *sepref*

sepref-def *mop-hp-update-parent'-imp-code*
is ‹*uncurry2 mop-hp-update-parent'-imp*›
:: ‹*atom-assn*^{k} $*_a$ *atom.option-assn*^{k} $*_a$ *hp-assn*^{d} $\to_a$ *hp-assn*›
unfolding *mop-hp-update-parent'-imp-def hp-assn-def*
apply (*rewrite at* ‹-[⌑:=-]› *value-of-atm-def*[*symmetric*])
apply (*rewrite* **in** ‹- [⌑:=-]› *annot-unat-snat-upcast*[**where** *'l*=‹*64*›])
by *sepref*

sepref-def *mop-hp-set-all-imp-code*
is ‹*uncurry6 mop-hp-set-all-imp*›
:: ‹*atom-assn*^{k} $*_a$ *atom.option-assn*^{k} $*_a$ *atom.option-assn*^{k} $*_a$ *atom.option-assn*^{k} $*_a$ *atom.option-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$ *hp-assn*^{d} $\to_a$ *hp-assn*›
unfolding *mop-hp-set-all-imp-def hp-assn-def*
apply (*rewrite at* ‹-[⌑:=-]› *value-of-atm-def*[*symmetric*])
apply (*rewrite* **in** ‹- [⌑:=-]› *annot-unat-snat-upcast*[**where** *'l*=‹*64*›])
apply (*rewrite at* ‹(-, -[⌑:=-], -)› *value-of-atm-def*[*symmetric*])
apply (*rewrite* **in** ‹(-, - [⌑:=-],-)› *annot-unat-snat-upcast*[**where** *'l*=‹*64*›])
apply (*rewrite at* ‹(-, -, -[⌑:=-], -)› *value-of-atm-def*[*symmetric*])
apply (*rewrite* **in** ‹(-, -, - [⌑:=-],-)› *annot-unat-snat-upcast*[**where** *'l*=‹*64*›])
apply (*rewrite at* ‹(-, -, -, -[⌑:=-], -)› *value-of-atm-def*[*symmetric*])
apply (*rewrite* **in** ‹(-, -, -, - [⌑:=-],-)› *annot-unat-snat-upcast*[**where** *'l*=‹*64*›])
apply (*rewrite at* ‹(-, -, -, -, -[⌑:=-], -)› *value-of-atm-def*[*symmetric*])
apply (*rewrite* **in** ‹(-, -, -, -, - [⌑:=-],-)› *annot-unat-snat-upcast*[**where** *'l*=‹*64*›])
by *sepref*

sepref-register *mop-hp-set-all-imp*
mop-hp-update-parent'-imp mop-hp-update-nxt'-imp mop-hp-update-child'-imp mop-hp-update-prev'-imp
mop-hp-read-score-imp mop-hp-read-nxt-imp mop-hp-read-prev-imp mop-hp-read-parent-imp mop-hp-read-child-imp
maybe-mop-hp-update-prev'-imp maybe-mop-hp-update-nxt'-imp maybe-mop-hp-update-child'-imp maybe-mop-hp-update-

sepref-def *mop-hp-insert-impl-code*
is ‹*uncurry2 mop-hp-insert-impl*›
:: ‹*atom-assn*^{k} $*_a$ *uint64-nat-assn*^{k} $*_a$ *hp-assn*^{d} $\to_a$ *hp-assn*›
unfolding *mop-hp-insert-impl-def*
atom.fold-option
by *sepref*

sepref-def *maybe-mop-hp-update-prev'-imp-code*
is ‹*uncurry2 maybe-mop-hp-update-prev'-imp*›
:: ‹*atom.option-assn*^{k} $*_a$ *atom.option-assn*^{k} $*_a$ *hp-assn*^{d} $\to_a$ *hp-assn*›
unfolding *maybe-mop-hp-update-prev'-imp-def*
atom.fold-option
by *sepref*

sepref-def *maybe-mop-hp-update-nxt'-imp-code*
is ‹*uncurry2 maybe-mop-hp-update-nxt'-imp*›
:: ‹*atom.option-assn*^{k} $*_a$ *atom.option-assn*^{k} $*_a$ *hp-assn*^{d} $\to_a$ *hp-assn*›
unfolding *maybe-mop-hp-update-nxt'-imp-def*
atom.fold-option
by *sepref*

sepref-def *maybe-mop-hp-update-child'-imp-code*
  is ‹*uncurry2 maybe-mop-hp-update-child'-imp*›
  :: ‹*atom.option-assn*^{k} $*_a$ *atom.option-assn*^{k} $*_a$ *hp-assn*^{d} $\to_a$ *hp-assn*›
  unfolding *maybe-mop-hp-update-child'-imp-def*
    atom.fold-option
  by *sepref*

sepref-def *maybe-mop-hp-update-parent'-imp-code*
  is ‹*uncurry2 maybe-mop-hp-update-parent'-imp*›
  :: ‹*atom.option-assn*^{k} $*_a$ *atom.option-assn*^{k} $*_a$ *hp-assn*^{d} $\to_a$ *hp-assn*›
  unfolding *maybe-mop-hp-update-parent'-imp-def*
    atom.fold-option
  by *sepref*

sepref-def *mop-hp-link-imp-impl*
  is ‹*uncurry2 mop-hp-link-imp*›
  :: ‹*atom-assn*^{k} $*_a$ *atom-assn*^{k} $*_a$ *hp-assn*^{d} $\to_a$ *hp-assn* $\times_a$ *atom-assn*›
  unfolding *mop-hp-link-imp-def*
    atom.fold-option
  by *sepref*

sepref-register *mop-hp-link-imp mop-vsids-pass*$_1$*-imp mop-vsids-pass*$_2$*-imp mop-merge-pairs-imp*
  mop-vsids-pop-min-impl mop-unroot-hp-tree

sepref-def *mop-vsids-pass*$_1$*-imp-code*
  is ‹*uncurry mop-vsids-pass*$_1$*-imp*›
  :: ‹*hp-assn*^{d} $*_a$ *atom-assn*^{k} $\to_a$ *hp-assn* $\times_a$ *atom-assn*›
  unfolding *mop-vsids-pass*$_1$*-imp-def*
    atom.fold-option
  by *sepref*

sepref-def *mop-vsids-pass*$_2$*-imp-code*
  is ‹*uncurry mop-vsids-pass*$_2$*-imp*›
  :: ‹*hp-assn*^{d} $*_a$ *atom-assn*^{k} $\to_a$ *hp-assn*›
  unfolding *mop-vsids-pass*$_2$*-imp-def*
    atom.fold-option
  by *sepref*

sepref-def *mop-merge-pairs-imp-code*
  is ‹*uncurry mop-merge-pairs-imp*›
  :: ‹*hp-assn*^{d} $*_a$ *atom-assn*^{k} $\to_a$ *hp-assn*›
  unfolding *mop-merge-pairs-imp-def*
  by *sepref*

sepref-def *mop-vsids-pop-min-impl-code*
  is *mop-vsids-pop-min-impl*
  :: ‹*hp-assn*^{d} $\to_a$ *atom.option-assn* $\times_a$ *hp-assn*›
  unfolding *mop-vsids-pop-min-impl-def*
    atom.fold-option
  by *sepref*

definition *mop-source-node-impl* **where**
  mop-source-node-impl = *RETURN o source-node-impl*
sepref-register *mop-source-node-impl*

sepref-def *mop-source-node-impl-code*
  is *mop-source-node-impl*
  :: ‹$hp\text{-}assn^k \rightarrow_a atom.option\text{-}assn$›
  unfolding *mop-source-node-impl-def*
  by *sepref*

sepref-register
  source-node-impl :: (nat,nat)*pairing-heaps-imp* ⇒ -

hide-const (**open**) *NEMonad.ASSERT NEMonad.RETURN NEMonad.SPEC*
lemma *mop-unroot-hp-tree-alt-def*:
  ‹*mop-unroot-hp-tree arr h* = *do* {
    a ← *mop-source-node-impl arr*;
    nnext ← *mop-hp-read-nxt-imp h arr*;
    parent ← *mop-hp-read-parent-imp h arr*;
    prev ← *mop-hp-read-prev-imp h arr*;
   if prev = *None* ∧ *parent* = *None* ∧ ¬(*a* ≠ *None* ∧ *the a* = *h*) *then RETURN* (*update-source-node-impl None arr*)
    else if a ≠ *None* ∧ *the a* = *h then RETURN* (*update-source-node-impl None arr*)
    else do {
      ASSERT (*a* ≠ *None*);
      let a' = *the a*;
      arr ← *maybe-mop-hp-update-child'-imp parent nnext arr*;
      arr ← *maybe-mop-hp-update-nxt'-imp prev nnext arr*;
      arr ← *maybe-mop-hp-update-prev'-imp nnext prev arr*;
      arr ← *maybe-mop-hp-update-parent'-imp nnext parent arr*;

      arr ← *mop-hp-update-nxt'-imp h None arr*;
      arr ← *mop-hp-update-prev'-imp h None arr*;
      arr ← *mop-hp-update-parent'-imp h None arr*;

      arr ← *mop-hp-update-nxt'-imp h* (*Some a'*) *arr*;
      arr ← *mop-hp-update-prev'-imp a'* (*Some h*) *arr*;
      RETURN (*update-source-node-impl None arr*)
    }
}›
   unfolding *mop-unroot-hp-tree-def mop-source-node-impl-def*
   by (*cases* ‹*source-node-impl arr*›)
    (*auto intro*!: *bind-cong*[*OF refl*] *simp*: *Let-def*)

sepref-def *mop-unroot-hp-tree-code*
  is ‹*uncurry* (*mop-unroot-hp-tree* :: (nat,nat)*pairing-heaps-imp* ⇒ -)›
  :: ‹$hp\text{-}assn^d *_a atom\text{-}assn^k \rightarrow_a hp\text{-}assn$›
  unfolding *mop-unroot-hp-tree-alt-def*
    atom.fold-option short-circuit-conv
  by *sepref*

sepref-def *mop-hp-update-score-imp-code*
  is ‹*uncurry2 mop-hp-update-score-imp*›
  :: ‹$atom\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a hp\text{-}assn^d \rightarrow_a hp\text{-}assn$›
  unfolding *mop-hp-update-score-imp-def hp-assn-def*
  apply (*rewrite at* ‹-[⌑:=-]› *value-of-atm-def*[*symmetric*])
  apply (*rewrite* **in** ‹- [⌑:=-]› *annot-unat-snat-upcast*[**where** *'l*=‹*64*›])
  by *sepref*

lemma *Some-eq-not-None-sepref-id-work-around*: ‹*Some h = a* ⟷ (*a* ≠ *None* ∧ *h = the a*)›
by (*cases a*) *auto*

sepref-def *mop-rescale-and-reroot-code*
is ‹*uncurry2 mop-rescale-and-reroot*›
:: ‹*atom-assn*^{k} *∗*$_a$ *uint64-nat-assn*^{k} *∗*$_a$ *hp-assn*^{d} *→*$_a$ *hp-assn*›
unfolding *mop-rescale-and-reroot-def Some-eq-not-None-sepref-id-work-around*
unfolding *atom.fold-option short-circuit-conv*
by *sepref*

sepref-def *mop-hp-is-in-code*
is ‹*uncurry mop-hp-is-in*›
:: ‹*atom-assn*^{k} *∗*$_a$ *hp-assn*^{k} *→*$_a$ *bool1-assn*›
unfolding *mop-hp-is-in-def Some-eq-not-None-sepref-id-work-around*
unfolding *atom.fold-option short-circuit-conv*
by *sepref*

sepref-def *mop-vsids-pop-min2-impl-code*
is *mop-vsids-pop-min2-impl*
:: ‹*hp-assn*^{d} *→*$_a$ *atom-assn* ×$_a$ *hp-assn*›
unfolding *mop-vsids-pop-min2-impl-def*
unfolding *atom.fold-option*
by *sepref*

lemma *mop-hp-insert-impl-spec2*:
‹(*uncurry2 mop-hp-insert-impl*, *uncurry2 hp-insert*) ∈
nat-rel ×$_f$ *nat-rel* ×$_f$ ⟨⟨*nat-rel*⟩*option-rel*,⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel* →$_f$
⟨⟨⟨*nat-rel*⟩*option-rel*,⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel*⟩*nres-rel*›
by (*intro frefI nres-relI*) (*auto intro*!: *mop-hp-insert-impl-spec*[*THEN order-trans*])

lemma *mop-rescale-and-reroot-spec2*:
‹(*uncurry2 mop-rescale-and-reroot*, *uncurry2 rescale-and-reroot*) ∈
nat-rel ×$_f$ *nat-rel* ×$_f$ ⟨⟨*nat-rel*⟩*option-rel*,⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel* →$_f$
⟨⟨⟨*nat-rel*⟩*option-rel*,⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel*⟩*nres-rel*›
by (*intro frefI nres-relI*) (*auto intro*!: *mop-rescale-and-reroot-spec*[*THEN order-trans*])

lemma *rescale-and-reroot-mop-prio-change-weight2*:
‹(*uncurry2 rescale-and-reroot*, *uncurry2* (*PR-CONST ACIDS.mop-prio-change-weight*)) ∈
nat-rel ×$_f$ *nat-rel* ×$_f$ *acids-encoded-hmrel* →$_f$ ⟨*acids-encoded-hmrel*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto intro*!: *rescale-and-reroot-mop-prio-change-weight*[*THEN order-trans*])

lemma *mop-hp-is-in-spec2*:
‹(*uncurry mop-hp-is-in*, *uncurry hp-is-in*) ∈ *nat-rel* ×$_f$ ⟨⟨*nat-rel*⟩*option-rel*, ⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel*
→$_f$ ⟨*bool-rel*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto intro*!: *mop-hp-is-in-spec*[*THEN order-trans*])

lemma *vsids-pop-min2-mop-prio-pop-min2*:
‹(*vsids-pop-min2*, *PR-CONST ACIDS.mop-prio-pop-min*) ∈ *acids-encoded-hmrel* →$_f$ ⟨*nat-rel* ×$_r$ *acids-encoded-hmrel*⟩*n*
by (*intro frefI nres-relI*)
(*auto intro*!: *vsids-pop-min2-mop-prio-pop-min*[*THEN order-trans*])

lemma *mop-vsids-pop-min2-impl2*:
shows ‹(*mop-vsids-pop-min2-impl*, *vsids-pop-min2*) ∈

⟨⟨*nat-rel*⟩*option-rel*,⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel* →$_f$
⟨*nat-rel* ×$_r$ ⟨⟨*nat-rel*⟩*option-rel*,⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto intro*!: *mop-vsids-pop-min2-impl*[*THEN order-trans*])

lemma *mop-hp-read-score-imp-mop-hp-read-score2*:
‹(*uncurry mop-hp-read-score-imp*, *uncurry mop-hp-read-score*) ∈
Id ×$_f$ ⟨⟨*nat-rel*⟩*option-rel*,⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel* →$_f$ ⟨*nat-rel*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto intro*!: *mop-hp-read-score-imp-mop-hp-read-score*[*THEN order-trans*])

thm *mop-hp-read-score-imp-mop-hp-read-score*
definition *acids-assn* :: ‹-› **where**
‹*acids-assn* = *hr-comp* (*hr-comp hp-assn* (⟨⟨*nat-rel*⟩*option-rel*, ⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel*))
acids-encoded-hmrel›

lemmas [*fcomp-norm-unfold*] = *acids-assn-def*[*symmetric*]

sepref-register *ACIDS.mop-prio-change-weight ACIDS.mop-prio-insert*
ACIDS.mop-prio-pop-min ACIDS.mop-prio-is-in

lemmas [*sepref-fr-rules*] =
mop-hp-insert-impl-code.refine[*FCOMP mop-hp-insert-impl-spec2*, *FCOMP hp-insert-spec-mop-prio-insert2*]
mop-rescale-and-reroot-code.refine[*FCOMP mop-rescale-and-reroot-spec2*, *FCOMP rescale-and-reroot-mop-prio-change-*
mop-hp-is-in-code.refine[*FCOMP mop-hp-is-in-spec2*, *FCOMP hp-is-in-mop-prio-is-in2*]
mop-vsids-pop-min2-impl-code.refine[*FCOMP mop-vsids-pop-min2-impl2*, *FCOMP vsids-pop-min2-mop-prio-pop-min2*]
mop-hp-read-score-imp-code.refine[*FCOMP mop-hp-read-score-imp-mop-hp-read-score2*, *FCOMP mop-hp-read-score-mo*

end
theory *IsaSAT-Arena*
imports
More-Sepref.WB-More-Refinement-List
IsaSAT-Literals
begin

Chapter 2

The memory representation: Arenas

We implement an "arena" memory representation: This is a flat representation of clauses, where all clauses and their headers are put one after the other. A lot of the work done here could be done automatically by a C compiler (see paragraph on Cadical below).

While this has some advantages from a performance point of view compared to an array of arrays, it allows to emulate pointers to the middle of array with extra information put before the pointer. This is an optimisation that is considered as important (at least according to Armin Biere).

In Cadical, the representation is done that way although it is implicit by putting an array into a structure (and rely on UB behaviour to make sure that the array is "inlined" into the structure). Cadical also uses another trick: the array is but inside a union. This union contains either the clause or a pointer to the new position if it has been moved (during GC-ing). There is no way for us to do so in a type-safe manner that works both for *uint64* and *nat* (unless we know some details of the implementation). For *uint64*, we could use the space used by the headers. However, it is not clear if we want to do do, since the behaviour would change between the two types, making a comparison impossible. This means that half of the blocking literals will be lost (if we iterate over the watch lists) or all (if we iterate over the clauses directly).

The order in memory is in the following order:

1. the saved position (was optional in cadical too; since sr-19, not optional);
2. the status and LBD;
3. the size;
4. the clause.

Remark that the information can be compressed to reduce the size in memory:

1. the saved position can be skipped for short clauses;
2. the LBD will most of the time be much shorter than a 32-bit integer, so only an approximation can be kept and the remaining bits be reused for the status;

In previous iteration, we had something a bit simpler:

1. the LBD was in a seperate field, allowing to store the complete LBD (which does not matter).

2. the activity was also stored and used for ties. This was beneficial on some problems (including the *eq.atree.braun* problems), but we later decided to remove it to consume less memory. This did not make a difference on the overall benchmark set. For ties, we use a pure MTF-like scheme and keep newer clauses (like CaDiCaL).

In our case, the refinement is done in two steps:

1. First, we refine our clause-mapping to a big list. This list contains the original elements. For type safety, we introduce a datatype that enumerates all possible kind of elements.
2. Then, we refine all these elements to uint32 elements.

In our formalisation, we distinguish active clauses (clauses that are not marked to be deleted) from dead clauses (that have been marked to be deleted but can still be accessed). Any dead clause can be removed from the addressable clauses (*vdom* for virtual domain). Remark that we actually do not need the full virtual domain, just the list of all active position (TODO?).

Remark that in our formalisation, we don't (at least not yet) plan to reuse freed spaces (the predicate about dead clauses must be strengthened to do so). Due to the fact that an arena is very different from an array of clauses, we refine our data structure by hand to the long list instead of introducing refinement rules. This is mostly done because iteration is very different (and it does not change what we had before anyway).

Some technical details: due to the fact that we plan to refine the arena to uint32 and that our clauses can be tautologies, the size does not fit into uint32 (technically, we have the bound *unat32-max + 1*). Therefore, we restrict the clauses to have at least length 2 and we keep *length C − 2* instead of *length C* (same for position saving). If we ever add a preprocessing path that removes tautologies, we could get rid of these two limitations.

To our own surprise, using an arena (without position saving) was exactly as fast as the our former resizable array of arrays. We did not expect this result since:

1. First, we cannot use *uint32* to iterate over clauses anymore (at least no without an additional trick like considering a slice).
2. Second, there is no reason why MLton would not already use the trick for array.

(We assume that there is no gain due the order in which we iterate over clauses, which seems a reasonnable assumption, even when considering than some clauses will subsume the previous one, and therefore, have a high chance to be in the same watch lists).

We can mark clause as used. This trick is used to implement a MTF-like scheme to keep clauses.

2.1 Status of a clause

datatype *clause-status* = *IRRED* | *LEARNED* | *DELETED*

instantiation *clause-status* :: *default*
begin

definition *default-clause-status* **where** ‹*default-clause-status* = *DELETED*›
instance by *standard*

end

2.2 Definition

The following definitions are the offset between the beginning of the clause and the specific headers before the beginning of the clause. Remark that the first offset is not always valid. Also remark that the fields are *before* the actual content of the clause.

definition *POS-SHIFT* :: *nat* **where**
‹*POS-SHIFT* = *3*›

definition *STATUS-SHIFT* :: *nat* **where**
‹*STATUS-SHIFT* = *2*›

abbreviation *LBD-SHIFT* :: *nat* **where**
‹*LBD-SHIFT* ≡ *STATUS-SHIFT*›

lemmas *LBD-SHIFT-def* = *STATUS-SHIFT-def*

definition *SIZE-SHIFT* :: *nat* **where**
‹*SIZE-SHIFT* = *1*›

definition *MAX-LENGTH-SHORT-CLAUSE* :: *nat* **where**
[*simp*]: ‹*MAX-LENGTH-SHORT-CLAUSE* = *4*›

definition *is-short-clause* **where**
[*simp*]: ‹*is-short-clause C* ⟷ *length C* ≤ *MAX-LENGTH-SHORT-CLAUSE*›

abbreviation *is-long-clause* **where**
‹*is-long-clause C* ≡ ¬*is-short-clause C*›

abbreviation (*input*) *MAX-HEADER-SIZE* :: ‹*nat*› **where**
‹*MAX-HEADER-SIZE* ≡ *3*›

abbreviation (*input*) *MIN-HEADER-SIZE* :: ‹*nat*› **where**
‹*MIN-HEADER-SIZE* ≡ *2*›

definition *header-size* :: ‹*nat clause-l* ⇒ *nat*› **where**
‹*header-size C* = (*if is-short-clause C then MIN-HEADER-SIZE else MAX-HEADER-SIZE*)›

lemmas *SHIFTS-def* = *POS-SHIFT-def STATUS-SHIFT-def SIZE-SHIFT-def*

In an attempt to avoid unfolding definitions and to not rely on the actual value of the positions of the headers before the clauses.

lemma *arena-shift-distinct*:
‹*i* > *MIN-HEADER-SIZE* ⟹ *i* − *SIZE-SHIFT* ≠ *i* − *LBD-SHIFT*›
‹*i* > *MIN-HEADER-SIZE* ⟹ *i* − *SIZE-SHIFT* ≠ *i* − *STATUS-SHIFT*›

‹*i* > *MAX-HEADER-SIZE* ⟹ *i* − *SIZE-SHIFT* ≠ *i* − *POS-SHIFT*›
‹*i* > *MAX-HEADER-SIZE* ⟹ *i* − *LBD-SHIFT* ≠ *i* − *POS-SHIFT*›
‹*i* > *MAX-HEADER-SIZE* ⟹ *i* − *STATUS-SHIFT* ≠ *i* − *POS-SHIFT*›

‹*i* > *MIN-HEADER-SIZE* ⟹ *j* > *MIN-HEADER-SIZE* ⟹ *i* − *SIZE-SHIFT* = *j* − *SIZE-SHIFT* ⟷ *i* = *j*›
‹*i* > *MIN-HEADER-SIZE* ⟹ *j* > *MIN-HEADER-SIZE* ⟹ *i* − *LBD-SHIFT* = *j* − *LBD-SHIFT* ⟷ *i* = *j*›
‹*i* > *MIN-HEADER-SIZE* ⟹ *j* > *MIN-HEADER-SIZE* ⟹ *i* − *STATUS-SHIFT* = *j* − *STATUS-SHIFT* ⟷ *i* = *j*›

‹$i >$ *MAX-HEADER-SIZE* $\Longrightarrow j >$ *MAX-HEADER-SIZE* $\Longrightarrow i -$ *POS-SHIFT* $= j -$ *POS-SHIFT* $\longleftrightarrow i = j$›

‹$i \geq$ *header-size C* $\Longrightarrow i -$ *SIZE-SHIFT* $\neq i -$ *LBD-SHIFT*›
‹$i \geq$ *header-size C* $\Longrightarrow i -$ *SIZE-SHIFT* $\neq i -$ *STATUS-SHIFT*›

‹$i \geq$ *header-size C* $\Longrightarrow$ *is-long-clause C* $\Longrightarrow i -$ *SIZE-SHIFT* $\neq i -$ *POS-SHIFT*›
‹$i \geq$ *header-size C* $\Longrightarrow$ *is-long-clause C* $\Longrightarrow i -$ *LBD-SHIFT* $\neq i -$ *POS-SHIFT*›
‹$i \geq$ *header-size C* $\Longrightarrow$ *is-long-clause C* $\Longrightarrow i -$ *STATUS-SHIFT* $\neq i -$ *POS-SHIFT*›

‹$i \geq$ *header-size C* $\Longrightarrow j \geq$ *header-size C'* $\Longrightarrow i -$ *SIZE-SHIFT* $= j -$ *SIZE-SHIFT* $\longleftrightarrow i = j$›
‹$i \geq$ *header-size C* $\Longrightarrow j \geq$ *header-size C'* $\Longrightarrow i -$ *LBD-SHIFT* $= j -$ *LBD-SHIFT* $\longleftrightarrow i = j$›
‹$i \geq$ *header-size C* $\Longrightarrow j \geq$ *header-size C'* $\Longrightarrow i -$ *STATUS-SHIFT* $= j -$ *STATUS-SHIFT* $\longleftrightarrow i = j$›
‹$i \geq$ *header-size C* $\Longrightarrow j \geq$ *header-size C'* $\Longrightarrow$ *is-long-clause C* $\Longrightarrow$ *is-long-clause C'* $\Longrightarrow$
$i -$ *POS-SHIFT* $= j -$ *POS-SHIFT* $\longleftrightarrow i = j$›
unfolding *POS-SHIFT-def STATUS-SHIFT-def LBD-SHIFT-def SIZE-SHIFT-def*
header-size-def
by (*auto split*: *if-splits simp*: *is-short-clause-def*)

lemma *header-size-ge0*[*simp*]: ‹*0* $<$ *header-size x1*›
by (*auto simp*: *header-size-def*)

datatype *arena-el* =
is-Lit: *ALit* (*xarena-lit*: ‹*nat literal*›) |
is-Size: *ASize* (*xarena-length*: *nat*) |
is-Pos: *APos* (*xarena-pos*: *nat*) |
is-Status: *AStatus* (*xarena-status*: *clause-status*) (*xarena-used*: *nat*) (*xarena-lbd*: *nat*)

type-synonym *arena* = ‹*arena-el list*›

definition *xarena-active-clause* :: ‹*arena* $\Rightarrow$ *nat clause-l* $\times$ *bool* $\Rightarrow$ *bool*› **where**
‹*xarena-active-clause arena* = (λ(*C*, *red*).
(*length C* $\geq$ *2* $\wedge$
header-size C + *length C* = *length arena* $\wedge$
(*is-long-clause C* $\longrightarrow$ (*is-Pos* (*arena*!(*header-size C* $-$ *POS-SHIFT*)) $\wedge$
xarena-pos(*arena*!(*header-size C* $-$ *POS-SHIFT*)) $\leq$ *length C* $-$ *2*))) $\wedge$
is-Status(*arena*!(*header-size C* $-$ *STATUS-SHIFT*)) $\wedge$
(*xarena-status*(*arena*!(*header-size C* $-$ *STATUS-SHIFT*)) = *IRRED* $\longleftrightarrow$ *red*) $\wedge$
(*xarena-status*(*arena*!(*header-size C* $-$ *STATUS-SHIFT*)) = *LEARNED* $\longleftrightarrow \neg$*red*) $\wedge$
is-Size(*arena*!(*header-size C* $-$ *SIZE-SHIFT*)) $\wedge$
xarena-length(*arena*!(*header-size C* $-$ *SIZE-SHIFT*)) + *2* = *length C* $\wedge$
drop (*header-size C*) *arena* = *map ALit C*
)›

As (*N* $\propto$ *i*, *irred N i*) is automatically simplified to *the* (*fmlookup N i*), we provide an alternative definition that uses the result after the simplification.

lemma *xarena-active-clause-alt-def*:
‹*xarena-active-clause arena* (*the* (*fmlookup N i*)) $\longleftrightarrow$ (
(*length* (*N*$\propto$*i*) $\geq$ *2* $\wedge$
header-size (*N*$\propto$*i*) + *length* (*N*$\propto$*i*) = *length arena* $\wedge$
(*is-long-clause* (*N*$\propto$*i*) $\longrightarrow$ (*is-Pos* (*arena*!(*header-size* (*N*$\propto$*i*) $-$ *POS-SHIFT*)) $\wedge$
xarena-pos(*arena*!(*header-size* (*N*$\propto$*i*) $-$ *POS-SHIFT*)) $\leq$ *length* (*N*$\propto$*i*) $-$ *2*)) $\wedge$
is-Status(*arena*!(*header-size* (*N*$\propto$*i*) $-$ *STATUS-SHIFT*)) $\wedge$
(*xarena-status*(*arena*!(*header-size* (*N*$\propto$*i*) $-$ *STATUS-SHIFT*)) = *IRRED* $\longleftrightarrow$ *irred N i*) $\wedge$
(*xarena-status*(*arena*!(*header-size* (*N*$\propto$*i*) $-$ *STATUS-SHIFT*)) = *LEARNED* $\longleftrightarrow \neg$*irred N i*) $\wedge$

```
    is-Size(arena!(header-size (N∝i) − SIZE-SHIFT)) ∧
    xarena-length(arena!(header-size (N∝i) − SIZE-SHIFT)) + 2 = length (N∝i) ∧
    drop (header-size (N∝i)) arena = map ALit (N∝i)
  ))›
proof −
  have C: ‹the (fmlookup N i) = (N ∝ i, irred N i)›
    by simp
  show ?thesis
    apply (subst C)
    unfolding xarena-active-clause-def prod.case
    by meson
qed
```

The extra information is required to prove "separation" between active and dead clauses. And it is true anyway and does not require any extra work to prove. TODO generalise LBD to extract from every clause?

```
definition arena-dead-clause :: ‹arena ⇒ bool› where
  ‹arena-dead-clause arena ⟷
   is-Status(arena!(MIN-HEADER-SIZE − STATUS-SHIFT)) ∧ xarena-status(arena!(MIN-HEADER-SIZE
− STATUS-SHIFT)) = DELETED ∧
    is-Size(arena!(MIN-HEADER-SIZE − SIZE-SHIFT))
›
```

When marking a clause as garbage, we do not care whether it was used or not.

```
definition extra-information-mark-to-delete where
  ‹extra-information-mark-to-delete arena i = arena[i − STATUS-SHIFT := AStatus DELETED 0 0]›
```

This extracts a single clause from the complete arena.

```
abbreviation clause-slice where
  ‹clause-slice arena N i ≡ Misc.slice (i − header-size (N∝i)) (i + length(N∝i)) arena›

abbreviation dead-clause-slice where
  ‹dead-clause-slice arena N i ≡ Misc.slice (i − MIN-HEADER-SIZE) i arena›
```

We now can lift the validity of the active and dead clauses to the whole memory and link it the mapping to clauses and the addressable space.

In our first try, the predicated *xarena-active-clause* took the whole arena as parameter. This however turned out to make the proof about updates less modular, since the slicing already takes care to ignore all irrelevant changes.

```
definition valid-arena :: ‹arena ⇒ nat clauses-l ⇒ nat set ⇒ bool› where
  ‹valid-arena arena N vdom ⟷
    (∀ i ∈# dom-m N. i < length arena ∧ i ≥ header-size (N∝i) ∧
        xarena-active-clause (clause-slice arena N i) (the (fmlookup N i))) ∧
    (∀ i ∈ vdom. i ∉# dom-m N ⟶ (i < length arena ∧ i ≥ MIN-HEADER-SIZE ∧
      arena-dead-clause (dead-clause-slice arena N i)))
›

lemma valid-arena-empty: ‹valid-arena [] fmempty {}›
  unfolding valid-arena-def
  by auto

definition arena-status where
  ‹arena-status arena i = xarena-status (arena!(i − STATUS-SHIFT))›
```

definition *arena-used* **where**
‹*arena-used arena i* = *xarena-used* (*arena*!(*i* − *STATUS-SHIFT*))›

definition *arena-length* **where**
‹*arena-length arena i* = 2 + *xarena-length* (*arena*!(*i* − *SIZE-SHIFT*))›

definition *arena-lbd* **where**
‹*arena-lbd arena i* = *xarena-lbd* (*arena*!(*i* − *LBD-SHIFT*))›

definition *arena-pos* **where**
‹*arena-pos arena i* = 2 + *xarena-pos* (*arena*!(*i* − *POS-SHIFT*))›

definition *arena-lit* **where**
‹*arena-lit arena i* = *xarena-lit* (*arena*!*i*)›

2.3 Separation properties

The following two lemmas talk about the minimal distance between two clauses in memory. They are important for the proof of correctness of all update function.

lemma *minimal-difference-between-valid-index*:
assumes ‹∀ *i* ∈# *dom-m N*. *i* < *length arena* ∧ *i* ≥ *header-size* (*N*∝*i*) ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
‹*i* ∈# *dom-m N*› **and** ‹*j* ∈# *dom-m N*› **and** ‹*j* > *i*›
shows ‹*j* − *i* ≥ *length* (*N*∝*i*) + *header-size* (*N*∝*j*)›
proof (*rule ccontr*)
assume *False*: ‹¬ *?thesis*›
let *?Ci* = ‹*the* (*fmlookup N i*)›
let *?Cj* = ‹*the* (*fmlookup N j*)›
have
1: ‹*xarena-active-clause* (*clause-slice arena N i*) (*N* ∝ *i*, *irred N i*)› **and**
2: ‹*xarena-active-clause* (*clause-slice arena N j*) (*N* ∝ *j*, *irred N j*)› **and**
i-le: ‹*i* < *length arena*› **and**
i-ge: ‹*i* ≥ *header-size*(*N*∝*i*)›**and**
j-le: ‹*j* < *length arena*› **and**
j-ge: ‹*j* ≥ *header-size*(*N*∝*j*)›
using *assms*
by *auto*

have *Ci*: ‹*?Ci* = (*N* ∝ *i*, *irred N i*)› **and** *Cj*: ‹*?Cj* = (*N* ∝ *j*, *irred N j*)›
by *auto*

have
eq: ‹*Misc.slice i* (*i* + *length* (*N* ∝ *i*)) *arena* = *map ALit* (*N* ∝ *i*)› **and**
‹*length* (*N* ∝ *i*) − *Suc 0* < *length* (*N* ∝ *i*)› **and**
length-Ni: ‹*length* (*N*∝*i*) ≥ *2*›
using *1 i-ge*
unfolding *xarena-active-clause-def extra-information-mark-to-delete-def prod.case*
apply *simp-all*
apply *force*
done

from *arg-cong*[*OF this*(*1*), *of* ‹λ*n*. *n* ! (*length* (*N*∝*i*) − *1*)›] *this*(*2*−)
have *lit*: ‹*is-Lit* (*arena* ! (*i* + *length*(*N*∝*i*) − *1*))›
using *i-le i-ge* **by** (*auto simp*: *map-nth slice-nth*)

have
Cj2: ‹*2* ≤ *length* (*N* ∝ *j*)›
using *2 j-le j-ge*
unfolding *xarena-active-clause-def extra-information-mark-to-delete-def prod.case*
header-size-def
by *simp*
have *headerj*: ‹*header-size* (*N* ∝ *j*) ≥ *MIN-HEADER-SIZE*›
unfolding *header-size-def* **by** (*auto split*: *if-splits*)
then have [*simp*]: ‹*header-size* (*N* ∝ *j*) − *POS-SHIFT* < *length* (*N* ∝ *j*) + *header-size* (*N* ∝ *j*)›
using *Cj2*
by *linarith*
have [*simp*]:
‹*is-long-clause* (*N* ∝ *j*) ⟶ *j* + (*header-size* (*N* ∝ *j*) − *POS-SHIFT*) − *header-size* (*N* ∝ *j*) = *j* − *POS-SHIFT*›
‹*j* + (*header-size* (*N* ∝ *j*) − *STATUS-SHIFT*) − *header-size* (*N* ∝ *j*) = *j* − *STATUS-SHIFT*›
‹*j* + (*header-size* (*N* ∝ *j*) − *SIZE-SHIFT*) − *header-size* (*N* ∝ *j*) = *j* − *SIZE-SHIFT*›
‹*j* + (*header-size* (*N* ∝ *j*) − *LBD-SHIFT*) − *header-size* (*N* ∝ *j*) = *j* − *LBD-SHIFT*›
using *Cj2 headerj* **unfolding** *POS-SHIFT-def STATUS-SHIFT-def LBD-SHIFT-def SIZE-SHIFT-def*
by (*auto simp*: *header-size-def*)

have
pos: ‹*is-long-clause* (*N* ∝ *j*) ⟶ *is-Pos* (*arena* ! (*j* − *POS-SHIFT*))› **and**
st: ‹*is-Status* (*arena* ! (*j* − *STATUS-SHIFT*))› **and**
size: ‹*is-Size* (*arena* ! (*j* − *SIZE-SHIFT*))›
using *2 j-le j-ge Cj2 headerj*
unfolding *xarena-active-clause-def extra-information-mark-to-delete-def prod.case*
by (*simp-all add*: *slice-nth*)
have *False* **if** *ji*: ‹*j* − *i* ≥ *length* (*N*∝*i*)›
proof −
have *Suc3*: ‹*3* = *Suc* (*Suc* (*Suc 0*))›
by *auto*
have *Suc4*: ‹*4* = *Suc* (*Suc* (*Suc* (*Suc 0*)))›
by *auto*
have *j-i-1*[*iff*]:
‹*j* − *1* = *i* + *length* (*N* ∝ *i*) − *1* ⟷ *j* = *i* + *length* (*N* ∝ *i*)›
‹*j* − *2* = *i* + *length* (*N* ∝ *i*) − *1* ⟷ *j* = *i* + *length* (*N* ∝ *i*) + *1*›
‹*j* − *3* = *i* + *length* (*N* ∝ *i*) − *1* ⟷ *j* = *i* + *length* (*N* ∝ *i*) + *2*›
‹*j* − *4* = *i* + *length* (*N* ∝ *i*) − *1* ⟷ *j* = *i* + *length* (*N* ∝ *i*) + *3*›
using *False that j-ge i-ge length-Ni* **unfolding** *Suc4 header-size-def numeral-2-eq-2*
by (*auto split*: *if-splits*)
have *H4*: ‹*Suc* (*j* − *i*) ≤ *length* (*N* ∝ *i*) + *3* ⟹ *j* − *i* = *length* (*N* ∝ *i*) ∨
j − *i* = *length* (*N* ∝ *i*) + *1* ∨ *j* − *i* = *length* (*N* ∝ *i*) + *2*›
using *False ji j-ge i-ge length-Ni* **unfolding** *Suc3 Suc4*
by (*auto simp*: *le-Suc-eq header-size-def split*: *if-splits*)
have *H5*: ‹*Suc* (*j* − *i*) ≤ *length* (*N* ∝ *i*) + *4* ⟹ *j* − *i* = *length* (*N* ∝ *i*) ∨
j − *i* = *length* (*N* ∝ *i*) + *1* ∨
(*is-long-clause* (*N* ∝ *j*) ∧ *j* = *i*+*length* (*N* ∝ *i*) + *2*)›
using *False ji j-ge i-ge length-Ni* **unfolding** *Suc3 Suc4*
by (*auto simp*: *le-Suc-eq header-size-def split*: *if-splits*)
consider
‹*is-long-clause* (*N* ∝ *j*)› ‹*j* − *POS-SHIFT* = *i* + *length*(*N*∝*i*) − *1*› |
‹*j* − *STATUS-SHIFT* = *i* + *length*(*N*∝*i*) − *1*› |
‹*j* − *LBD-SHIFT* = *i* + *length*(*N*∝*i*) − *1*› |
‹*j* − *SIZE-SHIFT* = *i* + *length*(*N*∝*i*) − *1*›
using *False ji j-ge i-ge length-Ni*
unfolding *header-size-def not-less-eq-eq STATUS-SHIFT-def SIZE-SHIFT-def*

```
      LBD-SHIFT-def le-Suc-eq POS-SHIFT-def j-i-1
    apply (cases ‹is-short-clause (N ∝ j)›)
    subgoal
      using H4 by auto
    subgoal
      using H5 by auto
    done
  then show False
    using lit pos st size by cases auto
qed
moreover have False if ji: ‹j − i < length (N∝i)›
proof −
  from arg-cong[OF eq, of ‹λxs. xs ! (j−i−1)›]
  have ‹is-Lit (arena ! (j−1))›
    using that j-le i-le ‹j > i›
    by (auto simp: slice-nth)
  then show False
    using size unfolding SIZE-SHIFT-def by auto
qed
ultimately show False
  by linarith
qed

lemma minimal-difference-between-invalid-index:
  assumes ‹valid-arena arena N vdom› and
    ‹i ∈# dom-m N› and ‹j ∉# dom-m N› and ‹j ≥ i› and ‹j ∈ vdom›
  shows ‹j − i ≥ length (N∝i) + MIN-HEADER-SIZE›
proof (rule ccontr)
  assume False: ‹¬ ?thesis›
  let ?Ci = ‹the (fmlookup N i)›
  let ?Cj = ‹the (fmlookup N j)›
  have
    1: ‹xarena-active-clause (clause-slice arena N i) (N ∝ i, irred N i)› and
    2: ‹arena-dead-clause (dead-clause-slice arena N j)› and
    i-le: ‹i < length arena› and
    i-ge: ‹i ≥ header-size(N∝i)›and
    j-le: ‹j < length arena› and
    j-ge: ‹j ≥ MIN-HEADER-SIZE›
    using assms unfolding valid-arena-def
    by auto

  have Ci: ‹?Ci = (N ∝ i, irred N i)› and Cj: ‹?Cj = (N ∝ j, irred N j)›
    by auto

  have
    eq: ‹Misc.slice i (i + length (N ∝ i)) arena = map ALit (N ∝ i)› and
    ‹length (N ∝ i) − Suc 0 < length (N ∝ i)› and
    length-Ni: ‹length (N∝i) ≥ 2› and
    pos: ‹is-long-clause (N ∝ i) ⟶
      is-Pos (arena ! (i − POS-SHIFT))› and
    status: ‹is-Status (arena ! (i − STATUS-SHIFT))› and
    size: ‹is-Size (arena ! (i − SIZE-SHIFT))› and
    st-init: ‹(xarena-status (arena ! (i − STATUS-SHIFT)) = IRRED) = (irred N i)› and
    st-learned: ‹(xarena-status (arena ! (i − STATUS-SHIFT)) = LEARNED) = (¬ irred N i)›
    using 1 i-ge i-le
    unfolding xarena-active-clause-def extra-information-mark-to-delete-def prod.case
```

```
    unfolding STATUS-SHIFT-def LBD-SHIFT-def SIZE-SHIFT-def POS-SHIFT-def
   apply (simp-all add: header-size-def slice-nth split: if-splits)
  apply force+
  done

 have
  st: ‹is-Status (arena ! (j − STATUS-SHIFT))› and
  del: ‹xarena-status (arena ! (j − STATUS-SHIFT)) = DELETED›
  using 2 j-le j-ge unfolding arena-dead-clause-def STATUS-SHIFT-def
  by (simp-all add: header-size-def slice-nth)
 consider
  ‹j = i› |
  ‹j − STATUS-SHIFT ≥ i› and ‹j > i› |
  ‹j − STATUS-SHIFT < i›
  using False ‹j ≥ i› unfolding STATUS-SHIFT-def
  by linarith
 then show False
 proof cases
  case 1
  then show False
   using del st-init st-learned by auto
 next
  case 2
  then have ‹j − STATUS-SHIFT < i + length (N∝i)›
   using ‹j ≥ i› False j-ge
   unfolding not-less-eq-eq STATUS-SHIFT-def by simp
  with arg-cong[OF eq, of ‹λn. n ! (j − STATUS-SHIFT − i)›]
  have lit: ‹is-Lit (arena ! (j − STATUS-SHIFT))›
   using ‹j ≥ i› 2 i-le i-ge j-ge by (auto simp: map-nth slice-nth STATUS-SHIFT-def)
  with st
  show False by auto
 next
  case 3
  then consider
   ‹j − STATUS-SHIFT = i − STATUS-SHIFT› |
   ‹j − STATUS-SHIFT = i − SIZE-SHIFT› |
   ‹is-long-clause (N ∝ i)› and ‹j − STATUS-SHIFT = i − POS-SHIFT›
   using ‹j ≥ i›
   unfolding STATUS-SHIFT-def LBD-SHIFT-def SIZE-SHIFT-def POS-SHIFT-def
   by force
  then show False
   apply cases
   subgoal using st status st-init st-learned del by auto
   subgoal using st size by auto
   subgoal using st pos by auto
   done
 qed
qed
```

At first we had the weaker $(1::'a) \leq i - j$ which we replaced by $(4::'a) \leq i - j$. The former however was able to solve many more goals due to different handling between $1::'a$ (which is simplified to *Suc 0*) and $4::'a$ (whi::natch is not). Therefore, we replaced $4::'a$ by *Suc* (*Suc* (*Suc* (*Suc 0*)))

```
lemma minimal-difference-between-invalid-index2:
 assumes ‹valid-arena arena N vdom› and
```

```
    ‹i ∈# dom-m N› and ‹j ∉# dom-m N› and ‹j < i› and ‹j ∈ vdom›
  shows ‹i − j ≥ (Suc (Suc 0))› and
    ‹is-long-clause (N ∝ i) ⟹ i − j ≥ (Suc (Suc (Suc 0)))›
proof −
  let ?Ci = ‹the (fmlookup N i)›
  let ?Cj = ‹the (fmlookup N j)›
  have
    1: ‹xarena-active-clause (clause-slice arena N i) (N ∝ i, irred N i)› and
    2: ‹arena-dead-clause (dead-clause-slice arena N j)› and
    i-le: ‹i < length arena› and
    i-ge: ‹i ≥ header-size(N∝i)›and
    j-le: ‹j < length arena› and
    j-ge: ‹j ≥ MIN-HEADER-SIZE›
    using assms unfolding valid-arena-def
    by auto

  have Ci: ‹?Ci = (N ∝ i, irred N i)› and Cj: ‹?Cj = (N ∝ j, irred N j)›
    by auto


  have
    eq: ‹Misc.slice i (i + length (N ∝ i)) arena = map ALit (N ∝ i)› and
    ‹length (N ∝ i) − Suc 0 < length (N ∝ i)› and
    length-Ni: ‹length (N∝i) ≥ 2› and
    pos: ‹is-long-clause (N ∝ i) ⟶
      is-Pos (arena ! (i − POS-SHIFT))› and
    status: ‹is-Status (arena ! (i − STATUS-SHIFT))› and
    size: ‹is-Size (arena ! (i − SIZE-SHIFT))› and
    st-init: ‹(xarena-status (arena ! (i − STATUS-SHIFT)) = IRRED) ⟷ (irred N i)› and
    st-learned: ‹ (xarena-status (arena ! (i − STATUS-SHIFT)) = LEARNED) ⟷ ¬irred N i›
    using 1 i-ge i-le
    unfolding xarena-active-clause-def extra-information-mark-to-delete-def prod.case
      unfolding STATUS-SHIFT-def LBD-SHIFT-def SIZE-SHIFT-def POS-SHIFT-def
     apply (simp-all add: header-size-def slice-nth split: if-splits)
    apply force+
    done

  have
    st: ‹is-Status (arena ! (j − STATUS-SHIFT))› and
    del: ‹xarena-status (arena ! (j − STATUS-SHIFT)) = DELETED› and
    size': ‹is-Size (arena ! (j − SIZE-SHIFT))›
    using 2 j-le j-ge unfolding arena-dead-clause-def SHIFTS-def
    by (simp-all add: header-size-def slice-nth)
  have 4: ‹4 = Suc (Suc (Suc (Suc 0)))›
    by auto
  have [simp]: ‹a < 4 ⟹ j − Suc a = i − Suc 0 ⟷ i = j − a› for a
    using ‹i > j› j-ge i-ge
    by (auto split: if-splits simp: not-less-eq-eq le-Suc-eq )
  have [simp]: ‹Suc i − j = Suc a ⟷ i − j = a› for a
    using ‹i > j› j-ge i-ge
    by (auto split: if-splits simp: not-less-eq-eq le-Suc-eq)


  show 1: ‹i − j ≥ (Suc (Suc 0))› (is ?A)
  proof (rule ccontr)
    assume False: ‹¬?A›
```

```
    consider
        ‹i − STATUS-SHIFT = j − STATUS-SHIFT› |
        ‹i − STATUS-SHIFT = j − SIZE-SHIFT›
      using False ‹i > j› j-ge i-ge unfolding SHIFTS-def header-size-def 4
      by (auto split: if-splits simp: not-less-eq-eq le-Suc-eq )
    then show False
      apply cases
      subgoal using st status st-init st-learned del by auto
      subgoal using status size' by auto
      done
  qed

  show ‹i − j ≥ (Suc (Suc (Suc 0)))› (is ?A)
    if long: ‹is-long-clause (N ∝ i)›
  proof (rule ccontr)
    assume False: ‹¬ ?A›

    have [simp]: ‹a < 3 ⟹ a' < 2 ⟹ i − Suc a = j − Suc a' ⟷ i − a = j − a'› for a a'
      using ‹i > j› j-ge i-ge long
      by (auto split: if-splits simp: not-less-eq-eq le-Suc-eq )
    have ‹i − j = (Suc (Suc 0))›
      using 1 ‹i > j› False j-ge i-ge long unfolding SHIFTS-def header-size-def 4
      by (auto split: if-splits simp: not-less-eq-eq le-Suc-eq)
    then have ‹i − POS-SHIFT = j − SIZE-SHIFT›
      using 1 ‹i > j› j-ge i-ge long unfolding SHIFTS-def header-size-def 4
      by (auto split: if-splits simp: not-less-eq-eq le-Suc-eq)
    then show False
      using pos long size'
      by auto
  qed
qed

lemma valid-arena-in-vdom-le-arena:
  assumes ‹valid-arena arena N vdom› and ‹j ∈ vdom›
  shows ‹j < length arena› and ‹j ≥ MIN-HEADER-SIZE›
  using assms unfolding valid-arena-def
  by (cases ‹j ∈# dom-m N›; auto simp: header-size-def
    dest!: multi-member-split split: if-splits; fail)+

lemma valid-minimal-difference-between-valid-index:
  assumes ‹valid-arena arena N vdom› and
    ‹i ∈# dom-m N› and ‹j ∈# dom-m N› and ‹j > i›
  shows ‹j − i ≥ length (N∝i) + header-size (N∝j)›
  by (rule minimal-difference-between-valid-index[OF - assms(2−4)])
  (use assms(1) in ‹auto simp: valid-arena-def›)
```

Updates

Mark to delete **lemma** *clause-slice-extra-information-mark-to-delete*:

```
  assumes
    i: ‹i ∈# dom-m N› and
    ia: ‹ia ∈# dom-m N› and
    dom: ‹∀ i ∈# dom-m N. i < length arena ∧ i ≥ header-size (N∝i) ∧
        xarena-active-clause (clause-slice arena N i) (the (fmlookup N i))›
  shows
    ‹clause-slice (extra-information-mark-to-delete arena i) N ia =
```

```
    (if ia = i then extra-information-mark-to-delete (clause-slice arena N ia) (header-size (N∝i))
      else clause-slice arena N ia)›
proof −
  have ia-ge: ‹ia ≥ header-size(N ∝ ia)› ‹ia < length arena› and
   i-ge:  ‹i ≥ header-size(N ∝ i)› ‹i < length arena›
    using dom ia i unfolding xarena-active-clause-def
    by auto

  show ?thesis
    using minimal-difference-between-valid-index[OF dom i ia] i-ge
    minimal-difference-between-valid-index[OF dom ia i] ia-ge
    by (cases ‹ia < i›)
     (auto simp: extra-information-mark-to-delete-def STATUS-SHIFT-def drop-update-swap
       Misc.slice-def header-size-def split: if-splits)
qed

lemma clause-slice-extra-information-mark-to-delete-dead:
  assumes
    i: ‹i ∈# dom-m N› and
    ia: ‹ia ∉# dom-m N› ‹ia ∈ vdom› and
    dom: ‹valid-arena arena N vdom›
  shows
    ‹arena-dead-clause (dead-clause-slice (extra-information-mark-to-delete arena i) N ia) =
      arena-dead-clause (dead-clause-slice arena N ia)›
proof −
  have ia-ge: ‹ia ≥ MIN-HEADER-SIZE› ‹ia < length arena› and
   i-ge:  ‹i ≥ header-size(N ∝ i)› ‹i < length arena›
    using dom ia i unfolding valid-arena-def
    by auto
  show ?thesis
    using minimal-difference-between-invalid-index[OF dom i ia(1) - ia(2)] i-ge ia-ge
    using minimal-difference-between-invalid-index2[OF dom i ia(1) - ia(2)] ia-ge
    by (cases ‹ia < i›)
     (auto simp: extra-information-mark-to-delete-def STATUS-SHIFT-def drop-update-swap
       arena-dead-clause-def
       Misc.slice-def header-size-def split: if-splits)
qed

lemma length-extra-information-mark-to-delete[simp]:
  ‹length (extra-information-mark-to-delete arena i) = length arena›
  unfolding extra-information-mark-to-delete-def by auto

lemma valid-arena-mono: ‹valid-arena ab ar vdom1 ⟹ vdom2 ⊆ vdom1 ⟹ valid-arena ab ar vdom2›
  unfolding valid-arena-def
  by fast

lemma valid-arena-extra-information-mark-to-delete:
  assumes arena: ‹valid-arena arena N vdom› and i: ‹i ∈# dom-m N›
  shows ‹valid-arena (extra-information-mark-to-delete arena i) (fmdrop i N) (insert i vdom)›
proof −
  let ?arena = ‹extra-information-mark-to-delete arena i›
  have [simp]: ‹i ∉# remove1-mset i (dom-m N)›
    ‹⋀ia. ia ∉# remove1-mset i (dom-m N) ⟷ ia =i ∨ (i ≠ ia ∧ ia ∉# dom-m N)›
    using assms distinct-mset-dom[of N]
    by (auto dest!: multi-member-split simp: add-mset-eq-add-mset)
  have
```

dom: ‹∀ *i*∈#*dom-m N*.
i < *length arena* ∧
header-size (*N* ∝ *i*) ≤ *i* ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
dom': ‹⋀*i*. *i*∈#*dom-m N* ⟹
i < *length arena* ∧
header-size (*N* ∝ *i*) ≤ *i* ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
vdom: ‹⋀*i*. *i*∈*vdom* ⟶ *i* ∉# *dom-m N* ⟶ *MIN-HEADER-SIZE* ≤ *i* ∧ *arena-dead-clause* (*dead-clause-slice arena N i*)›
using *assms* **unfolding** *valid-arena-def* **by** *auto*
have ‹*ia*∈#*dom-m* (*fmdrop i N*) ⟹
ia < *length ?arena* ∧
header-size (*fmdrop i N* ∝ *ia*) ≤ *ia* ∧
xarena-active-clause (*clause-slice ?arena* (*fmdrop i N*) *ia*) (*the* (*fmlookup* (*fmdrop i N*) *ia*))› **for** *ia*
using *dom'*[*of ia*] *clause-slice-extra-information-mark-to-delete*[*OF i - dom*, *of ia*]
by *auto*
moreover have ‹*ia* ≠ *i* ⟶ *ia*∈*insert i vdom* ⟶
ia ∉# *dom-m* (*fmdrop i N*) ⟶
MIN-HEADER-SIZE ≤ *ia* ∧ *arena-dead-clause*
(*dead-clause-slice* (*extra-information-mark-to-delete arena i*) (*fmdrop i N*) *ia*)› **for** *ia*
using *vdom*[*of ia*] *clause-slice-extra-information-mark-to-delete-dead*[*OF i - - arena*, *of ia*]
by *auto*
moreover have ‹*MIN-HEADER-SIZE* ≤ *i* ∧ *arena-dead-clause*
(*dead-clause-slice* (*extra-information-mark-to-delete arena i*) (*fmdrop i N*) *i*)›
using *dom'*[*of i*, *OF i*]
unfolding *arena-dead-clause-def xarena-active-clause-alt-def*
extra-information-mark-to-delete-def **apply** (*cases* ‹*is-short-clause* (*N* ∝ *i*)›)
by (*simp-all add*: *SHIFTS-def header-size-def Misc.slice-def drop-update-swap min-def*) *force+*
ultimately show *?thesis*
using *assms* **unfolding** *valid-arena-def*
by *auto*
qed

lemma *valid-arena-extra-information-mark-to-delete'*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *i*: ‹*i* ∈# *dom-m N*›
shows ‹*valid-arena* (*extra-information-mark-to-delete arena i*) (*fmdrop i N*) *vdom*›
using *valid-arena-extra-information-mark-to-delete*[*OF assms*]
by (*auto intro*: *valid-arena-mono*)

Removable from addressable space **lemma** *valid-arena-remove-from-vdom*:
assumes ‹*valid-arena arena N* (*insert i vdom*)›
shows ‹*valid-arena arena N vdom*›
using *assms valid-arena-def*
by (*auto dest*!: *in-diffD*)

Update LBD **abbreviation** *MAX-LBD* :: ‹*nat*› **where**
‹*MAX-LBD* ≡ *67108863*›

lemma *MAX-LBD-alt-def*:
‹*MAX-LBD* = (*2^26−1*)›
by *auto*

definition *shorten-lbd* :: ‹*nat* ⇒ *nat*› **where**

‹*shorten-lbd n* = (*if n* ≥ *MAX-LBD then MAX-LBD else n*)›

definition *update-lbd* **where**
‹*update-lbd C lbd arena* = *arena*[*C* − *LBD-SHIFT* := *AStatus* (*arena-status arena C*)
(*arena-used arena C*) (*shorten-lbd lbd*)]›

lemma *clause-slice-update-lbd*:
assumes
i: ‹*i* ∈# *dom-m N*› **and**
ia: ‹*ia* ∈# *dom-m N*› **and**
dom: ‹∀ *i* ∈# *dom-m N*. *i* < *length arena* ∧ *i* ≥ *header-size* (*N*∝*i*) ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))›
shows
‹*clause-slice* (*update-lbd i lbd arena*) *N ia* =
(*if ia* = *i then update-lbd* (*header-size* (*N*∝*i*)) *lbd* (*clause-slice arena N ia*)
else clause-slice arena N ia)›
proof −
have *ia-ge*: ‹*ia* ≥ *header-size*(*N* ∝ *ia*)› ‹*ia* < *length arena*› **and**
i-ge: ‹*i* ≥ *header-size*(*N* ∝ *i*)› ‹*i* < *length arena*›
using *dom ia i* **unfolding** *xarena-active-clause-def*
by *auto*

show *?thesis*
using *minimal-difference-between-valid-index*[*OF dom i ia*] *i-ge*
minimal-difference-between-valid-index[*OF dom ia i*] *ia-ge*
by (*cases* ‹*ia* < *i*›)
(*auto simp*: *extra-information-mark-to-delete-def drop-update-swap*
update-lbd-def SHIFTS-def arena-status-def arena-used-def
Misc.slice-def header-size-def split: *if-splits*)
qed

lemma *length-update-lbd*[*simp*]:
‹*length* (*update-lbd i lbd arena*) = *length arena*›
by (*auto simp*: *update-lbd-def*)

lemma *clause-slice-update-lbd-dead*:
assumes
i: ‹*i* ∈# *dom-m N*› **and**
ia: ‹*ia* ∉# *dom-m N*› ‹*ia* ∈ *vdom*› **and**
dom: ‹*valid-arena arena N vdom*›
shows
‹*arena-dead-clause* (*dead-clause-slice* (*update-lbd i lbd arena*) *N ia*) =
arena-dead-clause (*dead-clause-slice arena N ia*)›
proof −
have *ia-ge*: ‹*ia* ≥ *MIN-HEADER-SIZE*› ‹*ia* < *length arena*› **and**
i-ge: ‹*i* ≥ *header-size*(*N* ∝ *i*)› ‹*i* < *length arena*›
using *dom ia i* **unfolding** *valid-arena-def*
by *auto*
show *?thesis*
using *minimal-difference-between-invalid-index*[*OF dom i ia*(*1*) - *ia*(*2*)] *i-ge ia-ge*
using *minimal-difference-between-invalid-index2*[*OF dom i ia*(*1*) - *ia*(*2*)] *ia-ge*
by (*cases* ‹*ia* < *i*›)
(*auto simp*: *extra-information-mark-to-delete-def drop-update-swap*
arena-dead-clause-def update-lbd-def SHIFTS-def
Misc.slice-def header-size-def split: *if-splits*)

qed

lemma *xarena-active-clause-update-lbd-same*:
assumes
‹$i \geq$ *header-size* $(N \propto i)$› **and**
‹$i <$ *length arena*› **and**
‹*xarena-active-clause* (*clause-slice arena N i*)
(*the* (*fmlookup N i*))›
shows ‹*xarena-active-clause* (*update-lbd* (*header-size* $(N \propto i)$) *lbd* (*clause-slice arena N i*))
(*the* (*fmlookup N i*))›
using *assms*
by (*cases* ‹*is-short-clause* $(N \propto i)$›)
(*simp-all add*: *xarena-active-clause-alt-def update-lbd-def SHIFTS-def Misc.slice-def*
header-size-def arena-status-def arena-used-def)

lemma *valid-arena-update-lbd*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *i*: ‹$i \in\#$ *dom-m N*›
shows ‹*valid-arena* (*update-lbd i lbd arena*) *N vdom*›
proof −
let *?arena* = ‹*update-lbd i lbd arena*›
have [*simp*]: ‹$i \notin\#$ *remove1-mset i* (*dom-m N*)›
‹$\bigwedge ia.$ $ia \notin\#$ *remove1-mset i* (*dom-m N*) $\longleftrightarrow$ $ia = i \vee (i \neq ia \wedge ia \notin\#$ *dom-m N*)›
using *assms distinct-mset-dom*[*of N*]
by (*auto dest*!: *multi-member-split simp*: *add-mset-eq-add-mset*)
have
dom: ‹$\forall\, i \in\#$*dom-m N*.
$i <$ *length arena* $\wedge$
header-size $(N \propto i) \leq i \wedge$
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
dom′: ‹$\bigwedge i.$ $i \in\#$*dom-m N* $\Longrightarrow$
$i <$ *length arena* $\wedge$
header-size $(N \propto i) \leq i \wedge$
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
vdom: ‹$\bigwedge i.$ $i \in$*vdom* $\longrightarrow$ $i \notin\#$ *dom-m N* $\longrightarrow$ *MIN-HEADER-SIZE* $\leq i \wedge$ *arena-dead-clause* (*dead-clause-slice arena N i*)›
using *assms* **unfolding** *valid-arena-def* **by** *auto*
have ‹$ia \in\#$*dom-m N* $\Longrightarrow$ $ia \neq i \Longrightarrow$
$ia <$ *length ?arena* $\wedge$
header-size $(N \propto ia) \leq ia \wedge$
xarena-active-clause (*clause-slice ?arena N ia*) (*the* (*fmlookup N ia*))› **for** *ia*
using *dom′*[*of ia*] *clause-slice-update-lbd*[*OF i - dom*, *of ia lbd*]
by *auto*
moreover have ‹$ia = i \Longrightarrow$
$ia <$ *length ?arena* $\wedge$
header-size $(N \propto ia) \leq ia \wedge$
xarena-active-clause (*clause-slice ?arena N ia*) (*the* (*fmlookup N ia*))› **for** *ia*
using *dom′*[*of ia*] *clause-slice-update-lbd*[*OF i - dom*, *of ia lbd*] *i*
by (*simp add*: *xarena-active-clause-update-lbd-same*)
moreover have ‹$ia \in$*vdom* $\longrightarrow$
$ia \notin\#$ *dom-m N* $\longrightarrow$
MIN-HEADER-SIZE $\leq ia \wedge$ *arena-dead-clause*
(*dead-clause-slice* (*update-lbd i lbd arena*) (*fmdrop i N*) *ia*)› **for** *ia*
using *vdom*[*of ia*] *clause-slice-update-lbd-dead*[*OF i - - arena*, *of ia*] *i*
by *auto*
ultimately show *?thesis*

using *assms* **unfolding** *valid-arena-def*
by *auto*
qed

Update saved position **definition** *update-pos-direct* **where**
‹*update-pos-direct C pos arena* = *arena*[*C* − *POS-SHIFT* := *APos pos*]›

definition *arena-update-pos* **where**
‹*arena-update-pos C pos arena* = *arena*[*C* − *POS-SHIFT* := *APos* (*pos* − *2*)]›

lemma *arena-update-pos-alt-def*:
‹*arena-update-pos C i N* = *update-pos-direct C* (*i* − *2*) *N*›
by (*auto simp*: *arena-update-pos-def update-pos-direct-def*)

lemma *clause-slice-update-pos*:
assumes
i: ‹*i* ∈# *dom-m N*› **and**
ia: ‹*ia* ∈# *dom-m N*› **and**
dom: ‹∀ *i* ∈# *dom-m N*. *i* < *length arena* ∧ *i* ≥ *header-size* (*N*∝*i*) ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
long: ‹*is-long-clause* (*N* ∝ *i*)›
shows
‹*clause-slice* (*update-pos-direct i pos arena*) *N ia* =
(*if ia* = *i then update-pos-direct* (*header-size* (*N*∝*i*)) *pos* (*clause-slice arena N ia*)
else clause-slice arena N ia)›
proof −
have *ia-ge*: ‹*ia* ≥ *header-size*(*N* ∝ *ia*)› ‹*ia* < *length arena*› **and**
i-ge: ‹*i* ≥ *header-size*(*N* ∝ *i*)› ‹*i* < *length arena*›
using *dom ia i* **unfolding** *xarena-active-clause-def*
by *auto*
show *?thesis*
using *minimal-difference-between-valid-index*[*OF dom i ia*] *i-ge*
minimal-difference-between-valid-index[*OF dom ia i*] *ia-ge long*
by (*cases* ‹*ia* < *i*›)
(*auto simp*: *extra-information-mark-to-delete-def drop-update-swap*
update-pos-direct-def SHIFTS-def
Misc.slice-def header-size-def split: *if-splits*)
qed

lemma *clause-slice-update-pos-dead*:
assumes
i: ‹*i* ∈# *dom-m N*› **and**
ia: ‹*ia* ∉# *dom-m N*› ‹*ia* ∈ *vdom*› **and**
dom: ‹*valid-arena arena N vdom*› **and**
long: ‹*is-long-clause* (*N* ∝ *i*)›
shows
‹*arena-dead-clause* (*dead-clause-slice* (*update-pos-direct i pos arena*) *N ia*) =
arena-dead-clause (*dead-clause-slice arena N ia*)›
proof −
have *ia-ge*: ‹*ia* ≥ *MIN-HEADER-SIZE*› ‹*ia* < *length arena*› **and**
i-ge: ‹*i* ≥ *header-size*(*N* ∝ *i*)› ‹*i* < *length arena*›
using *dom ia i long* **unfolding** *valid-arena-def*
by *auto*
show *?thesis*

using *minimal-difference-between-invalid-index*[*OF dom i ia*(*1*) - *ia*(*2*)] *i-ge ia-ge*
using *minimal-difference-between-invalid-index2*[*OF dom i ia*(*1*) - *ia*(*2*)] *ia-ge long*
by (*cases* ‹*ia* < *i*›)
(*auto simp*: *extra-information-mark-to-delete-def drop-update-swap*
arena-dead-clause-def update-pos-direct-def SHIFTS-def
Misc.slice-def header-size-def split: *if-splits*)
qed

lemma *xarena-active-clause-update-pos-same*:
assumes
‹*i* ≥ *header-size* (*N* ∝ *i*)› **and**
‹*i* < *length arena*› **and**
‹*xarena-active-clause* (*clause-slice arena N i*)
(*the* (*fmlookup N i*))› **and**
long: ‹*is-long-clause* (*N* ∝ *i*)› **and**
‹*pos* ≤ *length* (*N* ∝ *i*) − *2*›
shows ‹*xarena-active-clause* (*update-pos-direct* (*header-size* (*N*∝*i*)) *pos* (*clause-slice arena N i*))
(*the* (*fmlookup N i*))›
using *assms*
by (*simp-all add*: *update-pos-direct-def SHIFTS-def Misc.slice-def*
header-size-def xarena-active-clause-alt-def)

lemma *length-update-pos*[*simp*]:
‹*length* (*update-pos-direct i pos arena*) = *length arena*›
by (*auto simp*: *update-pos-direct-def*)

lemma *valid-arena-update-pos*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *i*: ‹*i* ∈# *dom-m N*› **and**
long: ‹*is-long-clause* (*N* ∝ *i*)›**and**
pos: ‹*pos* ≤ *length* (*N* ∝ *i*) − *2*›
shows ‹*valid-arena* (*update-pos-direct i pos arena*) *N vdom*›
proof −
let *?arena* = ‹*update-pos-direct i pos arena*›
have [*simp*]: ‹*i* ∉# *remove1-mset i* (*dom-m N*)›
‹⋀*ia*. *ia* ∉# *remove1-mset i* (*dom-m N*) ⟷ *ia* =*i* ∨ (*i* ≠ *ia* ∧ *ia* ∉# *dom-m N*)›
using *assms distinct-mset-dom*[*of N*]
by (*auto dest*!: *multi-member-split simp*: *add-mset-eq-add-mset*)
have
dom: ‹∀ *i*∈#*dom-m N*.
i < *length arena* ∧
header-size (*N* ∝ *i*) ≤ *i* ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
dom': ‹⋀*i*. *i*∈#*dom-m N* ⟹
i < *length arena* ∧
header-size (*N* ∝ *i*) ≤ *i* ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
vdom: ‹⋀*i*. *i*∈*vdom* ⟶ *i* ∉# *dom-m N* ⟶ *MIN-HEADER-SIZE* ≤ *i* ∧ *arena-dead-clause* (*dead-clause-slice arena N i*)›
using *assms* **unfolding** *valid-arena-def* **by** *auto*
have ‹*ia*∈#*dom-m N* ⟹ *ia* ≠ *i* ⟹
ia < *length ?arena* ∧
header-size (*N* ∝ *ia*) ≤ *ia* ∧
xarena-active-clause (*clause-slice ?arena N ia*) (*the* (*fmlookup N ia*))› **for** *ia*
using *dom'*[*of ia*] *clause-slice-update-pos*[*OF i - dom*, *of ia pos*] *long*
by *auto*
moreover have ‹*ia* = *i* ⟹

ia < *length ?arena* ∧
header-size (*N* ∝ *ia*) ≤ *ia* ∧
xarena-active-clause (*clause-slice ?arena N ia*) (*the* (*fmlookup N ia*))› **for** *ia*
using *dom'*[*of ia*] *clause-slice-update-pos*[*OF i - dom*, *of ia pos*] *i long pos*
by (*simp add*: *xarena-active-clause-update-pos-same*)
moreover have ‹*ia*∈*vdom* ⟶
ia ∉# *dom-m N* ⟶
MIN-HEADER-SIZE ≤ *ia* ∧ *arena-dead-clause*
(*dead-clause-slice* (*update-pos-direct i pos arena*) *N ia*)› **for** *ia*
using *vdom*[*of ia*] *clause-slice-update-pos-dead*[*OF i - - arena*, *of ia*] *i long*
by *auto*
ultimately show *?thesis*
using *assms* **unfolding** *valid-arena-def*
by *auto*
qed

Swap literals **definition** *swap-lits* **where**
‹*swap-lits C i j arena* = *swap arena* (*C* +*i*) (*C* + *j*)›

lemma *clause-slice-swap-lits*:
assumes
i: ‹*i* ∈# *dom-m N*› **and**
ia: ‹*ia* ∈# *dom-m N*› **and**
dom: ‹∀ *i* ∈# *dom-m N*. *i* < *length arena* ∧ *i* ≥ *header-size* (*N*∝*i*) ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
k: ‹*k* < *length* (*N* ∝ *i*)› **and**
l: ‹*l* < *length* (*N* ∝ *i*)›
shows
‹*clause-slice* (*swap-lits i k l arena*) *N ia* =
(*if ia* = *i then swap-lits* (*header-size* (*N*∝*i*)) *k l* (*clause-slice arena N ia*)
else clause-slice arena N ia)›
proof −
have *ia-ge*: ‹*ia* ≥ *header-size*(*N* ∝ *ia*)› ‹*ia* < *length arena*› **and**
i-ge: ‹*i* ≥ *header-size*(*N* ∝ *i*)› ‹*i* < *length arena*›
using *dom ia i* **unfolding** *xarena-active-clause-def*
by *auto*

show *?thesis*
using *minimal-difference-between-valid-index*[*OF dom i ia*] *i-ge*
minimal-difference-between-valid-index[*OF dom ia i*] *ia-ge k l*
by (*cases* ‹*ia* < *i*›)
(*auto simp*: *extra-information-mark-to-delete-def drop-update-swap*
swap-lits-def SHIFTS-def swap-def ac-simps
Misc.slice-def header-size-def split: *if-splits*)
qed

lemma *length-swap-lits*[*simp*]:
‹*length* (*swap-lits i k l arena*) = *length arena*›
by (*auto simp*: *swap-lits-def*)

lemma *clause-slice-swap-lits-dead*:
assumes
i: ‹*i* ∈# *dom-m N*› **and**
ia: ‹*ia* ∉# *dom-m N*› ‹*ia* ∈ *vdom*› **and**
dom: ‹*valid-arena arena N vdom*›**and**
k: ‹*k* < *length* (*N* ∝ *i*)› **and**

```
    l: ‹l < length (N ∝ i)›
  shows
    ‹arena-dead-clause (dead-clause-slice (swap-lits i k l arena) N ia) =
      arena-dead-clause (dead-clause-slice arena N ia)›
proof −
  have ia-ge: ‹ia ≥ MIN-HEADER-SIZE› ‹ia < length arena› and
   i-ge:  ‹i ≥ header-size(N ∝ i)› ‹i < length arena›
    using dom ia i unfolding valid-arena-def
    by auto
  show ?thesis
    using minimal-difference-between-invalid-index[OF dom i ia(1) - ia(2)] i-ge ia-ge
    using minimal-difference-between-invalid-index2[OF dom i ia(1) - ia(2)] ia-ge k l
    by (cases ‹ia < i›)
     (auto simp: extra-information-mark-to-delete-def drop-update-swap
      arena-dead-clause-def swap-lits-def SHIFTS-def swap-def ac-simps
       Misc.slice-def header-size-def split: if-splits)
qed

lemma xarena-active-clause-swap-lits-same:
  assumes
    ‹i ≥ header-size (N ∝ i)› and
    ‹i < length arena› and
    ‹xarena-active-clause (clause-slice arena N i)
     (the (fmlookup N i))›and
    k: ‹k < length (N ∝ i)› and
    l: ‹l < length (N ∝ i)›
  shows ‹xarena-active-clause (clause-slice (swap-lits i k l arena) N i)
     (the (fmlookup (N(i ↪ swap (N ∝ i) k l)) i))›
  using assms
  unfolding xarena-active-clause-alt-def
  by (cases ‹is-short-clause (N ∝ i)›)
    (simp-all add:  swap-lits-def SHIFTS-def min-def swap-nth-if map-swap swap-swap
    header-size-def ac-simps is-short-clause-def split: if-splits)

lemma is-short-clause-swap[simp]: ‹is-short-clause (swap (N ∝ i) k l) = is-short-clause (N ∝ i)›
  by (auto simp: header-size-def is-short-clause-def split: if-splits)

lemma header-size-swap[simp]: ‹header-size (swap (N ∝ i) k l) = header-size (N ∝ i)›
  by (auto simp: header-size-def split: if-splits)

lemma valid-arena-swap-lits:
  assumes arena: ‹valid-arena arena N vdom› and i: ‹i ∈# dom-m N› and
    k: ‹k < length (N ∝ i)› and
    l: ‹l < length (N ∝ i)›
  shows ‹valid-arena (swap-lits i k l arena) (N(i ↪ swap (N ∝ i) k l)) vdom›
proof −
  let ?arena = ‹swap-lits i k l arena›
  have [simp]: ‹i ∉# remove1-mset i (dom-m N)›
    ‹⋀ia. ia ∉# remove1-mset i (dom-m N) ⟷ ia =i ∨ (i ≠ ia ∧ ia ∉# dom-m N)›
    using assms distinct-mset-dom[of N]
    by (auto dest!: multi-member-split simp: add-mset-eq-add-mset)
  have
    dom: ‹∀ i∈#dom-m N.
        i < length arena ∧
        header-size (N ∝ i) ≤ i ∧
        xarena-active-clause (clause-slice arena N i) (the (fmlookup N i))› and
```

dom': ‹⋀*i*. *i*∈#*dom-m N* ⟹
i < *length arena* ∧
header-size (*N* ∝ *i*) ≤ *i* ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
vdom: ‹⋀*i*. *i*∈*vdom* ⟶ *i* ∉# *dom-m N* ⟶ *MIN-HEADER-SIZE* ≤ *i* ∧ *arena-dead-clause* (*dead-clause-slice arena N i*)›
using *assms* **unfolding** *valid-arena-def* **by** *auto*
have ‹*ia*∈#*dom-m N* ⟹ *ia* ≠ *i* ⟹
ia < *length ?arena* ∧
header-size (*N* ∝ *ia*) ≤ *ia* ∧
xarena-active-clause (*clause-slice ?arena N ia*) (*the* (*fmlookup N ia*))› **for** *ia*
using *dom'*[*of ia*] *clause-slice-swap-lits*[*OF i - dom*, *of ia k l*] *k l*
by *auto*
moreover have ‹*ia* = *i* ⟹
ia < *length ?arena* ∧
header-size (*N* ∝ *ia*) ≤ *ia* ∧
xarena-active-clause (*clause-slice ?arena N ia*)
(*the* (*fmlookup* (*N*(*i* ↪ *swap* (*N* ∝ *i*) *k l*)) *ia*))›
for *ia*
using *dom'*[*of ia*] *clause-slice-swap-lits*[*OF i - dom*, *of ia k l*] *i k l*
xarena-active-clause-swap-lits-same[*OF - - - k l*, *of arena*]
by *auto*
moreover have ‹*ia*∈*vdom* ⟶
ia ∉# *dom-m N* ⟶
MIN-HEADER-SIZE ≤ *ia* ∧ *arena-dead-clause* (*dead-clause-slice* (*swap-lits i k l arena*) (*fmdrop i N*) *ia*)›
for *ia*
using *vdom*[*of ia*] *clause-slice-swap-lits-dead*[*OF i - - arena*, *of ia*] *i k l*
by *auto*
ultimately show *?thesis*
using *i k l arena* **unfolding** *valid-arena-def*
by *auto*
qed

Learning a clause **definition** *append-clause-skeleton* **where**
‹*append-clause-skeleton pos st used lbd C arena* =
(*if is-short-clause C then*
arena @ (*AStatus st used lbd*) #
ASize (*length C* − 2) # *map ALit C*
else arena @ *APos pos* # (*AStatus st used lbd*) #
ASize (*length C* − 2) # *map ALit C*)›

definition *append-clause* **where**
‹*append-clause b C arena* =
append-clause-skeleton 0 (*if b then IRRED else LEARNED*) *0* (*shorten-lbd*(*length C* − 2)) *C arena*›

lemma *arena-active-clause-append-clause*:
assumes
‹*i* ≥ *header-size* (*N* ∝ *i*)› **and**
‹*i* < *length arena*› **and**
‹*xarena-active-clause* (*clause-slice arena N i*) (*the* (*fmlookup N i*))›
shows ‹*xarena-active-clause* (*clause-slice* (*append-clause-skeleton pos st used lbd C arena*) *N i*)
(*the* (*fmlookup N i*))›
proof −
have ‹*drop* (*header-size* (*N* ∝ *i*)) (*clause-slice arena N i*) = *map ALit* (*N* ∝ *i*)› **and**
‹*header-size* (*N* ∝ *i*) ≤ *i*› **and**

```
    ‹i < length arena›
    using assms
    unfolding xarena-active-clause-alt-def
    by auto
   from arg-cong[OF this(1), of length] this(2−)
   have ‹i + length (N ∝ i) ≤ length arena›
    unfolding xarena-active-clause-alt-def
    by (auto simp add: slice-len-min-If header-size-def is-short-clause-def split: if-splits)
  then have ‹clause-slice (append-clause-skeleton pos st used lbd C arena) N i =
    clause-slice arena N i›
    by (auto simp add: append-clause-skeleton-def)
  then show ?thesis
    using assms by simp
qed

lemma length-append-clause[simp]:
  ‹length (append-clause-skeleton pos st used lbd C arena) =
    length arena + length C + header-size C›
  ‹length (append-clause b C arena) = length arena + length C + header-size C›
  by (auto simp: append-clause-skeleton-def header-size-def
    append-clause-def)

lemma arena-active-clause-append-clause-same: ‹2 ≤ length C ⟹ st ≠ DELETED ⟹
    pos ≤ length C − 2 ⟹
    b ⟷ (st = IRRED) ⟹
    xarena-active-clause
     (Misc.slice (length arena) (length arena + header-size C + length C)
       (append-clause-skeleton pos st used lbd C arena))
     (the (fmlookup (fmupd (length arena + header-size C) (C, b) N)
       (length arena + header-size C)))›
  unfolding xarena-active-clause-alt-def append-clause-skeleton-def
  by (cases st)
   (auto simp: header-size-def slice-start0 SHIFTS-def slice-Cons split: if-splits)

lemma clause-slice-append-clause:
  assumes
    ia: ‹ia ∉# dom-m N› ‹ia ∈ vdom› and
    dom: ‹valid-arena arena N vdom› and
    ‹arena-dead-clause (dead-clause-slice (arena) N ia)›
  shows
    ‹arena-dead-clause (dead-clause-slice (append-clause-skeleton pos st used lbd C arena) N ia)›
proof −
  have ia-ge: ‹ia ≥ MIN-HEADER-SIZE› ‹ia < length arena›
    using dom ia unfolding valid-arena-def
    by auto
  then have ‹dead-clause-slice (arena) N ia =
      dead-clause-slice (append-clause-skeleton pos st used lbd C arena) N ia›
    by (auto simp add: extra-information-mark-to-delete-def drop-update-swap
      append-clause-skeleton-def
      arena-dead-clause-def swap-lits-def SHIFTS-def swap-def ac-simps
       Misc.slice-def header-size-def split: if-splits)
  then show ?thesis
    using assms by simp
qed
```

lemma *valid-arena-append-clause-skeleton*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *le-C*: ‹*length C ≥ 2*› **and**
b: ‹*b ⟷ (st = IRRED)*› **and** *st*: ‹*st ≠ DELETED*› **and**
pos: ‹*pos ≤ length C − 2*›
shows ‹*valid-arena* (*append-clause-skeleton pos st used lbd C arena*)
(*fmupd* (*length arena + header-size C*) (*C*, *b*) *N*)
(*insert* (*length arena + header-size C*) *vdom*)›
proof −
let *?arena* = ‹*append-clause-skeleton pos st used lbd C arena*›
let *?i*= ‹*length arena + header-size C*›
let *?N* = ‹(*fmupd* (*length arena + header-size C*) (*C*, *b*) *N*)›
let *?vdom* = ‹*insert* (*length arena + header-size C*) *vdom*›
have
dom: ‹∀ *i*∈#*dom-m N*.
i < length arena ∧
header-size (*N* ∝ *i*) ≤ *i* ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
dom': ‹⋀*i*. *i*∈#*dom-m N* ⟹
i < length arena ∧
header-size (*N* ∝ *i*) ≤ *i* ∧
xarena-active-clause (*clause-slice arena N i*) (*the* (*fmlookup N i*))› **and**
vdom: ‹⋀*i*. *i*∈*vdom* ⟶ *i* ∉# *dom-m N* ⟶ *i* ≤ *length arena* ∧ *MIN-HEADER-SIZE* ≤ *i* ∧
arena-dead-clause (*dead-clause-slice arena N i*)›
using *assms* **unfolding** *valid-arena-def* **by** *auto*
have [*simp*]: ‹*?i* ∉# *dom-m N*›
using *dom'*[*of ?i*]
by *auto*
have ‹*ia*∈#*dom-m N* ⟹
ia < length ?arena ∧
header-size (*N* ∝ *ia*) ≤ *ia* ∧
xarena-active-clause (*clause-slice ?arena N ia*) (*the* (*fmlookup N ia*))› **for** *ia*
using *dom'*[*of ia*] *arena-active-clause-append-clause*[*of N ia arena*]
by *auto*
moreover have ‹*ia* = *?i* ⟹
ia < length ?arena ∧
header-size (*?N* ∝ *ia*) ≤ *ia* ∧
xarena-active-clause (*clause-slice ?arena ?N ia*) (*the* (*fmlookup ?N ia*))› **for** *ia*
using *dom'*[*of ia*] *le-C arena-active-clause-append-clause-same*[*of C st pos b arena used*]
b st pos
by *auto*
moreover have ‹*ia*∈*vdom* ⟶
ia ∉# *dom-m N* ⟶ *ia < length* (*?arena*) ∧
MIN-HEADER-SIZE ≤ *ia* ∧ *arena-dead-clause* (*Misc.slice* (*ia* − *MIN-HEADER-SIZE*) *ia*
(*?arena*))› **for** *ia*
using *vdom*[*of ia*] *clause-slice-append-clause*[*of ia N vdom arena pos st used lbd C*, *OF* - - *arena*]
le-C b st
by *auto*
ultimately show *?thesis*
unfolding *valid-arena-def*
by *auto*
qed

lemma *valid-arena-append-clause*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *le-C*: ‹*length C ≥ 2*›
shows ‹*valid-arena* (*append-clause b C arena*)
(*fmupd* (*length arena + header-size C*) (*C*, *b*) *N*)

(*insert* (*length arena* + *header-size C*) *vdom*)›
using *valid-arena-append-clause-skeleton*[*OF assms*(*1*,*2*),
of b ‹*if b then IRRED else LEARNED*›]
by (*auto simp*: *append-clause-def*)

Refinement Relation

definition *status-rel*:: ‹(*nat* × *clause-status*) *set*› **where**
‹*status-rel* = {(*0*, *IRRED*), (*1*, *LEARNED*), (*3*, *DELETED*)}›

definition *bitfield-rel* **where**
‹*bitfield-rel n* = {(*a*, *b*). *b* ⟷ *a AND* (*2* ^ *n*) > *0*}›

definition *arena-el-relation* **where**
‹*arena-el-relation x el* = (*case el of*
AStatus n b lbd ⇒ (*x AND 0b11*, *n*) ∈ *status-rel* ∧ ((*x AND 0b1100*) >> *2*, *b*) ∈ *nat-rel* ∧ (*x* >> *5*, *lbd*) ∈ *nat-rel*
| *APos n* ⇒ (*x*, *n*) ∈ *nat-rel*
| *ASize n* ⇒ (*x*, *n*) ∈ *nat-rel*
| *ALit n* ⇒ (*x*, *n*) ∈ *nat-lit-rel*
)›

definition *arena-el-rel* **where**
arena-el-rel-interal-def: ‹*arena-el-rel* = {(*x*, *el*). *arena-el-relation x el*}›

lemmas *arena-el-rel-def* = *arena-el-rel-interal-def*[*unfolded arena-el-relation-def*]

Preconditions and Assertions for the refinement

The following lemma expresses the relation between the arena and the clauses and especially shows the preconditions to be able to generate code.

The conditions on *arena-status* are in the direction to simplify proofs: If we would try to go in the opposite direction, we could rewrite ¬ *irred N i* into *arena-status arena i* ≠ *LEARNED*, which is a weaker property.

The inequality on the length are here to enable simp to prove inequalities *Suc 0* < *arena-length arena C* automatically. Normally the arithmetic part can prove it from *2* ≤ *arena-length arena C*, but as this inequality is simplified away, it does not work.

lemma *arena-lifting*:
assumes *valid*: ‹*valid-arena arena N vdom*› **and**
i: ‹*i* ∈# *dom-m N*›
shows
‹*i* ≥ *header-size* (*N* ∝ *i*)› **and**
‹*i* < *length arena*›
‹*is-Size* (*arena* ! (*i* − *SIZE-SHIFT*))›
‹*length* (*N* ∝ *i*) = *arena-length arena i*›
‹*j* < *length* (*N* ∝ *i*) ⟹ *N* ∝ *i* ! *j* = *arena-lit arena* (*i* + *j*)› **and**
‹*j* < *length* (*N* ∝ *i*) ⟹ *is-Lit* (*arena* ! (*i*+*j*))› **and**
‹*j* < *length* (*N* ∝ *i*) ⟹ *i* + *j* < *length arena*› **and**
‹*N* ∝ *i* ! *0* = *arena-lit arena i*› **and**
‹*is-Lit* (*arena* ! *i*)› **and**
‹*i* + *length* (*N* ∝ *i*) ≤ *length arena*› **and**
‹*is-long-clause* (*N* ∝ *i*) ⟹ *is-Pos* (*arena* ! (*i* − *POS-SHIFT*))› **and**
‹*is-long-clause* (*N* ∝ *i*) ⟹ *arena-pos arena i* ≤ *arena-length arena i*› **and**
‹*True*› **and**

‹is-Status (arena ! (i − STATUS-SHIFT))› **and**
‹SIZE-SHIFT ≤ i› **and**
‹LBD-SHIFT ≤ i›
‹True› **and**
‹arena-length arena i ≥ 2› **and**
‹arena-length arena i ≥ Suc 0› **and**
‹arena-length arena i ≥ 0› **and**
‹arena-length arena i > Suc 0› **and**
‹arena-length arena i > 0› **and**
‹arena-status arena i = LEARNED ⟷ ¬irred N i› **and**
‹arena-status arena i = IRRED ⟷ irred N i› **and**
‹arena-status arena i ≠ DELETED› **and**
‹Misc.slice i (i + arena-length arena i) arena = map ALit (N ∝ i)›
proof −
have
dom: ‹⋀i. i∈#dom-m N ⟹
i < length arena ∧
header-size (N ∝ i) ≤ i ∧
xarena-active-clause (clause-slice arena N i) (the (fmlookup N i))›
using *valid* **unfolding** *valid-arena-def*
by *blast+*

have
i-le: ‹i < length arena› **and**
i-ge: ‹header-size (N ∝ i) ≤ i› **and**
xi: ‹xarena-active-clause (clause-slice arena N i) (the (fmlookup N i))›
using *dom*[*OF i*] **by** *fast+*

have
ge2: ‹2 ≤ length (N ∝ i)› **and**
‹header-size (N ∝ i) + length (N ∝ i) = length (clause-slice arena N i)› **and**
pos: ‹is-long-clause (N ∝ i) ⟶
is-Pos (clause-slice arena N i ! (header-size (N ∝ i) − POS-SHIFT)) ∧
xarena-pos (clause-slice arena N i ! (header-size (N ∝ i) − POS-SHIFT))
≤ length (N ∝ i) − 2› **and**
status: ‹is-Status
(clause-slice arena N i ! (header-size (N ∝ i) − STATUS-SHIFT))› **and**
init: ‹(xarena-status
(clause-slice arena N i ! (header-size (N ∝ i) − STATUS-SHIFT)) =
IRRED) =
irred N i› **and**
learned: ‹(xarena-status
(clause-slice arena N i ! (header-size (N ∝ i) − STATUS-SHIFT)) =
LEARNED) =
(¬ irred N i)› **and**
size: ‹is-Size (clause-slice arena N i ! (header-size (N ∝ i) − SIZE-SHIFT))› **and**
size': ‹Suc (Suc (xarena-length
(clause-slice arena N i !
(header-size (N ∝ i) − SIZE-SHIFT)))) =
length (N ∝ i)› **and**
clause: ‹Misc.slice i (i + length (N ∝ i)) arena = map ALit (N ∝ i)›
using *xi i-le i-ge* **unfolding** *xarena-active-clause-alt-def arena-length-def*
by *simp-all*
have [*simp*]:
‹clause-slice arena N i ! (header-size (N ∝ i) − STATUS-SHIFT) =
AStatus (arena-status arena i) (arena-used arena i) (arena-lbd arena i)›

using *size size' i-le i-ge ge2 status size'*
unfolding *header-size-def arena-length-def arena-lbd-def arena-status-def arena-used-def*
by (*auto simp*: *SHIFTS-def slice-nth simp*: *arena-lbd-def*)
have *HH*:
‹*arena-length arena i = length* (*N* ∝ *i*)› **and** ‹*is-Size* (*arena* ! (*i* − *SIZE-SHIFT*))›
using *size size' i-le i-ge ge2 status size' ge2*
unfolding *header-size-def arena-length-def arena-lbd-def arena-status-def*
by (*cases* ‹*arena* ! (*i* − *Suc 0*)›; *auto simp*: *SHIFTS-def slice-nth*; *fail*)+
then show ‹*length* (*N* ∝ *i*) = *arena-length arena i*› **and** ‹*is-Size* (*arena* ! (*i* − *SIZE-SHIFT*))›
using *i-le i-ge size' size ge2 HH* **unfolding** *numeral-2-eq-2*
by (*simp-all split*:)
show ‹*arena-length arena i* ≥ *2*›
‹*arena-length arena i* ≥ *Suc 0*› **and**
‹*arena-length arena i* ≥ *0*› **and**
‹*arena-length arena i* > *Suc 0*› **and**
‹*arena-length arena i* > *0*›
using *ge2* **unfolding** *HH* **by** *auto*
show
‹*i* ≥ *header-size* (*N* ∝ *i*)› **and**
‹*i* < *length arena*›
using *i-le i-ge* **by** *auto*
show *is-lit*: ‹*is-Lit* (*arena* ! (*i*+*j*))› ‹*N* ∝ *i* ! *j* = *arena-lit arena* (*i* + *j*)›
if ‹*j* < *length* (*N* ∝ *i*)›
for *j*
using *arg-cong*[*OF clause*, *of* ‹λ*xs*. *xs* ! *j*›] *i-le i-ge that*
by (*auto simp*: *slice-nth arena-lit-def*)

show *i-le-arena*: ‹*i* + *length* (*N* ∝ *i*) ≤ *length arena*›
using *arg-cong*[*OF clause*, *of length*] *i-le i-ge*
by (*auto simp*: *arena-lit-def slice-len-min-If*)
show ‹*is-Pos* (*arena* ! (*i* − *POS-SHIFT*))› **and**
‹*arena-pos arena i* ≤ *arena-length arena i*›
if ‹*is-long-clause* (*N* ∝ *i*)›
using *pos ge2 i-le i-ge that* **unfolding** *arena-pos-def HH*
by (*auto simp*: *SHIFTS-def slice-nth header-size-def*)
show ‹*True*› **and** ‹*True*› **and**
‹*is-Status* (*arena* ! (*i* − *STATUS-SHIFT*))›
using *ge2 i-le i-ge status* **unfolding** *arena-pos-def*
by (*auto simp*: *SHIFTS-def slice-nth header-size-def*)
show ‹*SIZE-SHIFT* ≤ *i*› **and** ‹*LBD-SHIFT* ≤ *i*›
using *i-ge* **unfolding** *header-size-def SHIFTS-def* **by** (*auto split*: *if-splits*)
show ‹*j* < *length* (*N* ∝ *i*) ⟹ *i* + *j* < *length arena*›
using *i-le-arena* **by** *linarith*
show
‹*N* ∝ *i* ! *0* = *arena-lit arena i*› **and**
‹*is-Lit* (*arena* ! *i*)›
using *is-lit*[*of 0*] *ge2* **by** *fastforce*+
show
‹*arena-status arena i* = *LEARNED* ⟷ ¬*irred N i* ›**and**
‹*arena-status arena i* = *IRRED* ⟷ *irred N i*› **and**
‹*arena-status arena i* ≠ *DELETED*›
using *learned init* **unfolding** *arena-status-def*
by (*auto simp*: *arena-status-def*)
show
‹*Misc.slice i* (*i* + *arena-length arena i*) *arena* = *map ALit* (*N* ∝ *i*)›
apply (*subst list-eq-iff-nth-eq*, *intro conjI allI*)

```
    subgoal
      using HH i-le-arena i-le
      by (auto simp: slice-nth slice-len-min-If)
    subgoal for j
      using HH i-le-arena i-le is-lit[of j]
      by (cases ‹arena ! (i + j)›)
       (auto simp: slice-nth slice-len-min-If
         arena-lit-def)
    done
qed


lemma arena-dom-status-iff:
  assumes valid: ‹valid-arena arena N vdom› and
   i: ‹i ∈ vdom›
  shows
    ‹i ∈# dom-m N ⟷ arena-status arena i ≠ DELETED› (is ‹?eq› is ‹?A ⟷ ?B›) and
    ‹is-Status (arena ! (i − STATUS-SHIFT))› (is ?stat) and
    ‹MIN-HEADER-SIZE ≤ i› (is ?ge)
proof −
  have H1: ?eq ?stat ?ge
    if ‹?A›
  proof −
    have
      ‹xarena-active-clause (clause-slice arena N i) (the (fmlookup N i))› and
      i-ge: ‹header-size (N ∝ i) ≤ i› and
      i-le: ‹i < length arena›
      using assms that unfolding valid-arena-def by blast+
    then have ‹is-Status (clause-slice arena N i ! (header-size (N ∝ i) − STATUS-SHIFT))› and
      ‹(xarena-status (clause-slice arena N i ! (header-size (N ∝ i) − STATUS-SHIFT)) = IRRED) =
       irred N i› and
      ‹(xarena-status (clause-slice arena N i ! (header-size (N ∝ i) − STATUS-SHIFT)) = LEARNED)
=
        (¬ irred N i)›
      unfolding xarena-active-clause-alt-def arena-status-def
      by blast+
    then show ?eq and ?stat and ?ge
      using i-ge i-le that
      unfolding xarena-active-clause-alt-def arena-status-def
      by (auto simp: SHIFTS-def header-size-def slice-nth split: if-splits)
  qed
  moreover have H2: ?eq
    if ‹?B›
  proof −
    have ?A
    proof (rule ccontr)
      assume ‹i ∉# dom-m N›
      then have
        ‹arena-dead-clause (Misc.slice (i − MIN-HEADER-SIZE) i arena)› and
        i-ge: ‹MIN-HEADER-SIZE ≤ i› and
        i-le: ‹i < length arena›
        using assms unfolding valid-arena-def by blast+
      then show False
        using ‹?B›
        unfolding arena-dead-clause-def
        by (auto simp: arena-status-def slice-nth SHIFTS-def)
```

```
    qed
    then show ?eq
      using arena-lifting[OF valid, of i] that
      by auto
  qed
  moreover have ?stat ?ge if ‹¬?A›
  proof −
    have
      ‹arena-dead-clause (Misc.slice (i − MIN-HEADER-SIZE) i arena)› and
      i-ge: ‹MIN-HEADER-SIZE ≤ i› and
      i-le: ‹i < length arena›
      using assms that unfolding valid-arena-def by blast+
    then show ?stat ?ge
      unfolding arena-dead-clause-def
      by (auto simp: SHIFTS-def slice-nth)
  qed
  ultimately show ?eq and ?stat and ?ge
    by blast+
qed

lemma valid-arena-one-notin-vdomD:
  ‹valid-arena M N vdom ⟹ Suc 0 ∉ vdom›
  using arena-dom-status-iff[of M N vdom 1]
  by auto
```

This is supposed to be used as for assertions. There might be a more "local" way to define it, without the need for an existentially quantified clause set. However, I did not find a definition which was really much more useful and more practical.

```
definition arena-is-valid-clause-idx :: ‹arena ⇒ nat ⇒ bool› where
‹arena-is-valid-clause-idx arena i ⟷
  (∃N vdom. valid-arena arena N vdom ∧ i ∈# dom-m N)›
```

This precondition has weaker preconditions is restricted to extracting the status (the other headers can be extracted but only garbage is returned).

```
definition arena-is-valid-clause-vdom :: ‹arena ⇒ nat ⇒ bool› where
‹arena-is-valid-clause-vdom arena i ⟷
  (∃N vdom. valid-arena arena N vdom ∧ (i ∈ vdom ∨ i ∈# dom-m N))›

lemma SHIFTS-alt-def:
  ‹POS-SHIFT = (Suc (Suc (Suc 0)))›
  ‹STATUS-SHIFT = (Suc (Suc 0))›
  ‹SIZE-SHIFT = Suc 0›
  by (auto simp: SHIFTS-def)


definition arena-is-valid-clause-idx-and-access :: ‹arena ⇒ nat ⇒ nat ⇒ bool› where
‹arena-is-valid-clause-idx-and-access arena i j ⟷
  (∃N vdom. valid-arena arena N vdom ∧ i ∈# dom-m N ∧ j < length (N ∝ i))›
```

This is the precondition for direct memory access: $N\ !\ i$ where $i = j + (j - i)$ instead of $N \propto j\ !\ (i - j)$.

```
definition arena-lit-pre where
‹arena-lit-pre arena i ⟷
  (∃j. i ≥ j ∧ arena-is-valid-clause-idx-and-access arena j (i − j))›
```

definition *arena-lit-pre2* **where**
‹*arena-lit-pre2 arena i j* ⟷
(∃ *N vdom*. *valid-arena arena N vdom* ∧ *i* ∈# *dom-m N* ∧ *j* < *length* (*N* ∝ *i*))›

definition *swap-lits-pre* **where**
‹*swap-lits-pre C i j arena* ⟷ *C* + *i* < *length arena* ∧ *C* + *j* < *length arena*›

definition *update-lbd-pre* **where**
‹*update-lbd-pre* = (λ((*C*, *lbd*), *arena*). *arena-is-valid-clause-idx arena C*)›

definition *get-clause-LBD-pre* **where**
‹*get-clause-LBD-pre* = *arena-is-valid-clause-idx*›

Saved position **definition** *get-saved-pos-pre* **where**
‹*get-saved-pos-pre arena C* ⟷ *arena-is-valid-clause-idx arena C* ∧
arena-length arena C > *MAX-LENGTH-SHORT-CLAUSE*›

definition *isa-update-pos-pre* **where**
‹*isa-update-pos-pre* = (λ((*C*, *pos*), *arena*). *arena-is-valid-clause-idx arena C* ∧ *pos* ≥ 2 ∧
pos ≤ *arena-length arena C* ∧ *arena-length arena C* > *MAX-LENGTH-SHORT-CLAUSE*)›

definition *mark-garbage-pre* **where**
‹*mark-garbage-pre* = (λ(*arena*, *C*). *arena-is-valid-clause-idx arena C*)›

lemma *length-clause-slice-list-update*[*simp*]:
‹*length* (*clause-slice* (*arena*[*i* := *x*]) *a b*) = *length* (*clause-slice arena a b*)›
by (*auto simp*: *Misc.slice-def*)

definition *mark-used-raw* **where**
‹*mark-used-raw arena i v* =
arena[*i* − *STATUS-SHIFT* := *AStatus* (*arena-status arena i*) ((*arena-used arena i*) *OR v*) (*arena-lbd arena i*)]›

lemma *length-mark-used-raw*[*simp*]: ‹*length* (*mark-used-raw arena C v*) = *length arena*›
by (*auto simp*: *mark-used-raw-def*)

lemma *valid-arena-mark-used-raw*:
assumes *C*: ‹*C* ∈# *dom-m N*› **and** *valid*: ‹*valid-arena arena N vdom*›
shows
‹*valid-arena* (*mark-used-raw arena C v*) *N vdom*›
proof −
let *?arena* = ‹*mark-used-raw arena C v*›
have *act*: ‹∀ *i*∈#*dom-m N*.
i < *length* (*arena*) ∧
header-size (*N* ∝ *i*) ≤ *i* ∧
xarena-active-clause (*clause-slice arena N i*)
(*the* (*fmlookup N i*))› **and**
dead: ‹⋀*i*. *i* ∈ *vdom* ⟹ *i* ∉# *dom-m N* ⟹ *i* < *length arena* ∧
MIN-HEADER-SIZE ≤ *i* ∧ *arena-dead-clause* (*Misc.slice* (*i* − *MIN-HEADER-SIZE*) *i arena*)›
and
C-ge: ‹*header-size* (*N* ∝ *C*) ≤ *C*› **and**
C-le: ‹*C* < *length arena*› **and**
C-act: ‹*xarena-active-clause* (*clause-slice arena N C*)
(*the* (*fmlookup N C*))›
using *assms*
by (*auto simp*: *valid-arena-def*)

```
  have
    [simp]: ‹clause-slice ?arena N C ! (header-size (N ∝ C) − STATUS-SHIFT) =
          AStatus (xarena-status (clause-slice arena N C ! (header-size (N ∝ C) − STATUS-SHIFT)))
            ((arena-used arena C) OR v) (arena-lbd ?arena C)› and
    [simp]: ‹clause-slice ?arena N C ! (header-size (N ∝ C) − SIZE-SHIFT) =
          clause-slice arena N C ! (header-size (N ∝ C) − SIZE-SHIFT)› and
    [simp]: ‹is-long-clause (N ∝ C) ⟹ clause-slice ?arena N C ! (header-size (N ∝ C) − POS-SHIFT)
=
          clause-slice arena N C ! (header-size (N ∝ C) − POS-SHIFT)› and
    [simp]: ‹length (clause-slice  ?arena N C) = length (clause-slice arena N C)› and
    [simp]: ‹Misc.slice C (C + length (N ∝ C)) ?arena =
      Misc.slice C (C + length (N ∝ C)) arena›
   using C-le C-ge unfolding SHIFTS-def mark-used-raw-def header-size-def arena-lbd-def arena-status-def
    by (auto simp: Misc.slice-def drop-update-swap split: if-splits)

  have ‹xarena-active-clause (clause-slice ?arena N C) (the (fmlookup N C))›
    using C-act C-le C-ge unfolding xarena-active-clause-alt-def
    by simp

  then have 1: ‹xarena-active-clause (clause-slice arena N i) (the (fmlookup N i)) ⟹
     xarena-active-clause (clause-slice ?arena N i) (the (fmlookup N i))›
    if ‹i ∈# dom-m N›
    for i
    using minimal-difference-between-valid-index[of N arena C i, OF act]
      minimal-difference-between-valid-index[of N arena i C, OF act] assms
      that C-ge
    by (cases ‹C < i›; cases ‹C > i›)
      (auto simp: mark-used-raw-def header-size-def STATUS-SHIFT-def
      split: if-splits)

  have 2:
    ‹arena-dead-clause (Misc.slice (i − MIN-HEADER-SIZE) i ?arena)›
    if ‹i ∈ vdom›‹i ∉# dom-m N›‹arena-dead-clause (Misc.slice (i − MIN-HEADER-SIZE) i arena)›
    for i
  proof −
    have i-ge: ‹i ≥ MIN-HEADER-SIZE› ‹i < length arena›
      using that valid unfolding valid-arena-def
      by auto
    show ?thesis
      using dead[of i] that C-le C-ge
      minimal-difference-between-invalid-index[OF valid, of C i]
      minimal-difference-between-invalid-index2[OF valid, of C i]
      by (cases ‹C < i›; cases ‹C > i›)
        (auto simp: mark-used-raw-def header-size-def STATUS-SHIFT-def C
          split: if-splits)
  qed
  show ?thesis
    using 1 2 valid
    by (auto simp: valid-arena-def)
qed


definition mark-unused where
  ‹mark-unused arena i =
  arena[i − STATUS-SHIFT := AStatus (xarena-status (arena!(i − STATUS-SHIFT)))
```

```
    (if (arena-used arena i) > 0 then arena-used arena i − 1 else 0)
      (arena-lbd arena i)]›

lemma length-mark-unused[simp]: ‹length (mark-unused arena C) = length arena›
  by (auto simp: mark-unused-def)

lemma valid-arena-mark-unused:
  assumes C: ‹C ∈# dom-m N› and valid: ‹valid-arena arena N vdom›
  shows
   ‹valid-arena (mark-unused arena C) N vdom›
proof −
  let ?arena = ‹mark-unused arena C› and
     ?used = ‹(if (arena-used arena C) > 0 then arena-used arena C − 1 else 0)›
  have act: ‹∀ i∈#dom-m N.
     i < length (arena) ∧
     header-size (N ∝ i) ≤ i ∧
     xarena-active-clause (clause-slice arena N i)
      (the (fmlookup N i))› and
    dead: ‹⋀i. i ∈ vdom ⟹ i ∉# dom-m N ⟹ i < length arena ∧
         MIN-HEADER-SIZE ≤ i ∧ arena-dead-clause (Misc.slice (i − MIN-HEADER-SIZE) i arena)›
and
    C-ge: ‹header-size (N ∝ C) ≤ C› and
    C-le: ‹C < length arena› and
    C-act: ‹xarena-active-clause (clause-slice arena N C)
      (the (fmlookup N C))›
    using assms
    by (auto simp: valid-arena-def)
  have
   [simp]: ‹clause-slice ?arena N C ! (header-size (N ∝ C) − STATUS-SHIFT) =
         AStatus (xarena-status (clause-slice arena N C ! (header-size (N ∝ C) − STATUS-SHIFT)))
           ?used (arena-lbd arena C)› and
   [simp]: ‹clause-slice ?arena N C ! (header-size (N ∝ C) − SIZE-SHIFT) =
         clause-slice arena N C ! (header-size (N ∝ C) − SIZE-SHIFT)› and
   [simp]: ‹is-long-clause (N ∝ C) ⟹ clause-slice ?arena N C ! (header-size (N ∝ C) − POS-SHIFT)
=
         clause-slice arena N C ! (header-size (N ∝ C) − POS-SHIFT)› and
   [simp]: ‹length (clause-slice  ?arena N C) = length (clause-slice arena N C)› and
   [simp]: ‹Misc.slice C (C + length (N ∝ C)) ?arena =
     Misc.slice C (C + length (N ∝ C)) arena›
    using C-le C-ge unfolding SHIFTS-def mark-unused-def header-size-def
    by (auto simp: Misc.slice-def drop-update-swap split: if-splits)

  have ‹xarena-active-clause (clause-slice ?arena N C) (the (fmlookup N C))›
    using C-act C-le C-ge unfolding xarena-active-clause-alt-def
    by simp

  then have 1: ‹xarena-active-clause (clause-slice arena N i) (the (fmlookup N i)) ⟹
     xarena-active-clause (clause-slice (mark-unused arena C) N i) (the (fmlookup N i))›
    if ‹i ∈# dom-m N›
    for i
    using minimal-difference-between-valid-index[of N arena C i, OF act]
      minimal-difference-between-valid-index[of N arena i C, OF act] assms
      that C-ge
    by (cases ‹C < i›; cases ‹C > i›)
      (auto simp: mark-unused-def header-size-def STATUS-SHIFT-def
      split: if-splits)
```

```
  have 2:
    ‹arena-dead-clause (Misc.slice (i − MIN-HEADER-SIZE) i ?arena)›
    if ‹i ∈ vdom›‹i ∉# dom-m N›‹arena-dead-clause (Misc.slice (i − MIN-HEADER-SIZE) i arena)›
    for i
  proof −
    have i-ge: ‹i ≥ MIN-HEADER-SIZE› ‹i < length arena›
      using that valid unfolding valid-arena-def
      by auto
    show ?thesis
      using dead[of i] that C-le C-ge
      minimal-difference-between-invalid-index[OF valid, of C i]
      minimal-difference-between-invalid-index2[OF valid, of C i]
      by (cases ‹C < i›; cases ‹C > i›)
        (auto simp: mark-unused-def header-size-def STATUS-SHIFT-def C
          split: if-splits)
  qed
  show ?thesis
    using 1 2 valid
    by (auto simp: valid-arena-def)
qed


definition marked-as-used :: ‹arena ⇒ nat ⇒ nat› where
  ‹marked-as-used arena C =  xarena-used (arena ! (C − STATUS-SHIFT))›

definition marked-as-used-pre where
  ‹marked-as-used-pre = arena-is-valid-clause-idx›

lemma valid-arena-vdom-le:
  assumes ‹valid-arena arena N ovdm›
  shows ‹finite ovdm› and ‹card ovdm ≤ length arena›
proof −
  have incl: ‹ovdm ⊆ {MIN-HEADER-SIZE..< length arena}›
    apply auto
    using assms valid-arena-in-vdom-le-arena by blast+
  from card-mono[OF - this] show ‹card ovdm ≤ length arena› by auto
  have ‹length arena ≥ MAX-HEADER-SIZE ∨ ovdm = {}›
    using incl by auto
  with card-mono[OF - incl]  have ‹ovdm ≠ {} ⟹ card ovdm < length arena›
    by auto
  from finite-subset[OF incl] show ‹finite ovdm› by auto
qed


lemma valid-arena-vdom-subset:
  assumes ‹valid-arena arena N (set vdom)› and ‹distinct vdom›
  shows ‹length vdom ≤ length arena›
proof −
  have ‹set vdom ⊆ {0 ..< length arena}›
    using assms by (auto simp: valid-arena-def)
  from card-mono[OF - this] show ?thesis using assms by (auto simp: distinct-card)
qed
```

2.4 MOP versions of operations

2.4.1 Access to literals

definition *mop-arena-lit* **where**
‹*mop-arena-lit arena s = do* {
ASSERT(*arena-lit-pre arena s*);
RETURN (*arena-lit arena s*)
}›

lemma *arena-lit-pre-le-lengthD*: ‹*arena-lit-pre arena C* ⟹ *C* < *length arena*›
apply (*auto simp*: *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*)
using *arena-lifting*(*7*) *nat-le-iff-add* **by** *auto*

definition *mop-arena-lit2* :: ‹*arena* ⇒ *nat* ⇒ *nat* ⇒ *nat literal nres*› **where**
‹*mop-arena-lit2 arena i j = do* {
ASSERT(*arena-lit-pre arena* (*i*+*j*));
let s = *i*+*j*;
RETURN (*arena-lit arena s*)
}›

named-theorems *mop-arena-lit* ‹*Theorems on mop−forms of arena constants*›

lemma *mop-arena-lit-itself*:
‹*mop-arena-lit arena k'* ≤ *SPEC*(λ*c*. (*c*, *N* ∝ *i*!*j*) ∈ *Id*) ⟹ *mop-arena-lit arena k'* ≤ *SPEC*(λ*c*. (*c*, *N* ∝ *i*!*j*) ∈ *Id*)›
‹*mop-arena-lit2 arena i' k'* ≤ *SPEC*(λ*c*. (*c*, *N* ∝ *i*!*j*) ∈ *Id*) ⟹ *mop-arena-lit2 arena i' k'* ≤ *SPEC*(λ*c*. (*c*, *N* ∝ *i*!*j*) ∈ *Id*)›
.

lemma [*mop-arena-lit*]:
assumes *valid*: ‹*valid-arena arena N vdom*› **and**
i: ‹*i* ∈# *dom-m N*›
shows
‹*k* = *i*+*j* ⟹ *j* < *length* (*N* ∝ *i*) ⟹ *mop-arena-lit arena k* ≤ *SPEC*(λ*c*. (*c*, *N* ∝ *i*!*j*) ∈ *Id*)›
‹*i*=*i'* ⟹ *j*=*j'* ⟹*j* < *length* (*N* ∝ *i*) ⟹ *mop-arena-lit2 arena i' j'* ≤ *SPEC*(λ*c*. (*c*, *N* ∝ *i*!*j*) ∈ *Id*)›
using *assms* **apply** (*auto simp*: *arena-lifting mop-arena-lit-def mop-arena-lit2-def Let-def*
intro!: *ASSERT-leI*)
apply (*metis arena-is-valid-clause-idx-and-access-def arena-lifting*(*4*) *arena-lit-pre-def diff-add-inverse le-add1*)+
done

lemma *mop-arena-lit2*[*mop-arena-lit*]:
assumes *valid*: ‹*valid-arena arena N vdom*› **and**
i: ‹(*C*, *C'*) ∈ *nat-rel*› ‹(*i*, *i'*) ∈ *nat-rel*›
shows
‹*mop-arena-lit2 arena C i* ≤ ⇓*Id* (*mop-clauses-at N C' i'*)›
using *assms* **unfolding** *mop-clauses-swap-def mop-arena-lit2-def mop-clauses-at-def*
by *refine-rcg*
(*auto simp*: *arena-lifting valid-arena-swap-lits arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
intro!: *exI*[*of* - *C*])

definition *mop-arena-lit2'* :: ‹*nat set* ⇒ *arena* ⇒ *nat* ⇒ *nat* ⇒ *nat literal nres*› **where**

‹*mop-arena-lit2'* *vdom* = *mop-arena-lit2*›

lemma *mop-arena-lit2'*[*mop-arena-lit*]:
assumes *valid*: ‹*valid-arena arena N vdom*› **and**
i: ‹(*C*, *C'*) ∈ *nat-rel*› ‹(*i*, *i'*) ∈ *nat-rel*›
shows
‹*mop-arena-lit2'* *vdom arena C i* ≤ ⇓*Id* (*mop-clauses-at N C' i'*)›
using *mop-arena-lit2*[*OF assms*]
unfolding *mop-arena-lit2'-def*
.

lemma *arena-lit-pre2-arena-lit*[*dest*]:
‹*arena-lit-pre2 N i j* ⟹ *arena-lit-pre N* (*i*+*j*)›
by (*auto simp*: *arena-lit-pre-def arena-lit-pre2-def arena-is-valid-clause-idx-and-access-def*
intro!: *exI*[*of* - *i*])

2.4.2 Swapping of literals

definition *mop-arena-swap* **where**
‹*mop-arena-swap C i j arena* = *do* {
ASSERT(*swap-lits-pre C i j arena*);
RETURN (*swap-lits C i j arena*)
}›

lemma *mop-arena-swap*[*mop-arena-lit*]:
assumes *valid*: ‹*valid-arena arena N vdom*› **and**
i: ‹(*C*, *C'*) ∈ *nat-rel*› ‹(*i*, *i'*) ∈ *nat-rel*› ‹(*j*, *j'*) ∈ *nat-rel*›
shows
‹*mop-arena-swap C i j arena* ≤ ⇓{(*N'*, *N*). *valid-arena N' N vdom*} (*mop-clauses-swap N C' i' j'*)›
using *assms* **unfolding** *mop-clauses-swap-def mop-arena-swap-def swap-lits-pre-def*
by *refine-rcg*
(*auto simp*: *arena-lifting valid-arena-swap-lits*)

lemma *mop-arena-swap2*:
assumes *valid*: ‹*valid-arena arena N vdom*› **and**
i: ‹(*C*, *C'*) ∈ *nat-rel*› ‹(*i*, *i'*) ∈ *nat-rel*› ‹(*j*, *j'*) ∈ *nat-rel*›
shows
‹*mop-arena-swap C i j arena* ≤ ⇓{(*N'*, *N*). *valid-arena N' N vdom* ∧ *length N'* = *length arena*}
(*mop-clauses-swap N C' i' j'*)›
using *assms* **unfolding** *mop-clauses-swap-def mop-arena-swap-def swap-lits-pre-def*
by *refine-rcg*
(*auto simp*: *arena-lifting valid-arena-swap-lits*)

2.4.3 Position Saving

definition *mop-arena-pos* :: ‹*arena* ⇒ *nat* ⇒ *nat nres*› **where**
‹*mop-arena-pos arena C* = *do* {
ASSERT(*get-saved-pos-pre arena C*);
RETURN (*arena-pos arena C*)
}›

definition *mop-arena-length* :: ‹*arena-el list* ⇒ *nat* ⇒ *nat nres*› **where**
‹*mop-arena-length arena C* = *do* {
ASSERT(*arena-is-valid-clause-idx arena C*);

RETURN (*arena-length arena C*)
}›

2.4.4 Clause length

lemma *mop-arena-length*:
‹(*uncurry mop-arena-length*, *uncurry* (*RETURN oo* (λ*N c*. *length* (*N* ∝ *c*)))) ∈
[λ(*N*, *i*). *i* ∈# *dom-m N*]$_f$ {(*N*, *N'*). *valid-arena N N' vdom*} ×$_f$ *nat-rel* → ⟨*nat-rel*⟩*nres-rel*›
unfolding *mop-arena-length-def*
by (*intro frefI nres-relI*)
(*auto 5 3 intro*!: *ASSERT-leI simp*: *append-ll-def arena-is-valid-clause-idx-def*
arena-lifting)

definition *mop-arena-lbd* **where**
‹*mop-arena-lbd arena C* = *do* {
ASSERT(*get-clause-LBD-pre arena C*);
RETURN(*arena-lbd arena C*)
}›

definition *mop-arena-update-lbd* **where**
‹*mop-arena-update-lbd C glue arena* = *do* {
ASSERT(*update-lbd-pre* ((*C*, *glue*), *arena*));
RETURN(*update-lbd C glue arena*)
}›

definition *mop-arena-status* **where**
‹*mop-arena-status arena C* = *do* {
ASSERT(*arena-is-valid-clause-vdom arena C*);
RETURN(*arena-status arena C*)
}›

definition *mop-marked-as-used* **where**
‹*mop-marked-as-used arena C* = *do* {
ASSERT(*marked-as-used-pre arena C*);
RETURN(*marked-as-used arena C*)
}›

definition *arena-other-watched* :: ‹*arena* ⇒ *nat literal* ⇒ *nat* ⇒ *nat* ⇒ *nat literal nres*› **where**
‹*arena-other-watched S L C i* = *do* {
ASSERT(*i* < *2* ∧ *arena-lit S* (*C* + *i*) = *L* ∧ *arena-lit-pre2 S C i* ∧
arena-lit-pre2 S C (*1*−*i*));
mop-arena-lit2 S C (*1* − *i*)
}›

definition *arena-act-pre* **where**
‹*arena-act-pre* = *arena-is-valid-clause-idx*›

definition *mark-used* :: ‹*arena* ⇒ *nat* ⇒ *arena*› **where**
mark-used-int-def: ‹*mark-used arena C* ≡ *mark-used-raw arena C 1*›

lemmas *mark-used-def* = *mark-used-int-def*[*unfolded mark-used-raw-def*]

lemmas *length-mark-used*[*simp*] =
length-mark-used-raw[*of - - 1*, *unfolded mark-used-int-def*[*symmetric*]]

lemmas *valid-arena-mark-used* =

valid-arena-mark-used-raw[*of* - - - - *1*, *unfolded mark-used-int-def*[*symmetric*]]

definition *mark-used2* :: ‹*arena* ⇒ *nat* ⇒ *arena*› **where**
mark-used2-int-def: ‹*mark-used2 arena C* ≡ *mark-used-raw arena C 2*›

lemmas *mark-used2-def* = *mark-used2-int-def*[*unfolded mark-used-raw-def*]

lemmas *length-mark-used2*[*simp*] =
length-mark-used-raw[*of* - - *2*, *unfolded mark-used2-int-def*[*symmetric*]]

lemmas *valid-arena-mark-used2* =
valid-arena-mark-used-raw[*of* - - - - *2*, *unfolded mark-used2-int-def*[*symmetric*]]

definition *mop-arena-mark-used* **where**
‹*mop-arena-mark-used C arena* = *do* {
ASSERT(*arena-act-pre C arena*);
RETURN (*mark-used C arena*)
}›

definition *mop-arena-mark-used2* **where**
‹*mop-arena-mark-used2 C arena* = *do* {
ASSERT(*arena-act-pre C arena*);
RETURN (*mark-used2 C arena*)
}›

definition *arena-shorten* :: ‹*nat* ⇒ *nat* ⇒ *arena* ⇒ *arena*› **where**
‹*arena-shorten C j N* =
(*if j* > *MAX-LENGTH-SHORT-CLAUSE then N*[*C* − *SIZE-SHIFT* := *ASize* (*j*−*2*), *C* − *POS-SHIFT* := *APos 0*]
else N[*C* − *SIZE-SHIFT* := *ASize* (*j*−*2*)])›

definition *arena-shorten-pre* **where**
‹*arena-shorten-pre C j arena* ⟷ *j* ≥ *2* ∧ *arena-is-valid-clause-idx arena C* ∧
j ≤ *arena-length arena C*›

definition *mop-arena-shorten* **where**
‹*mop-arena-shorten C j arena* = *do* {
ASSERT(*arena-shorten-pre C j arena*);
RETURN (*arena-shorten C j arena*)
}›

lemma *length-arena-shorten*[*simp*]:
‹*length* (*arena-shorten C' j' arena*) = *length arena*›
by (*auto simp*: *arena-shorten-def*)

lemma *valid-arena-arena-shorten*:
assumes *C*: ‹*C* ∈# *dom-m N*› **and**
j: ‹*j* ≤ *arena-length arena C*› **and**
valid: ‹*valid-arena arena N vdom*› **and**
j2: ‹*j* ≥ *2*›
shows ‹*valid-arena* (*arena-shorten C j arena*) (*N*(*C* ↪ *take j* (*N* ∝ *C*))) *vdom*›
proof −
let *?N* = ‹*N*(*C* ↪ *take j* (*N* ∝ *C*))›

have *header*: ‹*header-size* (*take j* (*N* ∝ *C*)) ≤ *header-size* (*N* ∝ *C*)›
by (*auto simp*: *header-size-def*)
let *?arena* = ‹(*if j* > *MAX-LENGTH-SHORT-CLAUSE then*
arena[*C* − *SIZE-SHIFT* := *ASize* (*j*−2), *C* − *POS-SHIFT* := *APos 0*]
else arena[*C* − *SIZE-SHIFT* := *ASize* (*j*−2)])›
have *dead-clause*: ‹*arena-dead-clause* (*Misc.slice* (*i* − 2) *i ?arena*) ⟷
arena-dead-clause (*Misc.slice* (*i* − 2) *i arena*)›
if *i*: ‹*i* ∈ *vdom*› ‹*i* ∉# *dom-m N*›
for *i*
proof −
have [*simp*]: ‹*Misc.slice* (*i* − 2) *i* (*arena*[*C* − *Suc 0* := *ASize* (*j* − 2)]) =
Misc.slice (*i* − 2) *i* (*arena*)›
using *minimal-difference-between-invalid-index*[*OF valid C*, *of i*]
minimal-difference-between-invalid-index2[*OF valid C*, *of i*]
arena-lifting(*1*,*4*,*15*,*18*)[*OF valid C*] *j*
valid-arena-in-vdom-le-arena[*OF valid*, *of i*]
apply −
apply (*subst slice-irrelevant*(*3*))
apply *auto*
by (*metis One-nat-def SIZE-SHIFT-def Suc-le-lessD Suc-pred* ‹⟦*i* ∉# *dom-m N*; *C* ≤ *i*; *i* ∈ *vdom*⟧
⟹ *length* (*N* ∝ *C*) + 2 ≤ *i* − *C*› *add-leD2 diff-diff-cancel diff-le-self le-diff-iff′ nat-le-Suc-less-imp*
that(*1*) *that*(*2*) *verit-comp-simplify1*(*3*))

have [*simp*]:
‹*Misc.slice* (*i* − 2) *i* (*arena*[*C* − *SIZE-SHIFT* := *ASize* (*j* − 2), *C* − *POS-SHIFT* := *APos 0*]) =
Misc.slice (*i* − 2) *i arena*› **if** *j5*: ‹*j* > *MAX-LENGTH-SHORT-CLAUSE*›
using *minimal-difference-between-invalid-index*[*OF valid C*, *of i*]
minimal-difference-between-invalid-index2[*OF valid C*, *of i*]
arena-lifting(*1*,*4*,*15*,*18*)[*OF valid C*] *j that i*
valid-arena-in-vdom-le-arena[*OF valid*, *of i*] *j*
apply −
apply (*subst slice-irrelevant*(*3*))
apply (*auto simp*: *SHIFTS-def not-less-eq header-size-def linorder-class.not-le*
split: *if-splits*)
by (*metis diff-diff-cancel diff-le-self le-diff-iff′ less-or-eq-imp-le numeral-3-eq-3 verit-comp-simplify1*(*3*))
show *?thesis*
using *that*
using *minimal-difference-between-invalid-index*[*OF valid C*, *of i*]
minimal-difference-between-invalid-index2[*OF valid C*, *of i*]
arena-lifting(*1*,*4*,*15*,*18*)[*OF valid C*] *j*
valid-arena-in-vdom-le-arena[*OF valid*, *of i*]
apply −
apply (*simp split*: *if-splits*, *intro conjI impI*)
subgoal
apply (*subst slice-irrelevant*(*3*))
apply (*cases* ‹*C* < *i*›)
apply (*auto simp*: *arena-dead-clause-def not-less-eq*
SHIFTS-def header-size-def)
by (*metis less-Suc-eq nat-le-Suc-less-imp*)
done
qed
have *other-active*: ‹*i* ≠ *C* ⟹
i ∈# *dom-m N* ⟹
xarena-active-clause (*clause-slice* (*?arena*) *N i*)
(*the* (*fmlookup N i*)) ⟷
xarena-active-clause (*clause-slice* (*arena*) *N i*)

```
    (the (fmlookup N i))› for i
    using
      arena-lifting(1,4,15,18)[OF valid C] j
      arena-lifting(18)[OF valid, of i]
      valid-minimal-difference-between-valid-index[OF valid C, of i]
      valid-minimal-difference-between-valid-index[OF valid - C, of i]
    apply −
    apply (simp split: if-splits, intro conjI impI)
    apply (subst slice-irrelevant(3))
    subgoal
      by (cases ‹C < i›)
       (auto simp: arena-dead-clause-def arena-lifting SHIFTS-def not-less-eq
        header-size-def split: if-splits)
    subgoal
      by (cases ‹C < i›)
        (auto simp: arena-dead-clause-def arena-lifting SHIFTS-def not-less-eq
        header-size-def split: if-splits)
    subgoal
      by (cases ‹C < i›)
        (auto simp: arena-dead-clause-def arena-lifting SHIFTS-def not-less-eq
        header-size-def split: if-splits)
   done
 have [simp]:
   ‹Misc.slice C (C + arena-length arena C) arena = map ALit (N ∝ C) ⟹ Misc.slice C (C + j)
arena = map ALit (take j (N ∝ C))›
   by (drule arg-cong[of - - ‹take j›])
    (use j j2 arena-lifting[OF valid C] in ‹auto simp: Misc.slice-def take-map›)

  have arena2: ‹xarena-active-clause (clause-slice arena N C) (the (fmlookup N C)) ⟹
    xarena-active-clause (clause-slice ?arena ?N C)
    (take j (N ∝ C), irred N C)›
    using j j2 arena-lifting[OF valid C] header
    apply (subst (asm) xarena-active-clause-alt-def)
    apply (subst xarena-active-clause-def)
    apply simp
    apply (intro conjI impI)
    apply (simp add: slice-head SHIFTS-def header-size-def
      slice-len-min-If slice-nth; fail)+
    done

  show ?thesis
    using assms header distinct-mset-dom[of N] arena2 other-active dead-clause
    unfolding valid-arena-def arena-shorten-def
    by (auto dest!: multi-member-split simp: add-mset-eq-add-mset)
qed

lemma mop-arena-shorten-spec:
  assumes C: ‹C ∈# dom-m N› and
    j: ‹j ≤ arena-length arena C› and
    valid: ‹valid-arena arena N vdom› and
    j2: ‹j ≥ 2› and
    ‹(C,C')∈nat-rel› ‹(j,j')∈nat-rel›
  shows ‹mop-arena-shorten C j arena ≤ SPEC(λc. (c, N(C' ↪ take j' (N ∝ C'))) ∈
      {(arena', N). valid-arena arena' N vdom ∧ length arena = length arena'})›
  unfolding mop-arena-shorten-def
  apply refine-vcg
```

```
  subgoal
    using assms
    unfolding arena-shorten-pre-def arena-is-valid-clause-idx-def by auto
  subgoal
    using assms
    by (auto intro!: valid-arena-arena-shorten)
  done

definition arenap-update-lit :: ‹nat ⇒ nat ⇒ nat literal ⇒ arena ⇒ arena› where
  ‹arenap-update-lit C j L N = N[C + j := ALit L]›

lemma length-arenap-update-lit[simp]: ‹length (arenap-update-lit C j L arena) = length arena›
  by (auto simp: arenap-update-lit-def)

lemma valid-arena-arenap-update-lit:
  assumes C: ‹C ∈# dom-m N› and
    j: ‹j < arena-length arena C› and
    valid: ‹valid-arena arena N vdom›
  shows ‹valid-arena (arenap-update-lit C j  L arena) (N(C ↪ (N ∝ C)[j := L])) vdom›
proof −
  let ?N = ‹N(C ↪ (N ∝ C)[j := L])›
  have header[simp]: ‹header-size (?N ∝ C) = header-size (N ∝ C)›
    by (auto simp: header-size-def)
  let ?arena = ‹arenap-update-lit C j L arena›
  have dead-clause: ‹arena-dead-clause (Misc.slice (i − 2) i ?arena) ⟷
    arena-dead-clause (Misc.slice (i − 2) i arena)›
    if i: ‹i ∈ vdom› ‹i ∉# dom-m N›
    for i
  proof −
    have [simp]: ‹Misc.slice (i − 2) i (arena[C − Suc 0 := ASize (j − 2)]) =
      Misc.slice (i − 2) i (arena)›
      using minimal-difference-between-invalid-index[OF valid C, of i]
        minimal-difference-between-invalid-index2[OF valid C, of i]
        arena-lifting(1,4,15,18)[OF valid C] j
        valid-arena-in-vdom-le-arena[OF valid, of i]
      apply −
      apply (subst slice-irrelevant(3))
      apply auto
     by (metis One-nat-def SIZE-SHIFT-def Suc-le-lessD Suc-pred ‹⟦i ∉# dom-m N; C ≤ i; i ∈ vdom⟧
⟹ length (N ∝ C) + 2 ≤ i − C› add-leD2 diff-diff-cancel diff-le-self le-diff-iff' nat-le-Suc-less-imp
that(1) that(2) verit-comp-simplify1(3))

    have [simp]:
      ‹Misc.slice (i − 2) i ?arena =
      Misc.slice (i − 2) i arena›
      using minimal-difference-between-invalid-index[OF valid C, of i]
        minimal-difference-between-invalid-index2[OF valid C, of i]
        arena-lifting(1,4,15,18)[OF valid C] j that i
        valid-arena-in-vdom-le-arena[OF valid, of i] j
      unfolding arenap-update-lit-def
      apply −
      apply (subst slice-irrelevant(3))
      apply (auto simp: SHIFTS-def not-less-eq header-size-def linorder-class.not-le
        split: if-splits)
      apply linarith
      apply linarith
```

```
    done
  show ?thesis
    using that
    using minimal-difference-between-invalid-index[OF valid C, of i]
      minimal-difference-between-invalid-index2[OF valid C, of i]
      arena-lifting(1,4,15,18)[OF valid C] j
      valid-arena-in-vdom-le-arena[OF valid, of i]
    unfolding arenap-update-lit-def
    apply −
    apply (subst slice-irrelevant(3))
    apply (cases ‹C < i›)
    apply (auto simp: arena-dead-clause-def not-less-eq
      SHIFTS-def header-size-def)
   done
qed
have other-active: ‹i ≠ C ⟹
  i ∈# dom-m N ⟹
  xarena-active-clause (clause-slice (?arena) N i)
  (the (fmlookup N i)) ⟷
  xarena-active-clause (clause-slice (arena) N i)
  (the (fmlookup N i))› for i
  using
    arena-lifting(1,4,15,18)[OF valid C] j
    arena-lifting(18)[OF valid, of i]
    valid-minimal-difference-between-valid-index[OF valid C, of i]
    valid-minimal-difference-between-valid-index[OF valid - C, of i]
  unfolding arenap-update-lit-def
  apply −
  apply (subst slice-irrelevant(3))
  subgoal
    by (cases ‹C < i›)
     (auto simp: arena-dead-clause-def arena-lifting SHIFTS-def not-less-eq
      header-size-def split: if-splits)
  subgoal
    by (cases ‹C < i›)
      (auto simp: arena-dead-clause-def arena-lifting SHIFTS-def not-less-eq
      header-size-def split: if-splits)
 done
have [simp]: ‹header-size ((N ∝ C)[j := L]) = header-size (N ∝ C)›
  by (auto simp: header-size-def)
have [simp]:
  ‹Misc.slice C (C + arena-length arena C) arena = map ALit (N ∝ C) ⟹
   drop (header-size (N ∝ C)) ((Misc.slice (C − header-size (N ∝ C)) (C + arena-length arena C)
arena)[j + header-size (N ∝ C) := ALit L]) =
  map ALit ((N ∝ C)[j := L])›
  by (drule arg-cong[of - - ‹ λxs. xs [j := ALit L]›])
    (use j arena-lifting(1−4)[OF valid C] in ‹auto simp: drop-update-swap map-update›)

 have arena2: ‹xarena-active-clause (clause-slice arena N C) (the (fmlookup N C)) ⟹
   xarena-active-clause (clause-slice ?arena ?N C)
   ((?N ∝ C), irred N C)›
   using j arena-lifting[OF valid C] header
   unfolding arenap-update-lit-def
   apply (subst (asm) xarena-active-clause-alt-def)
   apply (subst xarena-active-clause-def)
   apply (subst prod.simps)
```

apply *simp*
apply (*intro conjI impI*)
apply (*simp add*: *slice-head SHIFTS-def header-size-def*
slice-len-min-If slice-nth; *fail*)+
done
show *?thesis*
using *assms distinct-mset-dom*[*of N*] *arena2 other-active dead-clause*
unfolding *valid-arena-def arena-shorten-def*
by (*auto dest*!: *multi-member-split simp*: *add-mset-eq-add-mset*)
qed

definition *mop-arena-update-lit* :: ‹*nat* ⇒ *nat* ⇒ *nat literal* ⇒ *arena* ⇒ *arena nres*› **where**
‹*mop-arena-update-lit C j L arena* = *do* {
ASSERT(*arena-lit-pre2 arena C j*);
RETURN (*arenap-update-lit C j L arena*)
}›

lemma *mop-arena-update-lit-spec*:
assumes *C*: ‹*C* ∈# *dom-m N*› **and**
j: ‹*j* < *arena-length arena C*› **and**
valid: ‹*valid-arena arena N vdom*› **and**
‹(*j*,*j'*) ∈ *nat-rel*› **and**
‹(*C*,*C'*) ∈ *nat-rel*› **and**
‹(*L*,*L'*) ∈ *Id*›
shows
‹*mop-arena-update-lit C j L arena* ≤ *SPEC* (λ*c*. (*c*, (*N*(*C'* ↪ (*N* ∝ *C'*)[*j'* := *L'*]))) ∈
{(*arena'*, *N*). *valid-arena arena' N vdom* ∧ *length arena'* = *length arena*}›
unfolding *mop-arena-update-lit-def*
apply *refine-vcg*
subgoal using *assms* **unfolding** *arena-lit-pre2-def*
by (*auto simp*: *arena-lifting*)
subgoal using *assms* **by** (*auto intro*!: *valid-arena-arenap-update-lit*)
done

definition *arena-set-status* **where**
‹*arena-set-status arena C b*= *do* {
(*arena*[*C* − *STATUS-SHIFT* := *AStatus b* (*arena-used arena C*) (*arena-lbd arena C*)])
}›

lemma *length-arena-set-status*[*simp*]:
‹*length* (*arena-set-status arena C b*) = *length arena*›
by (*auto simp*: *arena-set-status-def*)

lemma *header-size-Suc-def*:
‹*header-size C* =
(*if is-short-clause C then* (*Suc* (*Suc 0*)) *else* (*Suc* (*Suc* (*Suc 0*))))›
unfolding *header-size-def*
by *auto*

lemma *valid-arena-arena-set-status*:
assumes
valid: ‹*valid-arena arena N vdm*› **and**
C: ‹*C* ∈# *dom-m N*› **and**
b: ‹*b* = *IRRED* ∨ *b* = *LEARNED*› **and**
b': ‹*b'* ⟷ *b* = *IRRED*›
shows ‹*valid-arena* (*arena-set-status arena C b*) (*fmupd C* (*N* ∝ *C*, *b'*) *N*) *vdm*›

proof −
have [*simp*]: ‹*i* − 2 ≤ *length arena*› **and**
[*simp*]: ‹*C* − 2 = *i* − 2 ⟷ *C* =*i*› **if** ‹*i* ∈ *vdm*› **for** *i*
apply (*meson less-imp-diff-less less-imp-le-nat that valid valid-arena-def*)
by (*metis C STATUS-SHIFT-def add-diff-inverse-nat arena-lifting*(16) *that valid valid-arena-in-vdom-le-arena*(2) *verit-comp-simplify1*(3))
have [*iff*]: ‹*arena-dead-clause* (*Misc.slice* (*i* − 2) *i* (*arena-set-status arena C b*)) ⟷
arena-dead-clause (*Misc.slice* (*i* − 2) *i arena*)›
if ‹*i* ∉# *dom-m N*› **and** ‹*i* ∈ *vdm*› **for** *i*
using *minimal-difference-between-invalid-index2*[*OF valid C that*(1) - *that*(2)]
minimal-difference-between-invalid-index[*OF valid C that*(1) - *that*(2)]
that
by (*cases* ‹*i* < *C*›)
(*auto simp*: *extra-information-mark-to-delete-def drop-update-swap*
arena-dead-clause-def SHIFTS-def arena-set-status-def ac-simps nth-list-update′ nth-drop
Misc.slice-def header-size-def split: *if-splits*)

have [*simp*]: ‹*header-size* (*N* ∝ *C*) − *POS-SHIFT* < *C* + *length* (*N* ∝ *C*) − (*C* − *header-size* (*N* ∝ *C*))›
‹*header-size* (*N* ∝ *C*) − *STATUS-SHIFT* < *C* + *length* (*N* ∝ *C*) − (*C* − *header-size* (*N* ∝ *C*))›
‹*header-size* (*N* ∝ *C*) − *SIZE-SHIFT* < *C* + *length* (*N* ∝ *C*) − (*C* − *header-size* (*N* ∝ *C*))›
apply (*smt* (*verit*, *ccfv-threshold*) *C add.right-neutral add-diff-inverse-nat diff-is-0-eq′ le-zero-eq*
length-greater-0-conv less-diff-conv less-imp-diff-less list.size(3) *nat.simps*(3)
nat-add-left-cancel-less numeral-2-eq-2 valid valid-arena-def xarena-active-clause-alt-def)
apply (*smt* (*verit*, *ccfv-SIG*) *C Nat.diff-add-assoc2 STATUS-SHIFT-def arena-lifting*(1)
arena-shift-distinct(16) *diff-diff-cancel diff-is-0-eq nat.simps*(3) *nat-le-linear*
not-add-less1 not-gr0 numeral-2-eq-2 valid zero-less-diff)
using *C SIZE-SHIFT-def arena-lifting*(1) *valid verit-comp-simplify1*(3) **by** *fastforce*

have [*iff*]: ‹*C* − *header-size* (*N* ∝ *C*) ≤ *length arena*›
by (*meson C arena-lifting*(2) *less-imp-diff-less less-imp-le-nat valid*)
have ‹*C* ≥ *header-size* (*N* ∝ *C*)› ‹*C* < *length arena*›
using *arena-lifting*[*OF valid C*] **by** *auto*
then have [*iff*]: ‹*C* − *LBD-SHIFT* < *length arena*›
‹*C* − *SIZE-SHIFT* < *length arena*›
‹*is-long-clause* (*N* ∝ *C*) ⟹ *header-size* (*N* ∝ *C*) ≥ *POS-SHIFT*› **and**
[*simp*]: ‹*C* − *header-size* (*N* ∝ *C*) + *header-size* (*N* ∝ *C*) = *C*›
by (*auto simp*: *LBD-SHIFT-def SIZE-SHIFT-def header-size-def POS-SHIFT-def split*: *if-splits*)

have [*simp*]: ‹*C* − *header-size* (*N* ∝ *C*) + (*header-size* (*N* ∝ *C*) − *LBD-SHIFT*) = *C* − *LBD-SHIFT*›
‹*C* − *header-size* (*N* ∝ *C*) + (*header-size* (*N* ∝ *C*) − *SIZE-SHIFT*) = *C* − *SIZE-SHIFT*›
‹*is-long-clause* (*N* ∝ *C*) ⟹ *C* − *header-size* (*N* ∝ *C*) + *header-size* (*N* ∝ *C*) − *POS-SHIFT* = *C* − *POS-SHIFT*›
apply (*smt* (*verit*, *best*) *C Nat.add-diff-assoc2 add.right-neutral add-diff-cancel-right*
add-diff-inverse-nat arena-lifting(1) *arena-shift-distinct*(16) *diff-is-0-eq′ less-imp-le-nat*
order-mono-setup.refl valid)
apply (*metis Nat.diff-add-assoc One-nat-def SIZE-SHIFT-def STATUS-SHIFT-def* ‹*header-size* (*N* ∝ *C*) ≤ *C*›
arena-shift-distinct(10) *diff-is-0-eq le-add-diff-inverse2 lessI less-or-eq-imp-le nat-le-linear numeral-2-eq-2*)
using *SHIFTS-alt-def*(1) *header-size-def* **by** *presburger*
have [*iff*]: ‹*C* − *LBD-SHIFT* = *C* − *SIZE-SHIFT* ⟷ *False*›
‹*is-long-clause* (*N* ∝ *C*) ⟹ *C* − *LBD-SHIFT* = *C* − *POS-SHIFT* ⟷ *False*›
‹*C* − *LBD-SHIFT* < *C*›
apply (*metis* ‹*header-size* (*N* ∝ *C*) ≤ *C*› *arena-shift-distinct*(10))

using ‹*header-size* (*N* ∝ *C*) ≤ *C*› *arena-shift-distinct*(*13*) **apply** *presburger*
by (*metis STATUS-SHIFT-def* ‹*header-size* (*N* ∝ *C*) ≤ *C*› *diff-less header-size-Suc-def le-zero-eq nat.simps*(*3*) *not-gr0 numeral-2-eq-2*)

let *?s* = ‹*clause-slice* (*arena-set-status arena C b*) *N C*›
let *?t* = ‹*clause-slice arena N C*›
have [*simp*]: ‹*is-Pos* (*?s* ! (*header-size* (*N* ∝ *C*) − *POS-SHIFT*)) = *is-Pos* (*?t* ! (*header-size* (*N* ∝ *C*) − *POS-SHIFT*))›
‹*is-Status* (*?s* ! (*header-size* (*N* ∝ *C*) − *STATUS-SHIFT*))›
‹*xarena-status* (*?s* ! (*header-size* (*N* ∝ *C*) − *LBD-SHIFT*)) = *b*›
‹*is-Size* (*?s* ! (*header-size* (*N* ∝ *C*) − *SIZE-SHIFT*)) = *is-Size* (*?t* ! (*header-size* (*N* ∝ *C*) − *SIZE-SHIFT*))›
‹*xarena-length* (*?s* ! (*header-size* (*N* ∝ *C*) − *SIZE-SHIFT*)) = *xarena-length* (*?t* ! (*header-size* (*N* ∝ *C*) − *SIZE-SHIFT*))›
‹*is-long-clause* (*N* ∝ *C*) ⟹ *xarena-pos* (*?s* ! (*header-size* (*N* ∝ *C*) − *POS-SHIFT*)) = *xarena-pos* (*?t* ! (*header-size* (*N* ∝ *C*) − *POS-SHIFT*))›
‹*length ?s* = *length ?t*›
‹*Misc.slice C* (*C* + *length* (*N* ∝ *C*)) (*arena-set-status arena C b*) = *Misc.slice C* (*C* + *length* (*N* ∝ *C*)) *arena*›
apply (*auto simp*: *arena-set-status-def Misc.slice-def nth-list-update'*)
apply (*metis C arena-el.distinct-disc*(*11*) *arena-lifting*(*14*) *valid*)
done
have ‹*xarena-active-clause* (*clause-slice arena N C*) (*the* (*fmlookup N C*))›
using *assms*(*1*,*2*) **unfolding** *valid-arena-def* **by** (*auto dest*!: *multi-member-split*)
then have [*simp*]: ‹*xarena-active-clause* (*clause-slice* (*arena-set-status arena C b*) *N C*) (*N* ∝ *C*, *b'*)›
using *b' b* **unfolding** *xarena-active-clause-def case-prod-beta*
by (*auto simp*: *xarena-active-clause-def*)
have [*simp*]: ‹(*clause-slice* (*arena-set-status arena C b*) *N i*) = (*clause-slice arena N i*)›
if ‹*C* ≠ *i*› **and** ‹*i* ∈# *dom-m N*› **for** *i*
using
valid-minimal-difference-between-valid-index[*OF valid that*(*2*) *C*]
valid-minimal-difference-between-valid-index[*OF valid C that*(*2*)]
that
apply (*cases* ‹*C* > *i*›)
apply (*auto simp*: *Misc.slice-def arena-set-status-def*)
apply (*subst drop-update-swap*)
using ‹*C* − *header-size* (*N* ∝ *C*) + (*header-size* (*N* ∝ *C*) − *LBD-SHIFT*) = *C* − *LBD-SHIFT*› **apply** *linarith*
apply (*subst take-update-cancel*)
using ‹*C* − *header-size* (*N* ∝ *C*) + (*header-size* (*N* ∝ *C*) − *LBD-SHIFT*) = *C* − *LBD-SHIFT*› **apply** *linarith*
apply *auto*
apply (*subst drop-upd-irrelevant*)
using ‹*C* − *LBD-SHIFT* < *C*› **apply** *linarith*
apply *auto*
done

show *?thesis*
using *assms*(*1*,*2*)
unfolding *valid-arena-def*
by *auto*
qed

definition *mop-arena-set-status* **where**
‹*mop-arena-set-status arena C b*= *do* {
ASSERT(*arena-is-valid-clause-idx arena C*);

RETURN(*arena-set-status arena C b*)
}›

lemma *mop-arena-status2*:
assumes ‹(*C*,*C'*)∈*nat-rel*› ‹*C* ∈ *vdom*›
‹*valid-arena arena N vdom*›
shows
‹*mop-arena-status arena C*
≤ *SPEC*
(λ*c*. (*c*, *C* ∈# *dom-m N*)
∈ {(*a*,*b*). (*b* ⟶ (*a* = *IRRED* ⟷ *irred N C*) ∧ (*a* = *LEARNED* ⟷ ¬*irred N C*)) ∧ (*a* = *DELETED* ⟷ ¬*b*)})›
using *assms arena-dom-status-iff*[*of arena N vdom C*] **unfolding** *mop-arena-status-def*
by (*cases* ‹*C* ∈# *dom-m N*›)
(*auto intro*!: *ASSERT-leI simp*: *arena-is-valid-clause-vdom-def*
arena-lifting)

lemma *mop-arena-status3*:
assumes ‹(*C*,*C'*)∈*nat-rel*› ‹*C* ∈# *dom-m N*›
‹*valid-arena arena N vdom*›
shows
‹*mop-arena-status arena C*
≤ *SPEC*
(λ*c*. (*c*, *irred N C*)
∈ {(*a*,*b*). (*a* = *IRRED* ⟷ *irred N C*) ∧ (*a* = *LEARNED* ⟷ ¬*irred N C*) ∧ *b* = (*irred N C*)∧ (*a* ≠ *DELETED*)})›
using *assms arena-dom-status-iff*[*of arena N vdom C*] **unfolding** *mop-arena-status-def*
by (*auto intro*!: *ASSERT-leI simp*: *arena-is-valid-clause-vdom-def*
arena-lifting)

lemma *mop-arena-status-vdom*:
assumes ‹*C* ∈ *vdom*› **and** ‹(*C*,*C'*)∈*nat-rel*›
‹*valid-arena arena N vdom*›
shows
‹*mop-arena-status arena C*
≤ *SPEC*
(λ*c*. (*c*, *C'* ∈# *dom-m N*)
∈ {(*a*,*b*). (*a* ≠ *DELETED* ⟷ *b*) ∧ (((*a* = *IRRED* ⟷ (*irred N C'* ∧ *b*)) ∧ (*a* = *LEARNED* ⟷ (¬*irred N C'* ∧ *b*))))})›
using *assms arena-lifting*(*24*,*25*)[*of arena N vdom C*] *arena-dom-status-iff*(*1*)[*of arena N vdom C*]
unfolding *mop-arena-status-def*
by (*cases* ‹*arena-status arena C'*›)
(*auto intro*!: *ASSERT-leI simp*: *arena-is-valid-clause-vdom-def*)

2.5 Virtual Domain

2.6 Virtual domain

The virtual domain is composed of the addressable (and accessible) elements, i.e., the domain and all the deleted clauses that are still present in the watch lists.

definition *vdom-m* :: ‹*nat multiset* ⇒ (*nat literal* ⇒ (*nat* × -) *list*) ⇒ (*nat*, *'b*) *fmap* ⇒ *nat set*› **where**
‹*vdom-m* 𝒜 *W N* = ⋃(((‘) *fst*) ‘ *set* ‘ *W* ‘ *set-mset* (ℒ$_{all}$ 𝒜)) ∪ *set-mset* (*dom-m N*)›

lemma *vdom-m-simps*[*simp*]:
‹*bh* ∈# *dom-m N* ⟹ *vdom-m* 𝒜 *W* (*N*(*bh* ↪ *C*)) = *vdom-m* 𝒜 *W N*›
‹*bh* ∉# *dom-m N* ⟹ *vdom-m* 𝒜 *W* (*N*(*bh* ↪ *C*)) = *insert bh* (*vdom-m* 𝒜 *W N*)›
by (*force simp*: *vdom-m-def split*: *if-splits*)+

lemma *vdom-m-simps2*[*simp*]:
‹*i* ∈# *dom-m N* ⟹ *vdom-m* 𝒜 (*W*(*L* := *W L* @ [(*i*, *C*)])) *N* = *vdom-m* 𝒜 *W N*›
‹*bi* ∈# *dom-m ax* ⟹ *vdom-m* 𝒜 (*bp*(*L*:= *bp L* @ [(*bi*, *av*′)])) *ax* = *vdom-m* 𝒜 *bp ax*›
by (*force simp*: *vdom-m-def split*: *if-splits*)+

lemma *vdom-m-simps3*[*simp*]:
‹*fst biav*′ ∈# *dom-m ax* ⟹ *vdom-m* 𝒜 (*bp*(*L*:= *bp L* @ [*biav*′])) *ax* = *vdom-m* 𝒜 *bp ax*›
by (*cases biav*′; *auto simp*: *dest*: *multi-member-split*[*of L*] *split*: *if-splits*)

What is the difference with the next lemma?

lemma [*simp*]:
‹*bf* ∈# *dom-m ax* ⟹ *vdom-m* 𝒜 *bj* (*ax*(*bf* ↪ *C*′)) = *vdom-m* 𝒜 *bj* (*ax*)›
by (*force simp*: *vdom-m-def split*: *if-splits*)+

lemma *vdom-m-simps4*[*simp*]:
‹*i* ∈# *dom-m N* ⟹
vdom-m 𝒜 (*W* (*L1* := *W L1* @ [(*i*, *C1*)], *L2* := *W L2* @ [(*i*, *C2*)])) *N* = *vdom-m* 𝒜 *W N*›
by (*auto simp*: *vdom-m-def image-iff dest*: *multi-member-split split*: *if-splits*)

This is *?i* ∈# *dom-m ?N* ⟹ *vdom-m ?*𝒜 (*?W*(*?L1.0* := *?W ?L1.0* @ [(*?i*, *?C1.0*)], *?L2.0* := *?W ?L2.0* @ [(*?i*, *?C2.0*)])) *?N* = *vdom-m ?*𝒜 *?W ?N* if the assumption of distinctness is not present in the context.

lemma *vdom-m-simps4*′[*simp*]:
‹*i* ∈# *dom-m N* ⟹
vdom-m 𝒜 (*W* (*L1* := *W L1* @ [(*i*, *C1*), (*i*, *C2*)])) *N* = *vdom-m* 𝒜 *W N*›
by (*auto simp*: *vdom-m-def image-iff dest*: *multi-member-split split*: *if-splits*)

We add a spurious dependency to the parameter of the locale:

definition *empty-watched* :: ‹*nat multiset* ⇒ *nat literal* ⇒ (*nat* × *nat literal* × *bool*) *list*› **where**
‹*empty-watched* 𝒜 = (λ-. [])›

lemma *vdom-m-empty-watched*[*simp*]:
‹*vdom-m* 𝒜 (*empty-watched* 𝒜′) *N* = *set-mset* (*dom-m N*)›
by (*auto simp*: *vdom-m-def empty-watched-def*)

The following rule makes the previous one not applicable. Therefore, we do not mark this lemma as simp.

lemma *vdom-m-simps5*:
‹*i* ∉# *dom-m N* ⟹ *vdom-m* 𝒜 *W* (*fmupd i C N*) = *insert i* (*vdom-m* 𝒜 *W N*)›
by (*force simp*: *vdom-m-def image-iff dest*: *multi-member-split split*: *if-splits*)

lemma *in-watch-list-in-vdom*:
assumes ‹*L* ∈# $\mathcal{L}_{all}$ 𝒜› **and** ‹*w* < *length* (*watched-by S L*)›
shows ‹*fst* (*watched-by S L* ! *w*) ∈ *vdom-m* 𝒜 (*get-watched-wl S*) (*get-clauses-wl S*)›
using *assms*
unfolding *vdom-m-def*
by (*cases S*) (*auto dest*: *multi-member-split*)

lemma *in-watch-list-in-vdom*′:

assumes ‹$L \in\# \mathcal{L}_{all}\ \mathcal{A}$› **and** ‹$A \in$ *set* (*watched-by S L*)›
shows ‹*fst A* $\in$ *vdom-m* $\mathcal{A}$ (*get-watched-wl S*) (*get-clauses-wl S*)›
using *assms*
unfolding *vdom-m-def*
by (*cases S*) (*auto dest*: *multi-member-split*)

lemma *in-dom-in-vdom*[*simp*]:
‹$x \in\#$ *dom-m N* $\Longrightarrow$ $x \in$ *vdom-m* $\mathcal{A}$ *W N*›
unfolding *vdom-m-def*
by (*auto dest*: *multi-member-split*)

lemma *in-vdom-m-upd*:
‹*x1f* $\in$ *vdom-m* $\mathcal{A}$ (*g*(*x1e* := (*g x1e*)[*x2* := (*x1f*, *x2f*)])) *b*›
if ‹*x2* < *length* (*g x1e*)› **and** ‹*x1e* $\in\# \mathcal{L}_{all}\ \mathcal{A}$›
using *that*
unfolding *vdom-m-def*
by (*auto dest*!: *multi-member-split intro*!: *set-update-memI img-fst*)

lemma *in-vdom-m-fmdropD*:
‹$x \in$ *vdom-m* $\mathcal{A}$ *ga* (*fmdrop C baa*) $\Longrightarrow$ $x \in$ (*vdom-m* $\mathcal{A}$ *ga baa*)›
unfolding *vdom-m-def*
by (*auto dest*: *in-diffD*)

end
theory *IsaSAT-Arena-LLVM*
imports *IsaSAT-Arena IsaSAT-Literals-LLVM Watched-Literals.WB-More-IICF-LLVM*
begin

2.7 Code Generation

hide-const (**open**) *NEMonad.RETURN NEMonad.ASSERT*

type-synonym *arena-assn* = ‹(*32 word*, *64*) *array-list*›

lemma *protected-bind-assoc*:
‹*Refine-Basic.bind*$*m*$(λ$_2$*x*. *f x*))$(λ$_2$*y*. *g y*) =
Refine-Basic.bind$*m*$(λ$_2$*x*. *Refine-Basic.bind*$(*f x*)$(λ$_2$*y*. *g y*))› **by** *simp*

lemma *convert-swap*: ‹*WB-More-Refinement-List.swap* = *More-List.swap*›
unfolding *WB-More-Refinement-List.swap-def More-List.swap-def* **..**

Code Generation

definition ‹*arena-el-impl-rel* ≡ *unat-rel*′ *TYPE*(*32*) *O arena-el-rel*›
lemmas [*fcomp-norm-unfold*] = *arena-el-impl-rel-def*[*symmetric*]
abbreviation ‹*arena-el-impl-assn* ≡ *pure arena-el-impl-rel*›

Arena Element Operations **context**
notes [*simp*] = *arena-el-rel-def*
notes [*split*] = *arena-el.splits*

notes [*intro*!] = *frefI*
begin

Literal

lemma *xarena-lit-refine1*: ‹(λ*eli*. *eli*, *xarena-lit*) ∈ [*is-Lit*]$_f$ *arena-el-rel* → *nat-lit-rel*› **by** *auto*
sepref-def *xarena-lit-impl* [*llvm-inline*]
is [] ‹*RETURN o* (λ*eli*. *eli*)› :: ‹*uint32-nat-assn*^{k} →$_a$ *uint32-nat-assn*› **by** *sepref*
lemmas [*sepref-fr-rules*] = *xarena-lit-impl.refine*[*FCOMP xarena-lit-refine1*]

lemma *ALit-refine1*: ‹(λ*x*. *x*,*ALit*) ∈ *nat-lit-rel* → *arena-el-rel*› **by** *auto*
sepref-def *ALit-impl* [*llvm-inline*] **is** [] ‹*RETURN o* (λ*x*. *x*)›
:: ‹*uint32-nat-assn*^{k} →$_a$ *uint32-nat-assn*› **by** *sepref*
lemmas [*sepref-fr-rules*] = *ALit-impl.refine*[*FCOMP ALit-refine1*]

LBD

lemma *xarena-lbd-refine1*: ‹(λ*eli*. *eli* >> 5, *xarena-lbd*) ∈ [*is-Status*]$_f$ *arena-el-rel* → *nat-rel*›
by (*auto simp*: *is-Status-def*)

sepref-def *xarena-lbd-impl* [*llvm-inline*]
is [] ‹(*RETURN o* (λ*eli*. *eli* >> 5))› :: ‹*uint32-nat-assn*^{k} →$_a$ *uint32-nat-assn*›
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

lemmas [*sepref-fr-rules*] = *xarena-lbd-impl.refine*[*FCOMP xarena-lbd-refine1*]

Size

lemma *xarena-length-refine1*: ‹(λ*eli*. *eli*, *xarena-length*) ∈ [*is-Size*]$_f$ *arena-el-rel* → *nat-rel*› **by** *auto*
sepref-def *xarena-len-impl* [*llvm-inline*] **is** [] ‹*RETURN o* (λ*eli*. *eli*)› :: ‹*uint32-nat-assn*^{k} →$_a$ *uint32-nat-assn*›
by *sepref*
lemmas [*sepref-fr-rules*] = *xarena-len-impl.refine*[*FCOMP xarena-length-refine1*]

lemma *ASize-refine1*: ‹(λ*x*. *x*,*ASize*) ∈ *nat-rel* → *arena-el-rel*› **by** *auto*
sepref-def *ASize-impl* [*llvm-inline*] **is** [] ‹*RETURN o* (λ*x*. *x*)› :: ‹*uint32-nat-assn*^{k} →$_a$ *uint32-nat-assn*›
by *sepref*
lemmas [*sepref-fr-rules*] = *ASize-impl.refine*[*FCOMP ASize-refine1*]

Position

lemma *xarena-pos-refine1*: ‹(λ*eli*. *eli*, *xarena-pos*) ∈ [*is-Pos*]$_f$ *arena-el-rel* → *nat-rel*› **by** *auto*
sepref-def *xarena-pos-impl* [*llvm-inline*] **is** [] ‹*RETURN o* (λ*eli*. *eli*)› :: ‹*uint32-nat-assn*^{k} →$_a$ *uint32-nat-assn*›
by *sepref*
lemmas [*sepref-fr-rules*] = *xarena-pos-impl.refine*[*FCOMP xarena-pos-refine1*]

lemma *APos-refine1*: ‹(λ*x*. *x*,*APos*) ∈ *nat-rel* → *arena-el-rel*› **by** *auto*
sepref-def *APos-impl* [*llvm-inline*] **is** [] ‹*RETURN o* (λ*x*. *x*)› :: ‹*uint32-nat-assn*^{k} →$_a$ *uint32-nat-assn*›
by *sepref*
lemmas [*sepref-fr-rules*] = *APos-impl.refine*[*FCOMP APos-refine1*]

Status

definition ‹*status-impl-rel* ≡ *unat-rel*′ *TYPE*(*32*) *O status-rel*›
lemmas [*fcomp-norm-unfold*] = *status-impl-rel-def*[*symmetric*]
abbreviation ‹*status-impl-assn* ≡ *pure status-impl-rel*›

lemma *xarena-status-refine1*: ‹(λ*eli*. *eli AND 0b11*, *xarena-status*) ∈ [*is-Status*]$_f$ *arena-el-rel* → *status-rel*› **by** (*auto simp*: *is-Status-def*)

sepref-def *xarena-status-impl* [*llvm-inline*] **is** [] ‹*RETURN o* (λ*eli. eli AND 0b11*)› :: ‹*uint32-nat-assn*^{k} →$_a$ *uint32-nat-assn*›
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*
lemmas [*sepref-fr-rules*] = *xarena-status-impl.refine*[*FCOMP xarena-status-refine1*]

lemma *xarena-used-refine1*: ‹(λ*eli.* (*eli AND 0b1100*) >> *2*, *xarena-used*) ∈ [*is-Status*]$_f$ *arena-el-rel* → *nat-rel*›
by (*auto simp*: *is-Status-def status-rel-def bitfield-rel-def*)

lemma *is-down'-32-2*[*simp*]: ‹*is-down' UCAST*(*32* → *2*)›
by (*auto simp*: *is-down'*)

lemma *bitAND-mod*: ‹*L AND* (*2^n* − *1*) = *L mod* (*2^n*)› **for** *L* :: *nat*
apply *transfer*
apply (*subst int-int-eq*[*symmetric*])
apply (*subst and-nat-def*)
using *AND-mod*[*of* ‹*int* -›]
apply (*auto simp*: *zmod-int bitval-bin-last*[*symmetric*])
done

lemma *nat-ex-numeral*: ‹∃ *m*. *n*=*0* ∨ *n* = *numeral m*› **for** *n* :: *nat*
apply (*induction n*)
apply *auto*
using *llvm-num-const-simps*(*67*) **apply** *blast*
using *pred-numeral-inc* **by** *blast*

lemma *xarena-used-implI*: ‹*x AND 12* >> *2* < *max-unat 2*› **for** *x* :: *nat*
using *nat-ex-numeral*[*of x*]
by (*auto simp*: *nat-shiftr-div nat-shifl-div numeral-eq-Suc Suc-numeral max-unat-def*
less-mult-imp-div-less
simp flip: *numeral-eq-Suc*)

sepref-def *xarena-used-impl* [*llvm-inline*] **is** [] ‹*RETURN o* (λ*eli.*(*eli AND 0b1100*) >> *2*)› :: ‹*uint32-nat-assn*^{k} →$_a$ *unat-assn' TYPE*(*2*)›
supply [*simp*] = *xarena-used-implI*
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
apply (*rewrite at* ‹*RETURN o* (λ-. ⌑)› *annot-unat-unat-downcast*[**where** *'l*=*2*])
by *sepref*

sepref-register ‹(=) :: *clause-status* ⇒ *clause-status* ⇒ -›

lemmas [*sepref-fr-rules*] = *xarena-used-impl.refine*[*FCOMP xarena-used-refine1*]

lemma *status-eq-refine1*: ‹((=),(=)) ∈ *status-rel* → *status-rel* → *bool-rel*›
by (*auto simp*: *status-rel-def*)

sepref-def *status-eq-impl* [*llvm-inline*] **is** [] ‹*uncurry* (*RETURN oo* (=))›
:: ‹(*unat-assn' TYPE*(*32*))^k *$_a$ (*unat-assn' TYPE*(*32*))^k →$_a$ *bool1-assn*›
by *sepref*

lemmas [*sepref-fr-rules*] = *status-eq-impl.refine*[*FCOMP status-eq-refine1*]

sepref-register *neq* : ‹(*op-neq* :: *clause-status* ⇒ - ⇒ -)›
lemma *status-neq-refine1*: ‹((≠),*op-neq*) ∈ *status-rel* → *status-rel* → *bool-rel*›
by (*auto simp*: *status-rel-def*)

definition ‹*AStatus-impl1 cs used lbd* ≡
(*cs AND unat-const TYPE*(*32*) *0b11*) + (*used* << *2*) + (*lbd* << *unat-const TYPE*(*32*) *5*)›

lemma *bang-eq-int*:
fixes *x* :: *int*
shows (*x* = *y*) = (∀ *n*. *x* !! *n* = *y* !! *n*)
using *bin-eqI* **by** *auto*

lemma *bang-eq-nat*:
fixes *x* :: *nat*
shows (*x* = *y*) = (∀ *n*. *x* !! *n* = *y* !! *n*)
unfolding *int-int-eq*[*symmetric*] *bang-eq-int*
by (*simp add*: *bit-of-nat-iff-bit*)

lemma *sum-bitAND-shift-pow2*:
‹(*a* + (*b* << (*n* + *m*))) *AND* (*2*^*n* − *1*) = *a AND* (*2*^*n* − *1*)› **for** *a b n* :: *nat*
unfolding *bitAND-mod*
apply (*auto simp*: *nat-shiftr-div*)
by (*metis mod-mult-self2 power-add semiring-normalization-rules*(*19*))

lemma *and-bang-nat*: ‹(*x AND y*) !! *n* = (*x* !! *n* ∧ *y* !! *n*)› **for** *x y n* :: *nat*
unfolding *and-nat-def*
by (*metis and-nat-def bit-and-iff*)

lemma *AND-12-AND-15-AND-12*: ‹*a AND 12* = (*a AND 15*) *AND 12*› **for** *a* :: *nat*
proof −
have [*simp*]: ‹(*12*::*nat*) !! *n* ⟹ (*15*::*nat*) !! *n*› **for** *n* :: *nat*
by (*auto simp*: *nat-set-bit-test-bit bin-nth-numeral-unfold*
nat-bin-nth-bl′ nth-Cons split: *nat.splits*)
show *?thesis*
by (*subst bang-eq-nat*, (*subst and-bang-nat*)+)
(*auto simp*: *and-bang-nat*)
qed

lemma *AStatus-shift-safe*:
‹*c* ≥ *2* ⟹ *x42* + (*x43* << *c*) *AND 3* = *x42 AND 3*›
‹(*x53* << *2*) *AND 3* = *0*›
‹*x42* + (*x43* << *4*) *AND 12* = *x42 AND 12*›
‹*x42* + (*x43* << *5*) *AND 12* = *x42 AND 12*›
‹*Suc* (*x42* + (*x43* << *5*)) *AND 12* = (*Suc x42*) *AND 12*›
‹*Suc* ((*x42*) + (*x43* << *5*)) *AND 3* = *Suc x42 AND 3*›
‹*Suc* (*x42* << *2*) *AND 3* = *Suc 0*›
‹*x42* ≤ *3* ⟹ *Suc* ((*x42* << *2*) + (*x43* << *5*)) >> *5* = *x43*›
for *x42 x43 x53* :: *nat*
proof −
show ‹*c* ≥ *2* ⟹ *x42* + (*x43* << *c*) *AND 3* = *x42 AND 3*›
using *sum-bitAND-shift-pow2*[*of x42 x43 2* ‹*c* − *2*›]
by *auto*

show ‹(*x53* << *2*) *AND 3* = *0*›
using *bitAND-mod*[*of - 2*]
by (*auto simp*: *nat-shiftr-div*)
have *15*: ‹(*15* :: *nat*) = *2* ^*4* −*1*› **by** *auto*
show *H*: ‹*x42* + (*x43* << *4*) *AND 12* = *x42 AND 12*› **for** *x42 x43* :: *nat*

apply (*subst AND-12-AND-15-AND-12*)
apply (*subst* (*2*) *AND-12-AND-15-AND-12*)
unfolding *bitAND-mod 15*
by (*auto simp*: *nat-shiftr-div*)
from *H*[*of x42* ‹*x43* << *1*›] **show** ‹*x42* + (*x43* << *5*) *AND 12* = *x42 AND 12*›
by (*auto simp*: *nat-shiftr-div ac-simps*)
from *H*[*of* ‹*Suc x42*› ‹*x43* << *1*›] **show** ‹*Suc* (*x42* + (*x43* << *5*)) *AND 12* = (*Suc x42*) *AND 12*›
by (*auto simp*: *nat-shiftr-div ac-simps*)
have [*simp*]: ‹(*a* + *x53* ∗ *32*) *mod 4* = (*a mod 4*)› **for** *a x53* :: *nat*
by (*metis* (*no-types*, *lifting*) *add-eq-self-zero cong-exp-iff-simps*(*1*) *cong-exp-iff-simps*(*2*)
mod-add-eq mod-eq-nat1E mod-mult-right-eq mult-0-right order-refl)
note [*simp*] = *this*[*of* ‹*Suc a*› **for** *a*, *simplified*]
show ‹*Suc* ((*x42*) + (*x43* << *5*)) *AND 3* = *Suc x42 AND 3*›
using *bitAND-mod*[*of* - *2*]
by (*auto simp*: *nat-shiftr-div*)
show ‹*Suc* (*x42* << *2*) *AND 3* = *Suc 0*›
using *bitAND-mod*[*of* - *2*]
by (*auto simp*: *nat-shiftr-div mod-Suc*)
show ‹*x42* ≤ *3* ⟹ *Suc* ((*x42* << *2*) + (*x43* << *5*)) >> *5* = *x43*›
by (*auto simp*: *nat-shiftr-div nat-shifl-div*)
qed

lemma *less-unat-AND-shift*: ‹*x42* < *2*^*n* ⟹ *x42* >> *n* = *0*› **for** *x42* :: *nat*
by (*auto simp*: *nat-shifl-div*)

lemma [*simp*]: ‹(*a* + (*w* << *n*)) >> *n* = (*a* >> *n*) + *w*› ‹((*w* << *n*)) >> *n* = *w*›
‹*n* ≤ *m* ⟹ ((*w* << *n*)) >> *m* = *w* >> (*m* − *n*)›
‹*n* ≥ *m* ⟹ ((*w* << *n*)) >> *m* = *w* << (*n* − *m*)› **for** *w n* :: *nat*
apply (*auto simp*: *nat-shiftr-div nat-shifl-div*)
apply (*metis div-mult2-eq le-add-diff-inverse nonzero-mult-div-cancel-right power-add power-eq-0-iff*
zero-neq-numeral)
by (*smt Groups.mult-ac*(*2*) *le-add-diff-inverse nonzero-mult-div-cancel-right power-add power-eq-0-iff*
semiring-normalization-rules(*19*) *zero-neq-numeral*)

lemma *less-numeral-pred*:
‹*a* ≤ *numeral b* ⟷ *a* = *numeral b* ∨ *a* ≤ *pred-numeral b*› **for** *a* :: *nat*
by (*auto simp*: *numeral-eq-Suc*)

lemma *nat-shiftl-numeral* [*simp*]:
(*numeral w* :: *nat*) << *numeral w*′ = *numeral* (*num.Bit0 w*) << *pred-numeral w*′
by (*metis mult-2 nat-shiftr-div numeral-Bit0 numeral-eq-Suc power.simps*(*2*)
semiring-normalization-rules(*18*) *semiring-normalization-rules*(*7*))

lemma *nat-shiftl-numeral*′ [*simp*]:
(*numeral w* :: *nat*) << *1* = *numeral* (*num.Bit0 w*)
(*1* :: *nat*) << *n* = *2* ^ *n*
using *nat-shiftl-numeral*[*of w num.One*, *unfolded numeral.numeral-One*]
by (*auto simp*: *nat-shiftr-div*)

lemma *shiftr-nat-alt-def*: ‹(*a* :: *nat*) >> *b* = *nat* (*int a* >> *b*)›
apply (*induction b*)
apply (*auto simp*: *nat-shiftr*)
by (*smt* (*z3*) *div-div-p2-eq-div-p2s*(*2*) *int-nat-eq nat-2 nat-int.Rep-inverse*′ *nat-shifl-div numeral-2-eq-2*
semiring-1-class.of-nat-power shiftr-int-def zdiv-int)

lemma *nat-shiftr-numeral* [*simp*]:
(*1* :: *nat*) >> *numeral w'* = *0*
(*numeral num.One* :: *nat*) >> *numeral w'* = *0*
(*numeral* (*num.Bit0 w*) :: *nat*) >> *numeral w'* = *numeral w* >> *pred-numeral w'*
(*numeral* (*num.Bit1 w*) :: *nat*) >> *numeral w'* = *numeral w* >> *pred-numeral w'*
unfolding *shiftr-nat-alt-def*
by (*auto simp*: *shiftr-def*)

lemma *nat-shiftr-numeral-Suc0* [*simp*]:
(*1* :: *nat*) >> *Suc 0* = *0*
(*numeral num.One* :: *nat*) >> *Suc 0* = *0*
(*numeral* (*num.Bit0 w*) :: *nat*) >> *Suc 0* = *numeral w*
(*numeral* (*num.Bit1 w*) :: *nat*) >> *Suc 0* = *numeral w*
unfolding *shiftr-nat-alt-def*
by *auto*

lemma *nat-shiftr-numeral1* [*simp*]:
(*1* :: *nat*) >> *1* = *0*
(*numeral num.One* :: *nat*) >> *1* = *0*
(*numeral* (*num.Bit0 w*) :: *nat*) >> *1* = *numeral w*
(*numeral* (*num.Bit1 w*) :: *nat*) >> *1* = *numeral w*
unfolding *shiftr-nat-alt-def*
by *auto*

lemma *nat-numeral-and-one*: ‹(*1* :: *nat*) *AND 1* = *1*›
by *simp*

lemma *AStatus-refine1*: ‹(*AStatus-impl1*, *AStatus*) ∈ *status-rel* → *br id* (λ*n*. *n* ≤ *3*) → *nat-rel* → *arena-el-rel*›
apply (*auto simp*: *status-rel-def bitfield-rel-def AStatus-impl1-def AStatus-shift-safe br-def*
less-unat-AND-shift
split: *if-splits*)
apply (*auto simp*: *less-numeral-pred le-Suc-eq nat-and-numerals nat-numeral-and-one*;
auto simp flip: *One-nat-def*)+
done

lemma *AStatus-implI*:
assumes ‹*b* << *5* < *max-unat 32*›
shows ‹*b* << *5* < *max-unat 32* − *7*› ‹(*a AND 3*) + *4* + (*b* << *5*) < *max-unat 32*›
‹(*a AND 3*) + (*b* << *5*) < *max-unat 32*›
proof −
show ‹*b* << *5* < *max-unat 32* − *7*›
using *assms*
by (*auto simp*: *max-unat-def nat-shiftr-div*)
have ‹(*a AND 3*) + *4* + (*b* << *5*) ≤ *7* + (*b* << *5*)›
using *AND-upper-nat2*[*of 3 a*]
by *auto*
also have ‹*7* + (*b* << *5*) < *max-unat 32*›
using ‹*b* << *5* < *max-unat 32* − *7*› **by** *auto*
finally show ‹(*a AND 3*) + *4* + (*b* << *5*) < *max-unat 32*› **.**
then show ‹(*a AND 3*) + (*b* << *5*) < *max-unat 32*›
by *auto*
qed

lemma *nat-shiftr-mono*: ‹*a* < *b* ⟹ *a* << *n* < *b* << *n*› **for** *a b* :: *nat*

by (*simp add*: *nat-shiftr-div*)

lemma *AStatus-implI3*:
assumes ‹(*ac* :: *2 word*, *ba*) ∈ *unat-rel*›
shows ‹(*a AND* (*3*::*nat*)) + (*ba* << (*2*::*nat*)) < *max-unat* (*32*::*nat*)› **and**
‹*b* << *5* < *max-unat 32* ⟹ (*a AND 3*) + (*ba* << *2*) + (*b* << *5*) < *max-unat 32*›
proof −
have ‹*ba* < *4*›
using *assms unat-lt-max-unat*[*of ac*] **by** (*auto simp*: *unat-rel-def unat.rel-def br-def max-unat-def*)
from *nat-shiftr-mono*[*OF this*, *of 2*] **have** ‹*ba* << *2* < *16*› **by** *auto*
moreover have ‹(*a AND* (*3*::*nat*)) ≤ *3*›
using *AND-upper-nat2*[*of a 3*] **by** *auto*
ultimately have ‹(*a AND* (*3*::*nat*)) + (*ba* << (*2*::*nat*)) < *19*›
by *linarith*
also have ‹*19* ≤ *max-unat 32*›
by (*auto simp*: *max-unat-def*)
finally show ‹(*a AND* (*3*::*nat*)) + (*ba* << (*2*::*nat*)) < *max-unat* (*32*::*nat*)› **.**

show ‹(*a AND 3*) + (*ba* << *2*) + (*b* << *5*) < *max-unat 32*› **if** ‹*b* << *5* < *max-unat 32*›
proof −
have ‹*b* << *5* < *max-unat 32* − *19*›
using *that*
by (*auto simp*: *max-unat-def nat-shiftr-div*)
then show *?thesis*
using ‹(*a AND* (*3*::*nat*)) + (*ba* << (*2*::*nat*)) < *19*› **by** *linarith*
qed
qed

lemma *AStatus-implI2*: ‹(*ac* :: *2 word*, *ba*) ∈ *unat-rel* ⟹ *ba* << (*2*::*nat*) < *max-unat* (*32*::*nat*)›
using *order.strict-trans2*[*OF unat-lt-max-unat*[*of ac*], *of* ‹*max-unat 28*›]
by (*auto simp*: *unat-rel-def unat.rel-def br-def max-unat-def nat-shiftr-div intro*!:)

lemma *is-up-2-32*[*simp*]: ‹*is-up′ UCAST*(*2* → *32*)›
by (*simp add*: *is-up′*)

sepref-def *AStatus-impl* [*llvm-inline*]
is [] ‹*uncurry2* (*RETURN ooo AStatus-impl1*)›
:: ‹[λ((*a*,*b*), *c*). *c* << *5* < *max-unat 32*]$_a$
uint32-nat-assn^k *$_a$* (*unat-assn′ TYPE*(*2*))^k *$_a$* *uint32-nat-assn*^{k} → *uint32-nat-assn*›
unfolding *AStatus-impl1-def*
supply [*split*] = *if-splits* **and** [*intro*] = *AStatus-implI AStatus-implI2 AStatus-implI3*
apply (*rewrite* **in** ‹¤ << *2*› *annot-unat-unat-upcast*[**where** *′l*=‹*32*›])
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

lemma *Collect-eq-simps3*: ‹*P O* {(*c*, *a*). *a* = *c* ∧ *Q c*} = {(*a*, *b*). (*a*, *b*) ∈ *P* ∧ *Q b*}›
‹*P O* {(*c*, *a*). *c* = *a* ∧ *Q c*} = {(*a*, *b*). (*a*, *b*) ∈ *P* ∧ *Q b*}›
by *auto*

lemma *unat-rel-2-br*: ‹(((*unat-rel* :: (*2 word* × -) *set*) *O br id* (λ*n*. *n* ≤ *3*))) = ((*unat-rel*))›
apply (*auto simp add*: *unat-rel-def unat.rel-def br-def Collect-eq-simps3 max-unat-def*)
subgoal for *a*
using *unat-lt-max-unat*[*of* ‹*a* :: *2 word*›] **by** (*auto simp*: *max-unat-def*)
done

lemmas [*sepref-fr-rules*] = *AStatus-impl.refine*[*FCOMP AStatus-refine1*, *unfolded unat-rel-2-br*]

Arena Operations

Length **abbreviation** ‹*arena-fast-assn* ≡ *al-assn′ TYPE*(*64*) *arena-el-impl-assn*›

lemma *arena-lengthI*:
assumes ‹*arena-is-valid-clause-idx a b*›
shows ‹*Suc 0* ≤ *b*›
and ‹*b* < *length a*›
and ‹*is-Size* (*a* ! (*b* − *Suc 0*))›
using *SIZE-SHIFT-def assms*
by (*auto simp*: *arena-is-valid-clause-idx-def arena-lifting*)

lemma *arena-length-alt*:
‹*arena-length arena i* = (
let l = *xarena-length* (*arena*!(*i* − *snat-const TYPE*(*64*) *1*))
in snat-const TYPE(*64*) *2* + *op-unat-snat-upcast TYPE*(*64*) *l*)›
by (*simp add*: *arena-length-def SIZE-SHIFT-def*)

sepref-register *arena-length*
sepref-def *arena-length-impl*
is ‹*uncurry* (*RETURN oo arena-length*)›
:: ‹[*uncurry arena-is-valid-clause-idx*]$_a$ *arena-fast-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} → *snat-assn′ TYPE*(*64*)›
unfolding *arena-length-alt*
supply [*dest*] = *arena-lengthI*
by *sepref*

Literal at given position **lemma** *arena-lit-implI*:
assumes ‹*arena-lit-pre a b*›
shows ‹*b* < *length a*› ‹*is-Lit* (*a* ! *b*)›
using *assms* **unfolding** *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
by (*fastforce dest*: *arena-lifting*)+

sepref-register *arena-lit xarena-lit*
sepref-def *arena-lit-impl*
is ‹*uncurry* (*RETURN oo arena-lit*)›
:: ‹[*uncurry arena-lit-pre*]$_a$ *arena-fast-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} → *unat-lit-assn*›
supply [*intro*] = *arena-lit-implI*
unfolding *arena-lit-def*
by *sepref*

sepref-register *mop-arena-lit mop-arena-lit2*
sepref-def *mop-arena-lit-impl*
is ‹*uncurry* (*mop-arena-lit*)›
:: ‹*arena-fast-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} →$_a$ *unat-lit-assn*›
supply [*intro*] = *arena-lit-implI*
unfolding *mop-arena-lit-def*
by *sepref*

sepref-def *mop-arena-lit2-impl*
is ‹*uncurry2* (*mop-arena-lit2*)›
:: ‹[λ((*N*, -), -). *length N* ≤ *snat64-max*]$_a$

arena-fast-assn^k $*_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\to$ *unat-lit-assn*›
supply [*intro*] = *arena-lit-implI*
supply [*dest*] = *arena-lit-pre-le-lengthD*
unfolding *mop-arena-lit2-def*
by *sepref*

Status of the clause **lemma** *arena-status-implI*:
assumes ‹*arena-is-valid-clause-vdom a b*›
shows ‹2 ≤ *b*› ‹*b* − 2 < *length a*› ‹*is-Status* (*a* ! (*b*−2))›
using *assms STATUS-SHIFT-def arena-dom-status-iff*
unfolding *arena-is-valid-clause-vdom-def*
by (*auto dest*: *valid-arena-in-vdom-le-arena arena-lifting*)

sepref-register *arena-status xarena-status*
sepref-def *arena-status-impl*
is ‹*uncurry* (*RETURN oo arena-status*)›
:: ‹[*uncurry arena-is-valid-clause-vdom*]$_a$ *arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\to$ *status-impl-assn*›
supply [*intro*] = *arena-status-implI*
unfolding *arena-status-def STATUS-SHIFT-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

Swap literals **sepref-register** *swap-lits*
sepref-def *swap-lits-impl* **is** ‹*uncurry3* (*RETURN oooo swap-lits*)›
:: ‹[λ(((*C*,*i*),*j*),*arena*). *C* + *i* < *length arena* ∧ *C* + *j* < *length arena*]$_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *arena-fast-assn*^{d} $\to$ *arena-fast-assn*›
unfolding *swap-lits-def convert-swap*
unfolding *gen-swap*
by *sepref*

Get LBD **lemma** *get-clause-LBD-pre-implI*:
assumes ‹*get-clause-LBD-pre a b*›
shows ‹2 ≤ *b*› ‹*b* − 2 < *length a*› ‹*is-Status* (*a* ! (*b*−2))›
using *assms arena-dom-status-iff*
unfolding *arena-is-valid-clause-vdom-def get-clause-LBD-pre-def*
apply (*auto dest*: *valid-arena-in-vdom-le-arena simp*: *arena-lifting arena-is-valid-clause-idx-def*)
using *STATUS-SHIFT-def arena-lifting* **apply** *auto*
by (*meson less-imp-diff-less*)

sepref-register *arena-lbd mop-arena-lbd*
sepref-def *arena-lbd-impl*
is ‹*uncurry* (*RETURN oo arena-lbd*)›
:: ‹[*uncurry get-clause-LBD-pre*]$_a$ *arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\to$*uint32-nat-assn*›
unfolding *arena-lbd-def LBD-SHIFT-def*
supply [*dest*] = *get-clause-LBD-pre-implI*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *mop-arena-lbd-impl*
is ‹*uncurry mop-arena-lbd*›
:: ‹*arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\to_a$ *uint32-nat-assn*›
unfolding *mop-arena-lbd-def*
by *sepref*

used flag **sepref-register** *arena-used*

sepref-def *arena-used-impl*
is ‹*uncurry* (*RETURN oo arena-used*)›
:: ‹[*uncurry get-clause-LBD-pre*]$_a$ *arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\to$ *unat-assn*$'$ *TYPE*(*2*)›
unfolding *arena-used-def LBD-SHIFT-def*
supply [*dest*] = *get-clause-LBD-pre-implI*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

Get Saved Position **lemma** *arena-posI*:
assumes ‹*get-saved-pos-pre a b*›
shows ‹*3* $\le$ *b*›
and ‹*b* < *length a*›
and ‹*is-Pos* (*a* ! (*b* − *3*))›
using *POS-SHIFT-def assms is-short-clause-def*[*of* ‹- $\propto$ *b*›]
apply (*auto simp*: *get-saved-pos-pre-def arena-is-valid-clause-idx-def arena-lifting*
MAX-LENGTH-SHORT-CLAUSE-def[*symmetric*] *arena-lifting*(*11*) *arena-lifting*(*4*)
simp del: *MAX-LENGTH-SHORT-CLAUSE-def*)
using *arena-lifting*(*1*) *arena-lifting*(*4*) *header-size-def* **by** *fastforce*

lemma *arena-pos-alt*:
‹*arena-pos arena i* = (
let l = *xarena-pos* (*arena*!(*i* − *snat-const TYPE*(*64*) *3*))
in snat-const TYPE(*64*) *2* + *op-unat-snat-upcast TYPE*(*64*) *l*)›
by (*simp add*: *arena-pos-def POS-SHIFT-def*)

sepref-register *arena-pos*
sepref-def *arena-pos-impl*
is ‹*uncurry* (*RETURN oo arena-pos*)›
:: ‹[*uncurry get-saved-pos-pre*]$_a$ *arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\to$ *snat-assn*$'$ *TYPE*(*64*)›
unfolding *arena-pos-alt*
supply [*dest*] = *arena-posI*
by *sepref*

Update LBD **lemma** *update-lbdI*:
assumes ‹*update-lbd-pre* ((*b*, *lbd*), *a*)›
shows ‹*2* $\le$ *b*›
and ‹*b* −*2* < *length a*›
and ‹*arena-is-valid-clause-vdom a b*›
and ‹*get-clause-LBD-pre a b*›
using *LBD-SHIFT-def assms*
apply (*auto simp*: *arena-is-valid-clause-idx-def arena-lifting update-lbd-pre-def*
arena-is-valid-clause-vdom-def get-clause-LBD-pre-def
dest: *arena-lifting*(*10*))
by (*simp add*: *less-imp-diff-less valid-arena-def*)

lemma *shorten-lbd-le*: ‹*shorten-lbd baa* << *5* < *max-unat 32*›
proof −
have ‹*shorten-lbd baa* << *5* $\le$ *67108863* << *5*›
using *AND-upper-nat2*[*of baa 67108863*]
by (*auto simp*: *nat-shiftr-div shorten-lbd-def*)
also have ‹*67108863* << *5* < *max-unat 32*›
by (*auto simp*: *max-unat-def nat-shiftr-div*)
finally show *?thesis* **.**
qed

sepref-register *update-lbd AStatus shorten-lbd*
sepref-def *shorten-lbd-impl*
is ‹*RETURN o shorten-lbd*›
:: ‹*uint32-nat-assn*^{k} →$_a$ *uint32-nat-assn*›
unfolding *shorten-lbd-def*
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

sepref-def *update-lbd-impl*
is ‹*uncurry2* (*RETURN ooo update-lbd*)›
:: ‹[*update-lbd-pre*]$_a$ *sint64-nat-assn*^{k} *$_a$ *uint32-nat-assn*^{k} *$_a$ *arena-fast-assn*^{d} → *arena-fast-assn*›
unfolding *update-lbd-def LBD-SHIFT-def*
supply [*simp*] = *update-lbdI shorten-lbd-le*
and [*dest*] = *arena-posI*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *mop-arena-update-lbd-impl*
is ‹*uncurry2 mop-arena-update-lbd*›
:: ‹*sint64-nat-assn*^{k} *$_a$ *uint32-nat-assn*^{k} *$_a$ *arena-fast-assn*^{d} →$_a$ *arena-fast-assn*›
unfolding *mop-arena-update-lbd-def*
by *sepref*

Update Saved Position **lemma** *update-posI*:
assumes ‹*isa-update-pos-pre* ((*b*, *pos*), *a*)›
shows ‹*3* ≤ *b*› ‹*2* ≤ *pos*› ‹*b*−*3* < *length a*›
using *assms POS-SHIFT-def*
unfolding *isa-update-pos-pre-def*
apply (*auto simp*: *arena-is-valid-clause-idx-def arena-lifting*)

apply (*metis* (*full-types*) *MAX-LENGTH-SHORT-CLAUSE-def arena-is-valid-clause-idx-def arena-posI*(*1*) *get-saved-pos-pre-def*)
by (*simp add*: *less-imp-diff-less valid-arena-def*)

lemma *update-posI2*:
assumes ‹*isa-update-pos-pre* ((*b*, *pos*), *a*)›
assumes ‹*rdomp* (*al-assn arena-el-impl-assn* :: - ⇒ *arena-assn* ⇒ *assn*) *a*›
shows ‹*pos* − *2* < *max-unat 32*›
proof −
obtain *N vdom* **where**
‹*valid-arena a N vdom*› **and**
‹*b* ∈# *dom-m N*›
using *assms*(*1*) **unfolding** *isa-update-pos-pre-def arena-is-valid-clause-idx-def*
by *auto*
then have *eq*: ‹*length* (*N* ∝ *b*) = *arena-length a b*› **and**
le: ‹*b* < *length a*› **and**
size: ‹*is-Size* (*a* ! (*b* − *SIZE-SHIFT*))›
by (*auto simp*: *arena-lifting*)

have ‹*i*<*length a* ⟹ *rdomp arena-el-impl-assn* (*a* ! *i*)› **for** *i*
using *rdomp-al-dest'*[*OF assms*(*2*)]
by *auto*
from *this*[*of* ‹*b* − *SIZE-SHIFT*›] **have** ‹*rdomp arena-el-impl-assn* (*a* ! (*b* − *SIZE-SHIFT*))›
using *le* **by** *auto*
then have ‹*length* (*N* ∝ *b*) ≤ *unat32-max* + *2*›

```
    using size_eq unfolding rdomp-pure
    apply (auto simp: rdomp-def arena-el-impl-rel-def is-Size-def
      comp-def pure-def unat-rel-def unat.rel-def br-def
      arena-length-def unat32-max-def)
     subgoal for x
      using unat-lt-max-unat[of x]
      apply (auto simp: max-unat-def)
      done
    done
  then show ?thesis
    using assms POS-SHIFT-def
    unfolding isa-update-pos-pre-def
    by (auto simp: arena-is-valid-clause-idx-def arena-lifting eq
      unat32-max-def max-unat-def)
qed

sepref-register arena-update-pos
sepref-def update-pos-impl
  is ‹uncurry2 (RETURN ooo arena-update-pos)›
   :: ‹[isa-update-pos-pre]_a sint64-nat-assn^k *_a sint64-nat-assn^k *_a arena-fast-assn^d → arena-fast-assn›
  unfolding arena-update-pos-def POS-SHIFT-def
  apply (annot-snat-const ‹TYPE(64)›)
  apply (rewrite at ‹APos ⌑› annot-snat-unat-downcast[where 'l=32])
  supply [simp] = update-posI and [dest] = update-posI2
  by sepref


sepref-register IRRED LEARNED DELETED
lemma IRRED-impl[sepref-import-param]: ‹(0,IRRED) ∈ status-impl-rel›
  unfolding status-impl-rel-def status-rel-def unat-rel-def unat.rel-def
  by (auto simp: in-br-conv)

lemma LEARNED-impl[sepref-import-param]: ‹(1,LEARNED) ∈ status-impl-rel›
  unfolding status-impl-rel-def status-rel-def unat-rel-def unat.rel-def
  by (auto simp: in-br-conv)

lemma DELETED-impl[sepref-import-param]: ‹(3,DELETED) ∈ status-impl-rel›
  unfolding status-impl-rel-def status-rel-def unat-rel-def unat.rel-def
  by (auto simp: in-br-conv)


lemma mark-garbageI:
  assumes ‹mark-garbage-pre (a, b)›
  shows ‹2 ≤ b› ‹b-2 < length a›
  using assms STATUS-SHIFT-def
  unfolding mark-garbage-pre-def
  apply (auto simp: arena-is-valid-clause-idx-def arena-lifting)
  by (simp add: less-imp-diff-less valid-arena-def)

sepref-register extra-information-mark-to-delete
sepref-def mark-garbage-impl is ‹uncurry (RETURN oo extra-information-mark-to-delete)›
  :: ‹[mark-garbage-pre]_a arena-fast-assn^d *_a sint64-nat-assn^k → arena-fast-assn›
  unfolding extra-information-mark-to-delete-def STATUS-SHIFT-def
  apply (rewrite at ‹AStatus - - ⌑› annot-snat-unat-downcast[where 'l=32])
  apply (rewrite at ‹AStatus - ⌑› unat-const-fold[where 'a=2])
  apply (annot-snat-const ‹TYPE(64)›)
```

supply [*simp*] = *mark-garbageI*
by *sepref*

lemma *bit-shiftr-shiftl-same-le*:
‹*a* << *b* >> *b* ≤ *a*› **for** *a b c* :: *nat*
unfolding *nat-int-comparison*
by (*auto simp*: *nat-shiftr-div nat-shifl-div*)

lemma *bit-shiftl-shiftr-same-le*:
‹*a* >> *b* << *b* ≤ *a*› **for** *a b c* :: *nat*
by (*auto simp*: *nat-shiftr-div nat-shifl-div*)

lemma *valid-arena-arena-lbd-shift-le*:
assumes
‹*rdomp* (*al-assn arena-el-impl-assn*) *a*› **and**
‹*b* ∈# *dom-m N*› **and**
‹*valid-arena a N vdom*›
shows ‹*arena-lbd a b* << *5* < *max-unat 32*›
proof −
have ‹*2* ≤ *b*› ‹*b* − *2* < *length a*› **and** *st*: ‹*is-Status* (*a* ! (*b*−*2*))›
using *assms LBD-SHIFT-def* **by** (*auto simp*: *arena-is-valid-clause-idx-def*
less-imp-diff-less arena-lifting)
then have *H*: ‹*rdomp arena-el-impl-assn* (*a* ! (*b* − *2*))›
using *rdomp-al-dest'*[*of arena-el-impl-assn a*] *assms*
by *auto*
then obtain *x* :: ‹*32 word*› **and** *x51* :: ‹*clause-status*› **and** *x52* **where**
H: ‹*a* ! (*b* − *2*) = *AStatus x51 x52* (*unat x* >> *5*)›
‹(*unat x AND 3*, *x51*) ∈ *status-rel*›
using *st bit-shiftr-shiftl-same-le*[*of* ‹*arena-lbd a b*› *4*]
by (*auto simp*: *arena-el-impl-rel-def unat-rel-def unat.rel-def*
br-def arena-lbd-def LBD-SHIFT-def)

show *?thesis*
apply (*rule order.strict-trans1*[*of* - ‹*unat x*›])
using *bit-shiftl-shiftr-same-le*[*of* ‹*unat x*› *5*] *unat-lt-max-unat*[*of* ‹*x*›] *H*
by (*auto simp*: *arena-el-impl-rel-def unat-rel-def unat.rel-def*
br-def arena-lbd-def LBD-SHIFT-def)
qed

lemma *arena-mark-used-implI*:
assumes ‹*arena-act-pre a b*›
shows ‹*2* ≤ *b*› ‹*b* − *2* < *length a*› ‹*is-Status* (*a* ! (*b*−*2*))›
‹*arena-is-valid-clause-vdom a b*›
‹*get-clause-LBD-pre a b*›
‹*rdomp* (*al-assn arena-el-impl-assn*) *a* ⟹ *arena-lbd a b* << *5* < *max-unat 32*›
using *assms STATUS-SHIFT-def valid-arena-arena-lbd-shift-le*[*of a b*]
apply (*auto simp*: *arena-act-pre-def arena-is-valid-clause-idx-def arena-lifting*)
subgoal by (*simp add*: *less-imp-diff-less valid-arena-def*)
subgoal for *N vdom* **by** (*auto simp*: *arena-is-valid-clause-vdom-def arena-lifting*)
subgoal for *N vdom* **by** (*auto simp*: *arena-is-valid-clause-vdom-def arena-lifting*
get-clause-LBD-pre-def arena-is-valid-clause-idx-def)
done

lemma *mark-used-alt-def*:
‹*RETURN oo mark-used* =
(λ*arena i*. *do* {
lbd ← *RETURN* (*arena-lbd arena i*); *let status* = *arena-status arena i*;
RETURN (*arena*[*i* − *STATUS-SHIFT* := *AStatus status* (*arena-used arena i OR 1*) *lbd*])})›
by (*auto simp*: *mark-used-def Let-def intro*!: *ext*)

sepref-register *mark-used mark-used2*
sepref-def *mark-used-impl* **is** ‹*uncurry* (*RETURN oo mark-used*)›
:: ‹[*uncurry arena-act-pre*]$_a$ *arena-fast-assn*^{d} *$*_a$ *sint64-nat-assn*^{k} → *arena-fast-assn*›
unfolding *mark-used-def STATUS-SHIFT-def mark-used-alt-def*
supply [*intro*] = *arena-mark-used-implI*
apply (*rewrite at* ‹- *OR* ⌑› *unat-const-fold*[**where** *'a=2*])
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *mark-used2-impl* **is** ‹*uncurry* (*RETURN oo mark-used2*)›
:: ‹[*uncurry arena-act-pre*]$_a$ *arena-fast-assn*^{d} *$*_a$ *sint64-nat-assn*^{k} → *arena-fast-assn*›
unfolding *mark-used2-def STATUS-SHIFT-def mark-used-alt-def*
supply [*intro*] = *arena-mark-used-implI*
apply (*rewrite at* ‹- *OR* ⌑› *unat-const-fold*[**where** *'a=2*])
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-register *mark-unused*
sepref-def *mark-unused-impl* **is** ‹*uncurry* (*RETURN oo mark-unused*)›
:: ‹[*uncurry arena-act-pre*]$_a$ *arena-fast-assn*^{d} *$*_a$ *sint64-nat-assn*^{k} → *arena-fast-assn*›
unfolding *mark-unused-def STATUS-SHIFT-def*
supply [*intro*] = *arena-mark-used-implI*
apply (*rewrite at* ‹- − ⌑› *snat-const-fold*[**where** *'a=64*])
apply (*rewrite at* ‹- − ⌑› *snat-const-fold*[**where** *'a=64*])
apply (*annot-unat-const* ‹*TYPE*(*2*)›)
by *sepref*

sepref-def *mop-arena-mark-used-impl*
is ‹*uncurry mop-arena-mark-used*›
:: ‹*arena-fast-assn*^{d} *$*_a$ *sint64-nat-assn*^{k} →$_a$ *arena-fast-assn*›
unfolding *mop-arena-mark-used-def*
by *sepref*

sepref-def *mop-arena-mark-used2-impl*
is ‹*uncurry mop-arena-mark-used2*›
:: ‹*arena-fast-assn*^{d} *$*_a$ *sint64-nat-assn*^{k} →$_a$ *arena-fast-assn*›
unfolding *mop-arena-mark-used2-def*
by *sepref*

Marked as used? **lemma** *arena-marked-as-used-implI*:
assumes ‹*marked-as-used-pre a b*›
shows ‹*2* ≤ *b*› ‹*b* − *2* < *length a*› ‹*is-Status* (*a* ! (*b*−*2*))›
using *assms STATUS-SHIFT-def*
apply (*auto simp*: *marked-as-used-pre-def arena-is-valid-clause-idx-def arena-lifting*)
subgoal using *arena-lifting*(*2*) *less-imp-diff-less* **by** *blast*
done

sepref-register *marked-as-used*

sepref-def *marked-as-used-impl*
  is ‹*uncurry* (*RETURN oo marked-as-used*)›
    :: ‹[*uncurry marked-as-used-pre*]$_a$ *arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\to$ *unat-assn'* *TYPE*(*2*)›
  supply [*intro*] = *arena-marked-as-used-implI*
  unfolding *marked-as-used-def STATUS-SHIFT-def*
  apply (*annot-snat-const* ‹*TYPE*(*64*)›)
  by *sepref*

sepref-register *MAX-LENGTH-SHORT-CLAUSE mop-arena-status*
sepref-def *MAX-LENGTH-SHORT-CLAUSE-impl* **is** ‹*uncurry0* (*RETURN MAX-LENGTH-SHORT-CLAUSE*)›
:: ‹*unit-assn*^{k} $\to_a$ *sint64-nat-assn*›
  unfolding *MAX-LENGTH-SHORT-CLAUSE-def*
  apply (*annot-snat-const* ‹*TYPE*(*64*)›)
  by *sepref*

definition *arena-other-watched-as-swap* :: ‹*nat list* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *nat nres*› **where**
‹*arena-other-watched-as-swap S L C i* = *do* {
    ASSERT(*i* < *2* $\wedge$
      C + *i* < *length S* $\wedge$
      C < *length S* $\wedge$
      (*C* + *1*) < *length S*);
    K $\leftarrow$ *RETURN* (*S* ! *C*);
    K' $\leftarrow$ *RETURN* (*S* ! (*1* + *C*));
    RETURN (*L XOR K XOR K'*)
  }›

lemma *arena-other-watched-as-swap-arena-other-watched*:
  assumes
    N: ‹(*N*, *N'*) $\in$ ⟨*arena-el-rel*⟩*list-rel*› **and**
    L: ‹(*L*, *L'*) $\in$ *nat-lit-rel*› **and**
    C: ‹(*C*, *C'*) $\in$ *nat-rel*› **and**
    i: ‹(*i*, *i'*) $\in$ *nat-rel*›
  shows
    ‹*arena-other-watched-as-swap N L C i* $\le$ $\Downarrow$*nat-lit-rel*
       (*arena-other-watched N' L' C' i'*)›
proof −
   have *eq*: ‹*i* =*i'*› ‹*C*=*C'*›
     using *assms* **by** *auto*
   have *A*: ‹*Pos* (*L div 2*) = *A* $\Longrightarrow$ *even L* $\Longrightarrow$ *L* = *2* $*$ *atm-of A*› **for** *A* :: ‹*nat literal*›
     by (*cases A*)
      auto
   have *Ci*: ‹(*C'* + *i'*, *C'* + *i'*) $\in$ *nat-rel*›
     unfolding *eq* **by** *auto*
   have [*simp*]: ‹*L* = *N* ! (*C*+*i*)› **if** ‹*L'* = *arena-lit N'* (*C'* + *i'*)› ‹*C'* + *i'* < *length N'*›
     ‹*arena-lit-pre2 N' C i*›
     using *that param-nth*[*OF that*(*2*) *Ci N*] *C i L*
     unfolding *arena-lit-pre2-def*
     apply − **apply** *normalize-goal+*
     subgoal for *N''* *vdom*
       using *arena-lifting*(*6*)[*of N' N'' vdom C i*] *A*[*of* ‹*arena-lit N'* (*C'* + *i'*)›]
       apply (*simp only*: *list-rel-imp-same-length*[*of N*] *eq*)
     apply (*cases* ‹*N'* ! (*C'* + *i'*)›; *cases* ‹*arena-lit N'* (*C'* + *i'*)›)
     apply (*simp-all add*: *eq nat-lit-rel-def br-def*)
     apply (*auto split*: *if-splits simp*: *eq-commute*[*of* - ‹*Pos* (*L div 2*)›]
       eq-commute[*of* - ‹*ALit* (*Pos* (- *div 2*))›] *arena-lit-def*)

```
    using div2-even-ext-nat by blast
   done
  have [simp]: ‹N ! (C' + i') XOR N ! C' XOR N ! Suc C' = N ! (C' + (Suc 0 − i))› if ‹i < 2›
   using that i
   by (cases i; cases ‹i−1›)
    (auto simp: bin-pos-same-XOR3-nat)
 have Ci': ‹(C' + (1 − i'), C' + (1 − i')) ∈ nat-rel›
  unfolding eq by auto
 have [intro!]: ‹(N ! (Suc C' − i'), arena-lit N' (Suc C' − i')) ∈ nat-lit-rel›
   if ‹arena-lit-pre2 N' C i› ‹i < 2›
   using that param-nth[OF - Ci' N]
   unfolding arena-lit-pre2-def
   apply − apply normalize-goal+
   apply (subgoal-tac ‹C' + (Suc 0 − i') < length N'›)
   defer
    subgoal for N'' vdom
    using
      arena-lifting(7)[of N' N'' vdom C i]
      arena-lifting(7)[of N' N'' vdom C ‹Suc 0 − i›]
      arena-lifting(21,4)[of N' N'' vdom C]
     by (cases i')
      (auto simp: arena-lit-pre2-def list-rel-imp-same-length[of N] eq
       simp del: arena-el-rel-def)
   apply (subgoal-tac ‹(Suc 0 − i') < length (x ∝ C)›)
   defer
   subgoal for N'' vdom
    using
      arena-lifting(7)[of N' N'' vdom C i]
      arena-lifting(7)[of N' N'' vdom C ‹Suc 0 − i›]
      arena-lifting(21,4)[of N' N'' vdom C]
     by (cases i')
      (auto simp: arena-lit-pre2-def list-rel-imp-same-length[of N] eq
       simp del: arena-el-rel-def)
   subgoal for N'' vdom
    using
      arena-lifting(6)[of N' N'' vdom C ‹Suc 0 − i›]
    by (cases ‹N' ! (C' + (Suc 0 − i'))›)
     (auto simp: arena-lit-pre2-def list-rel-imp-same-length[of N] eq
       arena-lit-def arena-lifting)
   done
  show ?thesis
   using assms
   unfolding arena-other-watched-as-swap-def arena-other-watched-def
    le-ASSERT-iff mop-arena-lit2-def
   apply (refine-vcg)
   apply (auto simp: le-ASSERT-iff list-rel-imp-same-length arena-lit-pre2-def
    arena-lifting
    bin-pos-same-XOR3-nat)
    apply (metis (no-types, lifting) add.comm-neutral add-Suc-right arena-lifting(21,4,7))
    using arena-lifting(4) by auto
qed


sepref-def arena-other-watched-as-swap-impl
 is ‹uncurry3 arena-other-watched-as-swap›
 :: ‹(al-assn' (TYPE(64)) uint32-nat-assn)^k *_a uint32-nat-assn^k *_a sint64-nat-assn^k *_a
```

sint64-nat-assn^k $\rightarrow_a$ *uint32-nat-assn*›
supply[[*goals-limit=1*]]
unfolding *arena-other-watched-as-swap-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *arena-other-watched-as-swap-arena-other-watched'*:
‹(*arena-other-watched-as-swap*, *arena-other-watched*) ∈
⟨*arena-el-rel*⟩*list-rel* → *nat-lit-rel* → *nat-rel* → *nat-rel* →
⟨*nat-lit-rel*⟩*nres-rel*›
apply (*intro fun-relI nres-relI*)
using *arena-other-watched-as-swap-arena-other-watched*
by *blast*

lemma *arena-fast-al-unat-assn*:
‹*hr-comp* (*al-assn unat-assn*) (⟨*arena-el-rel*⟩*list-rel*) = *arena-fast-assn*›
unfolding *al-assn-def hr-comp-assoc*
by (*auto simp*: *arena-el-impl-rel-def list-rel-compp*)

lemmas [*sepref-fr-rules*] =
arena-other-watched-as-swap-impl.refine[*FCOMP arena-other-watched-as-swap-arena-other-watched'*,
unfolded arena-fast-al-unat-assn]

lemma *arena-lit-pre2-le-lengthD*: ‹*arena-lit-pre2 arena i j* ⟹ *i* + *j* < *length arena*›
apply (*auto simp*: *arena-lit-pre2-def*)
using *arena-lifting*(*7*) *nat-le-iff-add* **by** *auto*

sepref-def *mop-arena-update-lit-code*
is ‹*uncurry3 mop-arena-update-lit*›
:: ‹[λ(((-, -), -), *N*). *length N* ≤ *snat64-max*]$_a$
sint64-nat-assn^k $*_a$ *sint64-nat-assn*^{k} $*_a$ *unat-lit-assn*^{k} $*_a$ *arena-fast-assn*^{d} → *arena-fast-assn*›
supply [*intro*] = *arena-lit-implI*
supply [*dest*] = *arena-lit-pre2-le-lengthD*
unfolding *mop-arena-update-lit-def arenap-update-lit-def*
by *sepref*

lemma *arena-shorten-preI*:
assumes ‹*arena-shorten-pre C j arena*›
shows
‹*j* ≥ *2*› **and**
‹*C* − *Suc 0* < *length arena*› **and**
‹*C* ≥ *Suc 0*› **and**
‹*j* > *MAX-LENGTH-SHORT-CLAUSE* ⟹ *C* ≥ *3*›
using *assms arena-lifting*[*of arena - - C*]
unfolding *arena-shorten-pre-def* **by** (*auto simp*: *arena-is-valid-clause-idx-def SHIFTS-def*
header-size-def
intro: *less-imp-diff-less*)

sepref-def *mop-arena-shorten-code*
is ‹*uncurry2 mop-arena-shorten*›
:: ‹*sint64-nat-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $*_a$ *arena-fast-assn*^{d} $\rightarrow_a$ *arena-fast-assn*›
supply [*dest*] = *arena-shorten-preI*
supply [*dest*] = *arena-lit-pre2-le-lengthD*
unfolding *mop-arena-shorten-def arena-shorten-def SIZE-SHIFT-def POS-SHIFT-def*
apply (*rewrite at* ‹-[- := *ASize* (- − ¤), - := -]› *unat-const-fold*[**where** *'a=32*])

apply (*rewrite at* ‹-[- := *ASize* (- − ¤)]› *unat-const-fold*[**where** *'a=32*])
apply (*rewrite at* ‹-[- := -, - := *APos* ¤]› *unat-const-fold*[**where** *'a=32*])
apply (*annot-snat-const* ‹*TYPE* (*64*)›)
apply (*rewrite at* ‹(- < ¤)› *annot-unat-snat-upcast*[**where** *'l=64*])
by *sepref*

end

sepref-def *mop-arena-length-impl*
is ‹*uncurry mop-arena-length*›
:: ‹*arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*›
unfolding *mop-arena-length-def*
by *sepref*

sepref-def *mop-arena-status-impl*
is ‹*uncurry mop-arena-status*›
:: ‹*arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *status-impl-assn*›
supply [[*goals-limit=1*]]
unfolding *mop-arena-status-def*
by *sepref*

sepref-def *status-neq-impl* **is** [] ‹*uncurry* (*RETURN oo* (≠))›
:: ‹(*unat-assn' TYPE*(*32*))^k $*_a$ (*unat-assn' TYPE*(*32*))^k $\rightarrow_a$ *bool1-assn*›
by *sepref*

lemmas [*sepref-fr-rules*] = *status-neq-impl.refine*[*FCOMP status-neq-refine1*]

sepref-register *mop-arena-set-status*

lemma *arena-is-valid-clause-idxI*:
‹*arena-is-valid-clause-idx arena C* ⟹ *get-clause-LBD-pre arena C*›
‹*arena-is-valid-clause-idx arena C* ⟹ *C* ≥ *2*›
‹*arena-is-valid-clause-idx arena C* ⟹ *rdomp* (*al-assn arena-el-impl-assn*) *arena* ⟹ *arena-lbd arena C* << *5* < *max-unat 32*›
‹*arena-is-valid-clause-idx arena C* ⟹ *C* − *2* < *length arena*›
using *valid-arena-arena-lbd-shift-le*[*of arena C*]
unfolding *arena-is-valid-clause-vdom-def get-clause-LBD-pre-def arena-is-valid-clause-idx-def*
apply (*auto simp*: *arena-lifting*)
using *STATUS-SHIFT-def arena-lifting*(*16*) **apply** *auto*[*1*]
defer
using *arena-lifting*(*2*) *less-imp-diff-less* **apply** *blast*
done

sepref-def *mop-arena-set-status-impl*
is ‹*uncurry2 mop-arena-set-status*›
:: ‹*arena-fast-assn*^{d} $*_a$ *sint64-nat-assn*^{k} $*_a$ *status-impl-assn*^{k} $\rightarrow_a$ *arena-fast-assn*›
supply [*intro*] = *arena-is-valid-clause-idxI*
unfolding *mop-arena-set-status-def arena-set-status-def LBD-SHIFT-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

experiment begin

export-llvm
MAX-LENGTH-SHORT-CLAUSE-impl
mop-arena-status-impl
arena-length-impl
arena-lit-impl
arena-status-impl
swap-lits-impl
arena-lbd-impl
arena-pos-impl
update-lbd-impl
update-pos-impl
mark-garbage-impl
mark-used-impl
mark-unused-impl
marked-as-used-impl
MAX-LENGTH-SHORT-CLAUSE-impl
mop-arena-status-impl
end

end
theory *IsaSAT-Clauses*
imports *IsaSAT-Arena*
begin

Chapter 3

The memory representation: Manipulation of all clauses

Representation of Clauses

named-theorems *isasat-codegen* ‹*lemmas that should be unfolded to generate* (*efficient*) *code*›

type-synonym *clause-annot* = ‹*clause-status* × *nat* × *nat*›

type-synonym *clause-annots* = ‹*clause-annot list*›

definition *list-fmap-rel* :: ‹- ⇒ (*arena* × *nat clauses-l*) *set*› **where**
‹*list-fmap-rel vdom* = {(*arena*, *N*). *valid-arena arena N vdom*}›

lemma *nth-clauses-l*:
‹(*uncurry2* (*RETURN ooo* (λ*N i j*. *arena-lit N* (*i*+*j*))),
uncurry2 (*RETURN ooo* (λ*N i j*. *N* ∝ *i* ! *j*)))
∈ [λ((*N*, *i*), *j*). *i* ∈# *dom-m N* ∧ *j* < *length* (*N* ∝ *i*)]$_f$
list-fmap-rel vdom ×$_f$ *nat-rel* ×$_f$ *nat-rel* → ⟨*Id*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *list-fmap-rel-def arena-lifting*)

abbreviation *clauses-l-fmat* **where**
‹*clauses-l-fmat* ≡ *list-fmap-rel*›

type-synonym *vdom* = ‹*nat set*›

definition *fmap-rll* :: ‹(*nat*, *'a literal list* × *bool*) *fmap* ⇒ *nat* ⇒ *nat* ⇒ *'a literal*› **where**
[*simp*]: ‹*fmap-rll l i j* = *l* ∝ *i* ! *j*›

definition *fmap-rll-u* :: ‹(*nat*, *'a literal list* × *bool*) *fmap* ⇒ *nat* ⇒ *nat* ⇒ *'a literal*› **where**
[*simp*]: ‹*fmap-rll-u* = *fmap-rll*›

definition *fmap-rll-u64* :: ‹(*nat*, *'a literal list* × *bool*) *fmap* ⇒ *nat* ⇒ *nat* ⇒ *'a literal*› **where**
[*simp*]: ‹*fmap-rll-u64* = *fmap-rll*›

definition *fmap-length-rll-u* :: ‹(*nat*, *'a literal list* × *bool*) *fmap* ⇒ *nat* ⇒ *nat*› **where**
‹*fmap-length-rll-u l i* = *length-uint32-nat* (*l* ∝ *i*)›

declare *fmap-length-rll-u-def*[*symmetric*, *isasat-codegen*]

definition *fmap-length-rll-u64* :: ‹(*nat*, *'a literal list* × *bool*) *fmap* ⇒ *nat* ⇒ *nat*› **where**
‹*fmap-length-rll-u64 l i* = *length-uint32-nat* (*l* ∝ *i*)›

declare *fmap-length-rll-u-def*[*symmetric*, *isasat-codegen*]

definition *fmap-length-rll* :: ‹(*nat*, *'a literal list* × *bool*) *fmap* ⇒ *nat* ⇒ *nat*› **where**
[*simp*]: ‹*fmap-length-rll l i* = *length* (*l* ∝ *i*)›

definition *fmap-swap-ll* **where**
[*simp*]: ‹*fmap-swap-ll N i j f* = (*N*(*i* ↪ *swap* (*N* ∝ *i*) *j f*))›

From a performance point of view, appending several time a single element is less efficient than reserving a space that is large enough directly. However, in this case the list of clauses N is so large that there should not be any difference

definition *fm-add-new* **where**
‹*fm-add-new b C N0* = *do* {
let s = *length C* − *2*;
let lbd = *shorten-lbd s*;
let st = (*if b then AStatus IRRED 0 lbd else AStatus LEARNED 0 lbd*);
let l = *length N0*;
let N = (*if is-short-clause C then*
(((*N0* @ [*st*])) @ [*ASize s*]
else ((((*N0* @ [*APos 0*]) @ [*st*])) @ [*ASize* (*s*)]);
(*i*, *N*) ← *WHILE*$_T$ $^{\lambda(i,\ N).\ i < length\ C \longrightarrow length\ N < header\text{-}size\ C + length\ N0 + length\ C}$
(λ(*i*, *N*). *i* < *length C*)
(λ(*i*, *N*). *do* {
ASSERT(*i* < *length C*);
RETURN (*i*+*1*, *N* @ [*ALit* (*C* ! *i*)])
})
(*0*, *N*);
RETURN (*N*, *l* + *header-size C*)
}›

lemma *nth-append-clause*:
‹*a* < *length C* ⟹ *append-clause b C N* ! (*length N* + *header-size C* + *a*) = *ALit* (*C* ! *a*)›
unfolding *append-clause-def header-size-Suc-def append-clause-skeleton-def*
by (*auto simp*: *nth-Cons nth-append*)

lemma *fm-add-new-append-clause*:
‹*fm-add-new b C N* ≤ *RETURN* (*append-clause b C N*, *length N* + *header-size C*)›
unfolding *fm-add-new-def*
apply (*rewrite at* ‹*let* - = *length* - *in* -› *Let-def*)
apply (*refine-vcg WHILEIT-rule-stronger-inv*[**where** *R* = ‹*measure* (λ(*i*, -). *Suc* (*length C*) − *i*)› **and**
I' = ‹λ(*i*, *N'*). *N'* = *take* (*length N* + *header-size C* + *i*) (*append-clause b C N*) ∧
i ≤ *length C*›])
subgoal by *auto*
subgoal by (*auto simp*: *append-clause-def header-size-def*
append-clause-skeleton-def split: *if-splits*)
subgoal by (*auto simp*: *append-clause-def header-size-def*
append-clause-skeleton-def split: *if-splits*)
subgoal by *simp*
subgoal by *simp*
subgoal by *auto*

subgoal by (*auto simp*: *take-Suc-conv-app-nth nth-append-clause*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done

definition *fm-add-new-at-position*
:: ‹*bool* ⇒ *nat* ⇒ *'v clause-l* ⇒ *'v clauses-l* ⇒ *'v clauses-l*›
where
‹*fm-add-new-at-position b i C N* = *fmupd i* (*C*, *b*) *N*›

definition *AStatus-IRRED* **where**
‹*AStatus-IRRED* = *AStatus IRRED 0*›

definition *AStatus-IRRED2* **where**
‹*AStatus-IRRED2* = *AStatus IRRED 1*›

definition *AStatus-LEARNED* **where**
‹*AStatus-LEARNED* = *AStatus LEARNED 1*›

definition *AStatus-LEARNED2* **where**
‹*AStatus-LEARNED2* = *AStatus LEARNED 0*›

definition (**in** −)*fm-add-new-fast* **where**
[*simp*]: ‹*fm-add-new-fast* = *fm-add-new*›

lemma (**in** −)*append-and-length-code-fast*:
‹*length ba* ≤ *Suc* (*Suc unat32-max*) ⟹
2 ≤ *length ba* ⟹
length b ≤ *unat64-max* − (*unat32-max* + *5*) ⟹
(*aa*, *header-size ba*) ∈ *uint64-nat-rel* ⟹
(*ab*, *length b*) ∈ *uint64-nat-rel* ⟹
length b + *header-size ba* ≤ *unat64-max*›
by (*auto simp*: *unat64-max-def unat32-max-def header-size-def*)

definition (**in** −)*four-uint64-nat* **where**
[*simp*]: ‹*four-uint64-nat* = (*4* :: *nat*)›
definition (**in** −)*five-uint64-nat* **where**
[*simp*]: ‹*five-uint64-nat* = (*5* :: *nat*)›

definition *append-and-length-fast-code-pre* **where**
‹*append-and-length-fast-code-pre* ≡ λ((*b*, *C*), *N*). *length C* ≤ *unat32-max*+*2* ∧ *length C* ≥ *2* ∧
length N + *length C* + *MAX-HEADER-SIZE* ≤ *snat64-max*›

lemma *fm-add-new-alt-def*:
‹*fm-add-new b C N0* = *do* {
let s = *length C* − *2*;
let lbd = *shorten-lbd s*;
let st = (*if b then AStatus-IRRED lbd else AStatus-LEARNED2 lbd*);
let l = *length N0*;
let N =

```
      (if is-short-clause C
        then ((N0 @ [st])) @
          [ASize s]
        else (((N0 @ [APos 0]) @ [st])) @
          [ASize s]);
      (i, N) ←
        WHILE_T λ(i, N). i < length C ⟶ length N < header-size C + length N0 + length C
          (λ(i, N). i < length C)
          (λ(i, N). do {
              - ← ASSERT (i < length C);
              RETURN (i + 1, N @ [ALit (C ! i)])
            })
          (0, N);
        RETURN (N, l + header-size C)
    }›
  unfolding fm-add-new-def Let-def AStatus-LEARNED2-def AStatus-IRRED2-def
    AStatus-LEARNED-def AStatus-IRRED-def
  by auto

definition fmap-swap-ll-u64 where
  [simp]: ‹fmap-swap-ll-u64 = fmap-swap-ll›

definition fm-mv-clause-to-new-arena where
 ‹fm-mv-clause-to-new-arena C old-arena new-arena0 = do {
    ASSERT(arena-is-valid-clause-idx old-arena C);
   ASSERT(C ≥ (if (arena-length old-arena C) ≤ 4 then MIN-HEADER-SIZE else MAX-HEADER-SIZE));
   let st = C − (if (arena-length old-arena C) ≤ 4 then MIN-HEADER-SIZE else MAX-HEADER-SIZE);
    ASSERT(C +  (arena-length old-arena C) ≤ length old-arena);
    let en = C +  (arena-length old-arena C);
    (i, new-arena) ←
      WHILE_T λ(i, new-arena). i < en ⟶ length new-arena < length new-arena0 + (arena-length old-arena C) + (if  (arena-l
        (λ(i, new-arena). i < en)
        (λ(i, new-arena). do {
            ASSERT (i < length old-arena ∧ i < en);
            RETURN (i + 1, new-arena @ [old-arena ! i])
        })
        (st, new-arena0);
      RETURN (new-arena)
  }›

lemma valid-arena-append-clause-slice:
  assumes
    ‹valid-arena old-arena N vd› and
    ‹valid-arena new-arena N′ vd′› and
    ‹C ∈# dom-m N›
  shows ‹valid-arena (new-arena @ clause-slice old-arena N C)
    (fmupd (length new-arena + header-size (N ∝ C)) (N ∝ C, irred N C) N′)
    (insert (length new-arena + header-size (N ∝ C)) vd′)›
proof −
  define pos st lbd used where
    ‹pos = (if is-long-clause (N ∝ C) then arena-pos old-arena C − 2 else 0)› and
    ‹st = arena-status old-arena C› and
    ‹lbd = arena-lbd old-arena C› and
    ‹used = arena-used old-arena C›
  have ‹2 ≤ length (N ∝ C)›
```

unfolding *st-def used-def lbd-def*
append-clause-skeleton-def arena-status-def
xarena-status-def arena-used-def
xarena-used-def
arena-lbd-def xarena-lbd-def
using *arena-lifting*[*OF assms*(*1*,*3*)]
by (*auto simp*: *is-Status-def is-Pos-def is-Size-def*)
have
45: ‹*4* = (*Suc* (*Suc* (*Suc* (*Suc* *0*))))›
‹*5* = *Suc* (*Suc* (*Suc* (*Suc* (*Suc* *0*))))›
‹*3* = (*Suc* (*Suc* (*Suc* *0*)))›
‹*2* = (*Suc* (*Suc* *0*))›
by *auto*
have *sl*: ‹*clause-slice old-arena N C* =
(*if is-long-clause* (*N* ∝ *C*) *then* [*APos pos*]
else []) @
[*AStatus st used lbd*, *ASize* (*length* (*N* ∝ *C*) − *2*)] @
map ALit (*N* ∝ *C*)›
unfolding *st-def used-def lbd-def*
append-clause-skeleton-def arena-status-def
xarena-status-def arena-used-def
xarena-used-def
pos-def arena-pos-def
xarena-pos-def
arena-lbd-def xarena-lbd-def
arena-length-def xarena-length-def
using *arena-lifting*[*OF assms*(*1*,*3*)]
by (*auto simp*: *is-Status-def is-Pos-def is-Size-def*
header-size-def 45
slice-Suc-nth[*of* ‹*C* − *Suc* (*Suc* (*Suc* (*Suc* *0*)))›]
slice-Suc-nth[*of* ‹*C* − *Suc* (*Suc* (*Suc* *0*))›]
slice-Suc-nth[*of* ‹*C* − *Suc* (*Suc* *0*)›]
slice-Suc-nth[*of* ‹*C* − *Suc* *0*›]
SHIFTS-alt-def arena-length-def
arena-pos-def xarena-pos-def
arena-status-def xarena-status-def)

have ‹*2* ≤ *length* (*N* ∝ *C*)› **and**
‹*pos* ≤ *length* (*N* ∝ *C*) − *2*› **and**
‹*st* = *IRRED* ⟷ *irred N C*› **and**
‹*st* ≠ *DELETED*›
unfolding *st-def used-def lbd-def pos-def*
append-clause-skeleton-def st-def
using *arena-lifting*[*OF assms*(*1*,*3*)]
by (*cases* ‹*is-short-clause* (*N* ∝ *C*)›;
auto split: *arena-el.splits if-splits*
simp: *header-size-def arena-pos-def*; *fail*)+

then have ‹*valid-arena* (*append-clause-skeleton pos st used lbd* (*N* ∝ *C*) *new-arena*)
(*fmupd* (*length new-arena* + *header-size* (*N* ∝ *C*)) (*N* ∝ *C*, *irred N C*) *N'*)
(*insert* (*length new-arena* + *header-size* (*N* ∝ *C*)) *vd'*)›
apply −
by (*rule valid-arena-append-clause-skeleton*[*OF assms*(*2*), *of* ‹*N* ∝ *C*› - *st*
pos used lbd]) *auto*
moreover have
‹*append-clause-skeleton pos st used lbd* (*N* ∝ *C*) *new-arena* =

```
    new-arena @ clause-slice old-arena N C›
   by (auto simp: append-clause-skeleton-def sl)
  ultimately show ?thesis
   by auto
qed

lemma fm-mv-clause-to-new-arena:
  assumes ‹valid-arena old-arena N vd› and
   ‹valid-arena new-arena N' vd'› and
   ‹C ∈# dom-m N›
  shows ‹fm-mv-clause-to-new-arena C old-arena new-arena ≤
   SPEC(λnew-arena'.
     new-arena' = new-arena @ clause-slice old-arena N C ∧
     valid-arena (new-arena @ clause-slice old-arena N C)
       (fmupd (length new-arena + header-size (N ∝ C)) (N ∝ C, irred N C) N')
       (insert (length new-arena + header-size (N ∝ C)) vd'))›
proof −
  define st and en where
   ‹st = C − (if arena-length old-arena C ≤ 4 then MIN-HEADER-SIZE else MAX-HEADER-SIZE)›
and
   ‹en = C + arena-length old-arena C›
  have st:
   ‹st = C − header-size (N ∝ C)›
   using assms
   unfolding st-def
   by (auto simp: st-def header-size-def
      arena-lifting)
  show ?thesis
   using assms
   unfolding fm-mv-clause-to-new-arena-def st-def[symmetric]
    en-def[symmetric] Let-def
   apply (refine-vcg
    WHILEIT-rule-stronger-inv[where R = ‹measure (λ(i, N). en − i)› and
     I' = ‹λ(i, new-arena'). i ≤ C + length (N∝C) ∧ i ≥ st ∧
       new-arena' = new-arena @
   Misc.slice (C − header-size (N∝C)) i old-arena›])
   subgoal
    unfolding arena-is-valid-clause-idx-def
    by auto
   subgoal using arena-lifting(4)[OF assms(1)] by (auto
      dest!: arena-lifting(1)[of - N - C] simp: header-size-def split: if-splits)
   subgoal using arena-lifting(10, 4) en-def by auto
   subgoal
    by auto
   subgoal by auto
   subgoal
    using arena-lifting[OF assms(1,3)]
    by (auto simp: st)
   subgoal
    by (auto simp: st arena-lifting)
   subgoal
    using arena-lifting[OF assms(1,3)]
    by (auto simp: st en-def)
   subgoal
    using arena-lifting[OF assms(1,3)]
    by (auto simp: st en-def)
```

subgoal by *auto*
subgoal using *arena-lifting*[*OF assms*(*1*,*3*)]
by (*auto simp*: *slice-len-min-If en-def st-def header-size-def*)
subgoal
using *arena-lifting*[*OF assms*(*1*,*3*)]
by (*auto simp*: *st en-def*)
subgoal
using *arena-lifting*[*OF assms*(*1*,*3*)]
by (*auto simp*: *st*)
subgoal
by (*auto simp*: *st en-def arena-lifting*[*OF assms*(*1*,*3*)]
slice-append-nth)
subgoal by *auto*
subgoal by (*auto simp*: *en-def arena-lifting*)
subgoal
using *valid-arena-append-clause-slice*[*OF assms*]
by *auto*
done
qed

lemma *size-learned-clss-dom-m*: ‹*size* (*learned-clss-l N*) ≤ *size* (*dom-m N*)›
unfolding *ran-m-def*
apply (*rule order-trans*[*OF size-filter-mset-lesseq*])
by (*auto simp*: *ran-m-def*)

lemma *valid-arena-ge-length-clauses*:
assumes ‹*valid-arena arena N vdom*›
shows ‹*length arena* ≥ (∑ *C* ∈# *dom-m N*. *length* (*N* ∝ *C*) + *header-size* (*N* ∝ *C*))›
proof −
obtain *xs* **where**
mset-xs: ‹*mset xs* = *dom-m N*› **and** *sorted*: ‹*sorted xs*› **and** *dist*[*simp*]: ‹*distinct xs*› **and** *set-xs*: ‹*set xs* = *set-mset* (*dom-m N*)›
using *distinct-mset-dom distinct-mset-mset-distinct mset-sorted-list-of-multiset* **by** *fastforce*
then have *1*: ‹*set-mset* (*mset xs*) = *set xs*› **by** (*meson set-mset-mset*)

have *diff*: ‹*xs* ≠ [] ⟹ *a* ∈ *set xs* ⟹ *a* < *last xs* ⟹ *a* + *length* (*N* ∝ *a*) ≤ *last xs*› **for** *a*
using *valid-minimal-difference-between-valid-index*[*OF assms*, *of a* ‹*last xs*›]
mset-xs[*symmetric*] *sorted* **by** (*cases xs rule*: *rev-cases*; *auto simp*: *sorted-append*)
have ‹*set xs* ⊆ *set-mset* (*dom-m N*)›
using *mset-xs*[*symmetric*] **by** *auto*
then have ‹(∑ *A*∈*set xs*. *length* (*N* ∝ *A*) + *header-size* (*N* ∝ *A*)) ≤ *Max* (*insert 0* ((λ*A*. *A* + *length* (*N* ∝ *A*)) ‘ (*set xs*)))›
(**is** ‹*?P xs* ≤ *?Q xs*›)
using *sorted dist*
proof (*induction xs rule*: *rev-induct*)
case *Nil*
then show *?case* **by** *auto*
next
case (*snoc x xs*)
then have *IH*: ‹(∑ *A*∈*set xs*. *length* (*N* ∝ *A*) + *header-size* (*N* ∝ *A*))
≤ *Max* (*insert 0* ((λ*A*. *A* + *length* (*N* ∝ *A*)) ‘ *set xs*))› **and**
x-dom: ‹*x* ∈# *dom-m N*› **and**
x-max: ‹⋀*a*. *a* ∈ *set xs* ⟹ *x* > *a*› **and**
xs-N: ‹*set xs* ⊆ *set-mset* (*dom-m N*)›
apply (*clarsimp-all simp*: *sorted-append order.order-iff-strict sorted-wrt-append*)

apply (*metis* (*full-types*) *insert-subset order-less-le subsetI*)
apply (*metis* (*full-types*) *insert-subset order-less-le subsetI*)
apply (*meson distinct-sorted-append snoc.prems*(*2*) *snoc.prems*(*3*))
apply *blast*
done
have *x-ge*: ‹*header-size* (*N* ∝ *x*) ≤ *x*›
using *assms* ‹*x* ∈# *dom-m N*› *arena-lifting*(*1*) **by** *blast*
have *diff*: ‹*a* ∈ *set xs* ⟹ *a* + *length* (*N* ∝ *a*) + *header-size* (*N* ∝ *x*) ≤ *x*›
‹*a* ∈ *set xs* ⟹ *a* + *length* (*N* ∝ *a*) ≤ *x*› **for** *a*
using *valid-minimal-difference-between-valid-index*[*OF assms*, *of a x*]
x-max[*of a*] *xs-N x-dom* **by** *auto*

have ‹*?P* (*xs* @ [*x*]) ≤ *?P xs* + *length* (*N* ∝ *x*) + *header-size* (*N* ∝ *x*)›
using *snoc* **by** *auto*
also have ‹... ≤ *?Q xs* + (*length* (*N* ∝ *x*) + *header-size* (*N* ∝ *x*))›
using *IH* **by** *auto*
also have ‹... ≤ (*length* (*N* ∝ *x*) + *x*)›
by (*subst linordered-ab-semigroup-add-class.Max-add-commute2*[*symmetric*]; *auto intro*: *diff x-ge*)
also have ‹... = *Max* (*insert* (*x* + *length* (*N* ∝ *x*)) ((λ*x*. *x* + *length* (*N* ∝ *x*)) ‘ *set xs*))›
by (*subst eq-commute*)
(*auto intro*!: *linorder-class.Max-eqI intro*: *order-trans*[*OF diff*(*2*)])
finally show *?case* **by** *auto*
qed
also have ‹... ≤ (*if xs* = [] *then 0 else last xs* + *length* (*N* ∝ *last xs*))›
using *sorted distinct-sorted-append*[*of* ‹*butlast xs*› ‹*last xs*›] *dist*
by (*cases* ‹*xs*› *rule*: *rev-cases*)
(*auto intro*: *order-trans*[*OF diff*])
also have ‹... ≤ *length arena*›
using *arena-lifting*(*7*)[*OF assms*, *of* ‹*last xs*› ‹*length* (*N* ∝ *last xs*) − *1*›] *mset-xs*[*symmetric*] *assms*
by (*cases* ‹*xs*› *rule*: *rev-cases*) (*auto simp*: *arena-lifting*)
finally show *?thesis*
unfolding *mset-xs*[*symmetric*]
by (*subst distinct-sum-mset-sum*) *auto*
qed

lemma *valid-arena-size-dom-m-le-arena*: ‹*valid-arena arena N vdom* ⟹ *size* (*dom-m N*) ≤ *length arena*›
using *valid-arena-ge-length-clauses*[*of arena N vdom*]
ordered-comm-monoid-add-class.sum-mset-mono[*of* ‹*dom-m N*› ‹λ-. *1*›
‹λ*C*. *length* (*N* ∝ *C*) + *header-size* (*N* ∝ *C*)›]
by (*fastforce simp*: *header-size-def split*: *if-splits*)

end
theory *IsaSAT-Clauses-LLVM*
imports *IsaSAT-Clauses IsaSAT-Arena-LLVM*
begin

sepref-register *is-short-clause header-size fm-add-new-fast fm-mv-clause-to-new-arena*

abbreviation *clause-ll-assn* :: ‹*nat clause-l* ⇒ - ⇒ *assn*› **where**
‹*clause-ll-assn* ≡ *larray64-assn unat-lit-assn*›

sepref-def *is-short-clause-code*
is ‹*RETURN o is-short-clause*›
:: ‹ *clause-ll-assn*^{k} →$_a$ *bool1-assn*›
unfolding *is-short-clause-def*

by *sepref*

sepref-def *header-size-code*
is ‹*RETURN o header-size*›
:: ‹*clause-ll-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*›
unfolding *header-size-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *header-size-bound*: ‹*header-size x* $\leq$ *MAX-HEADER-SIZE*› **by** (*auto simp*: *header-size-def*)

lemma *fm-add-new-bounds1*: ⟦
length a2' $<$ *header-size baa* + *length b* + *length baa*;
length b + *length baa* + *MAX-HEADER-SIZE* $\leq$ *snat64-max* ⟧
$\Longrightarrow$ *Suc* (*length a2'*) $<$ *max-snat 64*

‹*length b* + *length baa* + *MAX-HEADER-SIZE* $\leq$ *snat64-max* $\Longrightarrow$ *length b* + *header-size baa* $<$ *max-snat 64*›
using *header-size-bound*[*of baa*]
by (*auto simp*: *max-snat-def snat64-max-def*)

sepref-def *append-and-length-fast-code*
is ‹*uncurry2 fm-add-new-fast*›
:: ‹[*append-and-length-fast-code-pre*]$_a$
bool1-assn^k $*_a$ *clause-ll-assn*^{k} $*_a$ (*arena-fast-assn*)^d $\rightarrow$
arena-fast-assn $\times_a$ *sint64-nat-assn*›
unfolding *fm-add-new-fast-def fm-add-new-def append-and-length-fast-code-pre-def*
apply (*rewrite at* ‹*APos* ⌑› *unat-const-fold*[**where** *'a=32*])+
apply (*rewrite at* ‹*length - − 2*› *annot-snat-unat-downcast*[**where** *'l=32*])

supply [*simp*] = *fm-add-new-bounds1*[*simplified*] *shorten-lbd-le*
apply (*rewrite at* ‹*AStatus -* ⌑› *unat-const-fold*[**where** *'a=2*])+
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *fm-mv-clause-to-new-arena-fast-code*
is ‹*uncurry2 fm-mv-clause-to-new-arena*›
:: ‹[λ((*n*, *arena*$_o$), *arena*). *length arena*$_o$ $\leq$ *snat64-max* $\wedge$ *length arena* + *arena-length arena*$_o$ *n* +
(*if arena-length arena*$_o$ *n* $\leq$ *4 then MIN-HEADER-SIZE else MAX-HEADER-SIZE*) $\leq$ *snat64-max*]$_a$
sint64-nat-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ *arena-fast-assn*^{d} $\rightarrow$ *arena-fast-assn*›
supply [[*goals-limit=1*]] *if-splits*[*split*]
unfolding *fm-mv-clause-to-new-arena-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

experiment begin
export-llvm
is-short-clause-code
header-size-code
append-and-length-fast-code
fm-mv-clause-to-new-arena-fast-code

end

end
theory *IsaSAT-Trail*
imports *IsaSAT-Literals*

begin

Chapter 4

Efficient Trail

Our trail contains several additional information compared to the simple trail:

- the (reversed) trail in an array (i.e., the trail in the same order as presented in "Automated Reasoning");
- the mapping from any *literal* (and not an atom) to its polarity;
- the mapping from a *atom* to its level or reason (in two different arrays);
- the current level of the state;
- the control stack.

We copied the idea from the mapping from a literals to it polarity instead of an atom to its polarity from a comment by Armin Biere in CaDiCal. We only observed a (at best) faint performance increase, but as it seemed slightly faster and does not increase the length of the formalisation, we kept it.
The control stack is the latest addition: it contains the positions of the decisions in the trail. It is mostly to enable fast restarts (since it allows to directly iterate over all decision of the trail), but might also slightly speed up backjumping (since we know how far we are going back in the trail). Remark that the control stack contains is not updated during the backjumping, but only *after* doing it (as we keep only the the beginning of it).

4.1 Types

type-synonym *trail-pol* =
  ‹*nat literal list* × *tri-bool list* × *nat list* × *nat list* × *nat* × *nat list* × *nat*›

definition *get-level-atm* **where**
  ‹*get-level-atm M L* = *get-level M* (*Pos L*)›

definition *polarity-atm* **where**
  ‹*polarity-atm M L* =
    (*if Pos L* ∈ *lits-of-l M then SET-TRUE*
    else if Neg L ∈ *lits-of-l M then SET-FALSE*
    else None)›

definition *defined-atm* :: ‹('*v*, *nat*) *ann-lits* ⇒ '*v* ⇒ *bool*› **where**

‹*defined-atm M L* = *defined-lit M* (*Pos L*)›

abbreviation *undefined-atm* **where**
‹*undefined-atm M L* ≡ ¬*defined-atm M L*›

4.2 Control Stack

inductive *control-stack* **where**
empty:
‹*control-stack* [] []› |
cons-prop:
‹*control-stack cs M* ⟹ *control-stack cs* (*Propagated L C* # *M*)› |
cons-dec:
‹*control-stack cs M* ⟹ *n* = *length M* ⟹ *control-stack* (*cs* @ [*n*]) (*Decided L* # *M*)›

inductive-cases *control-stackE*: ‹*control-stack cs M*›

lemma *control-stack-length-count-dec*:
‹*control-stack cs M* ⟹ *length cs* = *count-decided M*›
by (*induction rule*: *control-stack.induct*) *auto*

lemma *control-stack-le-length-M*:
‹*control-stack cs M* ⟹ *c*∈*set cs* ⟹ *c* < *length M*›
by (*induction rule*: *control-stack.induct*) *auto*

lemma *control-stack-propa*[*simp*]:
‹*control-stack cs* (*Propagated x21 x22* # *list*) ⟷ *control-stack cs list*›
by (*auto simp*: *control-stack.intros elim*: *control-stackE*)

lemma *control-stack-filter-map-nth*:
‹*control-stack cs M* ⟹ *filter is-decided* (*rev M*) = *map* (*nth* (*rev M*)) *cs*›
apply (*induction rule*: *control-stack.induct*)
subgoal by *auto*
subgoal for *cs M L C*
using *control-stack-le-length-M*[*of cs M*]
by (*auto simp*: *nth-append*)
subgoal for *cs M L*
using *control-stack-le-length-M*[*of cs M*]
by (*auto simp*: *nth-append*)
done

lemma *control-stack-empty-cs*[*simp*]: ‹*control-stack* [] *M* ⟷ *count-decided M* = *0*›
by (*induction M rule*:*ann-lit-list-induct*)
(*auto simp*: *control-stack.empty control-stack.cons-prop elim*: *control-stackE*)

This is an other possible definition. It is not inductive, which makes it easier to reason about appending (or removing) some literals from the trail. It is however much less clear if the definition is correct.

definition *control-stack′* **where**
‹*control-stack′ cs M* ⟷
(*length cs* = *count-decided M* ∧
(∀ *L*∈*set M*. *is-decided L* ⟶ (*cs* ! (*get-level M* (*lit-of L*) − *1*) < *length M* ∧
rev M!(*cs* ! (*get-level M* (*lit-of L*) − *1*)) = *L*)))›

lemma *control-stack-rev-get-lev*:

‹*control-stack cs M* ⟹
no-dup M ⟹ *L*∈*set M* ⟹ *is-decided L* ⟹ *rev M*!(*cs* ! (*get-level M* (*lit-of L*) − *1*)) = *L*›
apply (*induction arbitrary*: *L rule*: *control-stack.induct*)
subgoal by *auto*
subgoal for *cs M L C La*
using *control-stack-le-length-M*[*of cs M*] *control-stack-length-count-dec*[*of cs M*]
count-decided-ge-get-level[*of M* ‹*lit-of La*›]
apply (*auto simp*: *get-level-cons-if nth-append atm-of-eq-atm-of undefined-notin*)
by (*metis Suc-count-decided-gt-get-level Suc-less-eq Suc-pred count-decided-0-iff diff-is-0-eq*
le-SucI le-refl neq0-conv nth-mem)
subgoal for *cs M L*
using *control-stack-le-length-M*[*of cs M*] *control-stack-length-count-dec*[*of cs M*]
apply (*auto simp*: *nth-append get-level-cons-if atm-of-eq-atm-of undefined-notin*)
by (*metis Suc-count-decided-gt-get-level Suc-less-eq Suc-pred count-decided-0-iff diff-is-0-eq*
le-SucI le-refl neq0-conv)+
done

lemma *control-stack-alt-def-imp*:
‹*no-dup M* ⟹ (⋀*L*. *L* ∈*set M* ⟹ *is-decided L* ⟹ *cs* ! (*get-level M* (*lit-of L*) − *1*) < *length M* ∧
rev M!(*cs* ! (*get-level M* (*lit-of L*) − *1*)) = *L*) ⟹
length cs = *count-decided M* ⟹
control-stack cs M›
proof (*induction M arbitrary*: *cs rule*:*ann-lit-list-induct*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Decided L M*) **note** *IH* = *this*(*1*) **and** *n-d* = *this*(*2*) **and** *dec* = *this*(*3*) **and** *length* = *this*(*4*)
from *length* **obtain** *cs′ n* **where** *cs*[*simp*]: ‹*cs* = *cs′* @ [*n*]›
using *length* **by** (*cases cs rule*: *rev-cases*) *auto*
have [*simp*]: ‹*rev M* ! *n* ∈ *set M* ⟹ *is-decided* (*rev M* ! *n*) ⟹ *count-decided M* ≠ *0*›
by (*auto simp*: *count-decided-0-iff*)
have *dec′*: ‹*L′*∈*set M* ⟹ *is-decided L′* ⟹ *cs′* ! (*get-level M* (*lit-of L′*) − *1*) < *length M* ∧
rev M ! (*cs′* ! (*get-level M* (*lit-of L′*) − *1*)) = *L′*› **for** *L′*
using *dec*[*of L′*] *n-d length*
count-decided-ge-get-level[*of M* ‹*lit-of L′*›]
apply (*auto simp*: *get-level-cons-if atm-of-eq-atm-of undefined-notin*
split: *if-splits*)
apply (*auto simp*: *nth-append split*: *if-splits*)
done
have *le*: ‹*length cs′* = *count-decided M*›
using *length* **by** *auto*
have [*simp*]: ‹*n* = *length M*›
using *n-d dec*[*of* ‹*Decided L*›] *le undefined-notin*[*of M* ‹*rev M* ! *n*›] *nth-mem*[*of n* ‹*rev M*›]
by (*auto simp*: *nth-append split*: *if-splits*)
show *?case*
unfolding *cs*
apply (*rule control-stack.cons-dec*)
subgoal
apply (*rule IH*)
using *n-d dec′ le* **by** *auto*
subgoal by *auto*
done
next
case (*Propagated L m M*) **note** *IH* = *this*(*1*) **and** *n-d* = *this*(*2*) **and** *dec* = *this*(*3*) **and** *length* =
this(*4*)
have [*simp*]: ‹*rev M* ! *n* ∈ *set M* ⟹ *is-decided* (*rev M* ! *n*) ⟹ *count-decided M* ≠ *0*› **for** *n*

```
    by (auto simp: count-decided-0-iff)
  have dec': ‹L'∈set M ⟹ is-decided L' ⟹ cs ! (get-level M (lit-of L') − 1) < length M ∧
      rev M ! (cs ! (get-level M (lit-of L') − 1)) = L'› for L'
    using dec[of L'] n-d length
    count-decided-ge-get-level[of M ‹lit-of L'›]
    apply (cases L')
    apply (auto simp: get-level-cons-if atm-of-eq-atm-of undefined-notin
      split: if-splits)
    apply (auto simp: nth-append split: if-splits)
    done
  show ?case
    apply (rule control-stack.cons-prop)
    apply (rule IH)
    subgoal using n-d by auto
    subgoal using dec' by auto
    subgoal using length by auto
    done
qed

lemma control-stack-alt-def: ‹no-dup M ⟹ control-stack' cs M ⟷ control-stack cs M›
  using control-stack-alt-def-imp[of M cs] control-stack-rev-get-lev[of cs M]
    control-stack-length-count-dec[of cs M] control-stack-le-length-M[of cs M]
  unfolding control-stack'-def apply −
  apply (rule iffI)
  subgoal by blast
  subgoal
    using count-decided-ge-get-level[of M]
    by (metis One-nat-def Suc-count-decided-gt-get-level Suc-less-eq Suc-pred count-decided-0-iff
      less-imp-diff-less neq0-conv nth-mem)
  done

lemma control-stack-decomp:
  assumes
    decomp: ‹(Decided L # M1, M2) ∈ set (get-all-ann-decomposition M)› and
    cs: ‹control-stack cs M› and
    n-d: ‹no-dup M›
  shows ‹control-stack (take (count-decided M1) cs) M1›
proof −
  obtain M3 where M: ‹M = M3 @ M2 @ Decided L # M1›
    using decomp by auto
  define M2' where ‹M2' = M3 @ M2›
  have M: ‹M = M2' @ Decided L # M1›
    unfolding M M2'-def by auto
  have n-d1: ‹no-dup M1›
    using n-d no-dup-appendD unfolding M by auto
  have ‹control-stack' cs M›
    using cs
    apply (subst (asm) control-stack-alt-def[symmetric])
     apply (rule n-d)
    apply assumption
    done
  then have
    cs-M: ‹length cs = count-decided M› and
    L: ‹⋀L. L∈set M ⟹ is-decided L ⟹
      cs ! (get-level M (lit-of L) − 1) < length M ∧ rev M ! (cs ! (get-level M (lit-of L) − 1)) = L›
    unfolding control-stack'-def by auto
```

have *H*: ‹*L'* ∈ *set M1* ⟹ *undefined-lit M2'* (*lit-of L'*) ∧ *atm-of* (*lit-of L'*) ≠ *atm-of L*› **for** *L'*
using *n-d* **unfolding** *M*
by (*metis atm-of-eq-atm-of defined-lit-no-dupD*(1) *defined-lit-uminus lit-of.simps*(1)
no-dup-appendD no-dup-append-cons no-dup-cons undefined-notin)
have ‹*distinct M*›
using *no-dup-imp-distinct*[*OF n-d*] **.**
then have *K*: ‹*L'* ∈ *set M1* ⟹ *x* < *length M* ⟹ *rev M* ! *x* = *L'* ⟹ *x* < *length M1*› **for** *x L'*
unfolding *M* **apply** (*auto simp*: *nth-append nth-Cons split*: *if-splits nat.splits*)
by (*metis length-rev less-diff-conv local.H not-less-eq nth-mem set-rev undefined-notin*)
have *I*: ‹*L* ∈ *set M1* ⟹ *is-decided L* ⟹ *get-level M1* (*lit-of L*) > 0› **for** *L*
using *n-d* **unfolding** *M* **by** (*auto dest*!: *split-list*)
have *cs'*: ‹*control-stack'* (*take* (*count-decided M1*) *cs*) *M1*›
unfolding *control-stack'-def*
apply (*intro conjI ballI impI*)
subgoal using *cs-M* **unfolding** *M* **by** *auto*
subgoal for *L* **using** *n-d L*[*of L*] *H*[*of L*] *K*[*of L* ‹*cs* ! (*get-level M1* (*lit-of L*) − *Suc 0*)›]
count-decided-ge-get-level[*of* ‹*M1*› ‹*lit-of L*›] *I*[*of L*]
unfolding *M* **by** *auto*
subgoal for *L* **using** *n-d L*[*of L*] *H*[*of L*] *K*[*of L* ‹*cs* ! (*get-level M1* (*lit-of L*) − *Suc 0*)›]
count-decided-ge-get-level[*of* ‹*M1*› ‹*lit-of L*›] *I*[*of L*]
unfolding *M* **by** *auto*
done
show *?thesis*
apply (*subst control-stack-alt-def*[*symmetric*])
apply (*rule n-d1*)
apply (*rule cs'*)
done
qed

4.3 Encoding of the reasons

definition *DECISION-REASON* :: *nat* **where**
‹*DECISION-REASON* = *1*›

definition *ann-lits-split-reasons* **where**
‹*ann-lits-split-reasons* $\mathcal{A}$ = {((*M*, *reasons*), *M'*). *M* = *map lit-of* (*rev M'*) ∧
(∀ *L* ∈ *set M'*. *is-proped L* ⟶
reasons ! (*atm-of* (*lit-of L*)) = *mark-of L* ∧ *mark-of L* ≠ *DECISION-REASON*) ∧
(∀ *L* ∈ *set M'*. *is-decided L* ⟶ *reasons* ! (*atm-of* (*lit-of L*)) = *DECISION-REASON*) ∧
(∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *atm-of L* < *length reasons*)
}›

definition *zeroed-trail* :: ‹('*v*, *nat*) *ann-lits* ⇒ -› **where**
‹*zeroed-trail M zeroed* ⟷ *zeroed* ≤ *length M* ∧ (∀ *z* < *zeroed*. *is-proped* (*rev M* ! *z*) ∧ *mark-of* (*rev M* ! *z*) = *0*)›

lemma *zeored-trail-cons*[*simp*]:
‹*zeroed-trail M zeroed* ⟹ *zeroed-trail* (*L* # *M*) *zeroed*›
by (*auto simp*: *zeroed-trail-def nth-append split*: *if-splits*)

lemma *zeroed-trail-consD*:
assumes ‹*zeroed-trail* (*L* # *M*) *zeroed*› ‹*count-decided* (*L* # *M*) > *0*› ‹*no-dup* (*L* # *M*)›
shows ‹*zeroed-trail M zeroed*›
proof −
obtain *M1 M2 K* **where** ‹*L* # *M* = *M2* @ *Decided K* # *M1*› **and**

‹*get-level* (*L* # *M*) *K* = *1*›
by (*metis One-nat-def assms*(*2*) *assms*(*3*) *le-count-decided-decomp*)

then have ‹*zeroed* ≤ *length M*›
using *assms*(*1*) **unfolding** *zeroed-trail-def*
apply (*cases* ‹*zeroed* = *Suc* (*length M*)›)
apply (*auto elim*!: *list-Cons-eq-append-cases simp*: *nth-append split*: *if-splits*)
by (*metis* (*no-types*, *lifting*) *add.right-neutral add-Suc add-diff-cancel-left*′ *annotated-lit.disc*(*3*) *less-add-Suc2 nth-Cons-0 order-less-irrefl*)
then show ‹*?thesis*›
using *assms*(*1*) **unfolding** *zeroed-trail-def* **by** *auto*
qed

definition *trail-pol* :: ‹*nat multiset* ⇒ (*trail-pol* × (*nat*, *nat*) *ann-lits*) *set*› **where**
‹*trail-pol* $\mathcal{A}$ =
{((*M*′, *xs*, *lvls*, *reasons*, *k*, *cs*, *zeroed*), *M*). ((*M*′, *reasons*), *M*) ∈ *ann-lits-split-reasons* $\mathcal{A}$ ∧
no-dup M ∧
(∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *nat-of-lit L* < *length xs* ∧ *xs* ! (*nat-of-lit L*) = *polarity M L*) ∧
(∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *atm-of L* < *length lvls* ∧ *lvls* ! (*atm-of L*) = *get-level M L*) ∧
k = *count-decided M* ∧
(∀ *L*∈*set M*. *lit-of L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$) ∧
control-stack cs M ∧
zeroed-trail M zeroed ∧
isasat-input-bounded $\mathcal{A}$}›

4.4 Definition of the full trail

lemma *trail-pol-alt-def*:
‹*trail-pol* $\mathcal{A}$ = {((*M*′, *xs*, *lvls*, *reasons*, *k*, *cs*, *zeroed*), *M*).
((*M*′, *reasons*), *M*) ∈ *ann-lits-split-reasons* $\mathcal{A}$ ∧
no-dup M ∧
(∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *nat-of-lit L* < *length xs* ∧ *xs* ! (*nat-of-lit L*) = *polarity M L*) ∧
(∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *atm-of L* < *length lvls* ∧ *lvls* ! (*atm-of L*) = *get-level M L*) ∧
k = *count-decided M* ∧
(∀ *L*∈*set M*. *lit-of L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$) ∧
control-stack cs M ∧ *literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *M* ∧
length M < *unat32-max* ∧
length M ≤ *unat32-max div 2* + *1* ∧
count-decided M < *unat32-max* ∧
length M′ = *length M* ∧
M′ = *map lit-of* (*rev M*) ∧
zeroed-trail M zeroed ∧
isasat-input-bounded $\mathcal{A}$
}›
proof −
have [*intro*!]: ‹*length M* < *n* ⟹ *count-decided M* < *n*› **for** *M n*
using *length-filter-le*[*of is-decided M*]
by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-trail-def unat32-max-def count-decided-def*
simp del: *length-filter-le*
dest: *length-trail-unat32-max-div2*)
show *?thesis*
unfolding *trail-pol-def*
by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-trail-def unat32-max-def ann-lits-split-reasons-def*
dest: *length-trail-unat32-max-div2*

simp del: *isasat-input-bounded-def*)
qed

4.5 Code generation

4.5.1 Conversion between incomplete and complete mode

definition *trail-fast-of-slow* :: ‹(*nat*, *nat*) *ann-lits* ⇒ (*nat*, *nat*) *ann-lits*› **where**
‹*trail-fast-of-slow* = *id*›

definition *trail-pol-slow-of-fast* :: ‹*trail-pol* ⇒ *trail-pol*› **where**
‹*trail-pol-slow-of-fast* =
(λ(*M*, *val*, *lvls*, *reason*, *k*, *cs*). (*M*, *val*, *lvls*, *reason*, *k*, *cs*))›

definition *trail-slow-of-fast* :: ‹(*nat*, *nat*) *ann-lits* ⇒ (*nat*, *nat*) *ann-lits*› **where**
‹*trail-slow-of-fast* = *id*›

definition *trail-pol-fast-of-slow* :: ‹*trail-pol* ⇒ *trail-pol*› **where**
‹*trail-pol-fast-of-slow* =
(λ(*M*, *val*, *lvls*, *reason*, *k*, *cs*). (*M*, *val*, *lvls*, *reason*, *k*, *cs*))›

lemma *trail-pol-slow-of-fast-alt-def*:
‹*trail-pol-slow-of-fast M* = *M*›
by (*cases M*)
(*auto simp*: *trail-pol-slow-of-fast-def*)

lemma *trail-pol-fast-of-slow-trail-fast-of-slow*:
‹(*RETURN o trail-pol-fast-of-slow*, *RETURN o trail-fast-of-slow*)
∈ [λ*M*. (∀ *C L*. *Propagated L C* ∈ *set M* ⟶ *C* < *unat64-max*)]$_f$
trail-pol $\mathcal{A}$ → ⟨*trail-pol* $\mathcal{A}$⟩ *nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *trail-pol-def trail-pol-fast-of-slow-def*
trail-fast-of-slow-def)

lemma *trail-pol-slow-of-fast-trail-slow-of-fast*:
‹(*RETURN o trail-pol-slow-of-fast*, *RETURN o trail-slow-of-fast*)
∈ *trail-pol* $\mathcal{A}$ →$_f$ ⟨*trail-pol* $\mathcal{A}$⟩ *nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *trail-pol-def trail-pol-fast-of-slow-def*
trail-fast-of-slow-def trail-slow-of-fast-def
trail-pol-slow-of-fast-def)

lemma *trail-pol-same-length*[*simp*]: ‹(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$ ⟹ *length* (*fst M'*) = *length M*›
by (*auto simp*: *trail-pol-alt-def*)

definition *counts-maximum-level* **where**
‹*counts-maximum-level M C* = {*i*. *C* ≠ *None* ⟶ *i* = *card-max-lvl M* (*the C*)}›

lemma *counts-maximum-level-None*[*simp*]: ‹*counts-maximum-level M None* = *Collect* (λ-. *True*)›
by (*auto simp*: *counts-maximum-level-def*)

4.5.2 Level of a literal

definition *get-level-atm-pol-pre* **where**
‹*get-level-atm-pol-pre* = (λ((*M*, *xs*, *lvls*, *k*), *L*). *L* < *length lvls*)›

definition *get-level-atm-pol* :: ‹*trail-pol* ⇒ *nat* ⇒ *nat*› **where**
‹*get-level-atm-pol* = (λ(*M*, *xs*, *lvls*, *k*) *L*. *lvls* ! *L*)›

lemma *get-level-atm-pol-pre*:
assumes
‹*Pos L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
‹(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$›
shows ‹*get-level-atm-pol-pre* (*M'*, *L*)›
using *assms*
by (*auto 5 5 simp*: *trail-pol-def nat-lit-rel-def*
br-def get-level-atm-pol-pre-def intro!: *ext*)

lemma (**in** −) *get-level-get-level-atm*: ‹*get-level M L* = *get-level-atm M* (*atm-of L*)›
unfolding *get-level-atm-def*
by (*cases L*) (*auto simp*: *get-level-Neg-Pos*)

definition *get-level-pol* **where**
‹*get-level-pol M L* = *get-level-atm-pol M* (*atm-of L*)›

definition *get-level-pol-pre* **where**
‹*get-level-pol-pre* = (λ((*M*, *xs*, *lvls*, *k*), *L*). *atm-of L* < *length lvls*)›

lemma *get-level-pol-pre*:
assumes
‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
‹(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$›
shows ‹*get-level-pol-pre* (*M'*, *L*)›
using *assms*
by (*auto 5 5 simp*: *trail-pol-def nat-lit-rel-def*
br-def get-level-pol-pre-def intro!: *ext*)

lemma *get-level-get-level-pol*:
assumes
‹(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$› **and** ‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
shows ‹*get-level M L* = *get-level-pol M' L*›
using *assms*
by (*auto simp*: *get-level-pol-def get-level-atm-pol-def trail-pol-def*)

4.5.3 Current level

definition (**in** −) *count-decided-pol* **where**
‹*count-decided-pol* = (λ(-, -, -, -, *k*, -). *k*)›

lemma *count-decided-trail-ref*:
‹(*RETURN o count-decided-pol*, *RETURN o count-decided*) ∈ *trail-pol* $\mathcal{A}$ →$_f$ ⟨*nat-rel*⟩*nres-rel*›
by (*intro frefI nres-relI*) (*auto simp*: *trail-pol-def count-decided-pol-def*)

4.5.4 Polarity

definition (**in** −) *polarity-pol* :: ‹*trail-pol* ⇒ *nat literal* ⇒ *bool option*› **where**
‹*polarity-pol* = (λ(*M*, *xs*, *lvls*, *k*) *L*. *do* {
xs ! (*nat-of-lit L*)
})›

definition *polarity-pol-pre* **where**

‹*polarity-pol-pre* = (λ(*M*, *xs*, *lvls*, *k*) *L*. *nat-of-lit L* < *length xs*)›

lemma *polarity-pol-polarity*:
‹(*uncurry* (*RETURN oo polarity-pol*), *uncurry* (*RETURN oo polarity*)) ∈
[λ(*M*, *L*). *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$ *trail-pol* $\mathcal{A}$ ×$_f$ *Id* → ⟨⟨*bool-rel*⟩*option-rel*⟩*nres-rel*›
by (*intro nres-relI frefI*)
(*auto simp*: *trail-pol-def polarity-def polarity-pol-def*
dest!: *multi-member-split*)

lemma *polarity-pol-pre*:
‹(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$ ⟹ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹ *polarity-pol-pre M' L*›
by (*auto simp*: *trail-pol-def polarity-def polarity-pol-def polarity-pol-pre-def*
dest!: *multi-member-split*)

definition *mop-polarity-pol* :: ‹*trail-pol* ⇒ *nat literal* ⇒ *bool option nres*› **where**
‹*mop-polarity-pol* = (λ*M L*. *do* {
ASSERT(*polarity-pol-pre M L*);
RETURN (*polarity-pol M L*)
})›

4.5.5 Length of the trail

definition (**in** −) *isa-length-trail-pre* **where**
‹*isa-length-trail-pre* = (λ (*M'*, *xs*, *lvls*, *reasons*, *k*, *cs*). *length M'* ≤ *unat32-max*)›

definition (**in** −) *isa-length-trail* **where**
‹*isa-length-trail* = (λ (*M'*, *xs*, *lvls*, *reasons*, *k*, *cs*). *length-uint32-nat M'*)›

lemma *isa-length-trail-pre*:
‹(*M*, *M'*) ∈ *trail-pol* $\mathcal{A}$ ⟹ *isa-length-trail-pre M*›
by (*auto simp*: *isa-length-trail-def trail-pol-alt-def isa-length-trail-pre-def*)

lemma *isa-length-trail-length-u*:
‹(*RETURN o isa-length-trail*, *RETURN o length-uint32-nat*) ∈ *trail-pol* $\mathcal{A}$ →$_f$ ⟨*nat-rel*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *isa-length-trail-def trail-pol-alt-def*
intro!: *ASSERT-leI*)

definition *mop-isa-length-trail* **where**
‹*mop-isa-length-trail* = (λ(*M*). *do* {
ASSERT(*isa-length-trail-pre M*);
RETURN (*isa-length-trail M*)
})›

lemma *mop-isa-length-trail-length-u*:
‹(*mop-isa-length-trail*, *RETURN o length-uint32-nat*) ∈ *trail-pol* $\mathcal{A}$ →$_f$ ⟨*nat-rel*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *mop-isa-length-trail-def isa-length-trail-def dest*: *isa-length-trail-pre*
intro!: *ASSERT-leI*, *auto simp*: *trail-pol-alt-def*)

4.5.6 Consing elements

definition *cons-trail-Propagated-tr-pre* **where**
‹*cons-trail-Propagated-tr-pre* = (λ((*L*, *C*), (*M*, *xs*, *lvls*, *reasons*, *k*)). *nat-of-lit L* < *length xs* ∧
nat-of-lit (−*L*) < *length xs* ∧ *atm-of L* < *length lvls* ∧ *atm-of L* < *length reasons* ∧ *length M* <
unat32-max)›

definition *cons-trail-Propagated-tr* :: ‹*nat literal* ⇒ *nat* ⇒ *trail-pol* ⇒ *trail-pol nres*› **where**
‹*cons-trail-Propagated-tr* = (λ*L C* (*M′*, *xs*, *lvls*, *reasons*, *k*, *cs*, *zeroed*). *do* {
ASSERT(*cons-trail-Propagated-tr-pre* ((*L*, *C*), (*M′*, *xs*, *lvls*, *reasons*, *k*, *cs*, *zeroed*)));
RETURN (*M′* @ [*L*], *let xs* = *xs*[*nat-of-lit L* := *SET-TRUE*] *in xs*[*nat-of-lit* (−*L*) := *SET-FALSE*],
lvls[*atm-of L* := *k*], *reasons*[*atm-of L*:= *C*], *k*, *cs*, *zeroed*)})›

lemma *in-list-pos-neg-notD*: ‹*Pos* (*atm-of* (*lit-of La*)) ∉ *lits-of-l bc* ⟹
Neg (*atm-of* (*lit-of La*)) ∉ *lits-of-l bc* ⟹
La ∈ *set bc* ⟹ *False*›
by (*metis Neg-atm-of-iff Pos-atm-of-iff lits-of-def rev-image-eqI*)

lemma *cons-trail-Propagated-tr-pre*:
assumes ‹(*M′*, *M*) ∈ *trail-pol* $\mathcal{A}$› **and**
‹*undefined-lit M L*› **and**
‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
‹*C* ≠ *DECISION-REASON*›
shows ‹*cons-trail-Propagated-tr-pre* ((*L*, *C*), *M′*)›
using *assms*
by (*auto simp*: *trail-pol-alt-def ann-lits-split-reasons-def uminus-*$\mathcal{A}_{in}$*-iff*
cons-trail-Propagated-tr-pre-def
intro!: *ext*)

lemma *cons-trail-Propagated-tr*:
‹(*uncurry2* (*cons-trail-Propagated-tr*), *uncurry2* (*cons-trail-propagate-l*)) ∈
[λ((*L*, *C*), *M*). *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ *C* ≠ *DECISION-REASON*]$_f$
Id ×$_f$ *nat-rel* ×$_f$ *trail-pol* $\mathcal{A}$ → ⟨*trail-pol* $\mathcal{A}$⟩*nres-rel*›
unfolding *cons-trail-Propagated-tr-def cons-trail-propagate-l-def*
apply (*intro frefI nres-relI*)
subgoal for *x y*
using *cons-trail-Propagated-tr-pre*[*of* ‹*snd* (*x*)› ‹*snd* (*y*)› $\mathcal{A}$ ‹*fst* (*fst y*)› ‹*snd* (*fst y*)›]
unfolding *uncurry-def*
apply *refine-vcg*
subgoal by *auto*
subgoal
by (*cases* ‹*fst* (*fst y*)›)
(*auto simp add*: *trail-pol-def polarity-def uminus-lit-swap*
cons-trail-Propagated-tr-def Decided-Propagated-in-iff-in-lits-of-l nth-list-update′
ann-lits-split-reasons-def atms-of-$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$
uminus-$\mathcal{A}_{in}$*-iff atm-of-eq-atm-of*
intro!: *ASSERT-refine-right*
dest!: *in-list-pos-neg-notD dest*: *pos-lit-in-atms-of neg-lit-in-atms-of dest*!: *multi-member-split*
simp del: *nat-of-lit.simps*)
done
done

lemma *cons-trail-Propagated-tr2*:
‹(((*L*, *C*), *M*), ((*L′*, *C′*), *M′*)) ∈ *Id* ×$_f$ *Id* ×$_f$ *trail-pol* $\mathcal{A}$ ⟹ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹
C ≠ *DECISION-REASON* ⟹
cons-trail-Propagated-tr L C M
≤ ⇓ ({(*M″*, *M‴*). (*M″*, *M‴*) ∈ *trail-pol* $\mathcal{A}$ ∧ *M‴* = *Propagated L C* # *M′* ∧ *no-dup M‴*})
(*cons-trail-propagate-l L′ C′ M′*)›
using *cons-trail-Propagated-tr*[*THEN fref-to-Down-curry2*, *of* $\mathcal{A}$ *L′ C′ M′ L C M*]
unfolding *cons-trail-Propagated-tr-def cons-trail-propagate-l-def*

using *cons-trail-Propagated-tr-pre*[*of M M' A L C*]
unfolding *uncurry-def*
apply *refine-vcg*
subgoal by *auto*
subgoal
by (*auto simp*: *trail-pol-def*)
done

lemma *undefined-lit-count-decided-unat32-max*:
assumes
M-$\mathcal{L}_{all}$: ‹$\forall$ *L*∈*set M*. *lit-of L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and** *n-d*: ‹*no-dup M*› **and**
‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and** ‹*undefined-lit M L*› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
shows ‹*Suc* (*count-decided M*) ≤ *unat32-max*›
proof −
have *dist-atm-M*: ‹*distinct-mset* {#*atm-of* (*lit-of x*). *x* ∈# *mset M*#}›
using *n-d* **by** (*metis distinct-mset-mset-distinct mset-map no-dup-def*)
have *incl*: ‹*atm-of* ‘# *lit-of* ‘# *mset* (*Decided L # M*) ⊆# *remdups-mset* (*atm-of* ‘# $\mathcal{L}_{all}$ $\mathcal{A}$)›
apply (*subst distinct-subseteq-iff*[*THEN iffD1*])
using *assms dist-atm-M*
by (*auto simp*: *Decided-Propagated-in-iff-in-lits-of-l lits-of-def no-dup-distinct*
atm-of-eq-atm-of)
from *size-mset-mono*[*OF this*] **have** *1*: ‹*count-decided M* + *1* ≤ *size* (*remdups-mset* (*atm-of* ‘# $\mathcal{L}_{all}$ $\mathcal{A}$))›
using *length-filter-le*[*of is-decided M*] **unfolding** *unat32-max-def count-decided-def*
by (*auto simp del*: *length-filter-le*)
have *inj-on*: ‹*inj-on nat-of-lit* (*set-mset* (*remdups-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$)))›
by (*auto simp*: *inj-on-def*)
have *H*: ‹*xa* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹ *atm-of xa* ≤ *unat32-max div 2*› **for** *xa*
using *bounded*
by (*cases xa*) (*auto simp*: *unat32-max-def*)
have ‹*remdups-mset* (*atm-of* ‘# $\mathcal{L}_{all}$ $\mathcal{A}$) ⊆# *mset* [*0*..< *1* + (*unat32-max div 2*)]›
apply (*subst distinct-subseteq-iff*[*THEN iffD1*])
using *H distinct-image-mset-inj*[*OF inj-on*]
by (*force simp del*: *literal-of-nat.simps simp*: *distinct-mset-mset-set*
dest: *le-neq-implies-less*)+
note - = *size-mset-mono*[*OF this*]
moreover have ‹*size* (*nat-of-lit* ‘# *remdups-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$)) = *size* (*remdups-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
by *simp*
ultimately have *2*: ‹*size* (*remdups-mset* (*atm-of* ‘# ($\mathcal{L}_{all}$ $\mathcal{A}$))) ≤ *1* + *unat32-max div 2*›
by *auto*

show *?thesis*
using *1 2* **by** (*auto simp*: *unat32-max-def*)

from *size-mset-mono*[*OF incl*] **have** *1*: ‹*length M* + *1* ≤ *size* (*remdups-mset* (*atm-of* ‘# $\mathcal{L}_{all}$ $\mathcal{A}$))›
unfolding *unat32-max-def count-decided-def*
by (*auto simp del*: *length-filter-le*)
with *2* **have** ‹*length M* ≤ *unat32-max*›
by *auto*
qed

lemma *length-trail-unat32-max*:
assumes
M-$\mathcal{L}_{all}$: ‹$\forall$ *L*∈*set M*. *lit-of L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and** *n-d*: ‹*no-dup M*› **and**

```
    bounded: ‹isasat-input-bounded 𝒜›
  shows ‹length M ≤ unat32-max›
proof −
  have dist-atm-M: ‹distinct-mset {#atm-of (lit-of x). x ∈# mset M#}›
    using n-d by (metis distinct-mset-mset-distinct mset-map no-dup-def)
  have incl: ‹atm-of `# lit-of `# mset M ⊆# remdups-mset (atm-of `# ℒ_all 𝒜)›
    apply (subst distinct-subseteq-iff[THEN iffD1])
    using assms dist-atm-M
    by (auto simp: Decided-Propagated-in-iff-in-lits-of-l lits-of-def no-dup-distinct
        atm-of-eq-atm-of)

  have inj-on: ‹inj-on nat-of-lit (set-mset (remdups-mset (ℒ_all 𝒜)))›
    by (auto simp: inj-on-def)
  have H: ‹xa ∈# ℒ_all 𝒜 ⟹ atm-of xa ≤ unat32-max div 2› for xa
    using bounded
    by (cases xa) (auto simp: unat32-max-def)
  have ‹remdups-mset (atm-of `# ℒ_all 𝒜) ⊆# mset [0..< 1 + (unat32-max div 2)]›
    apply (subst distinct-subseteq-iff[THEN iffD1])
    using H distinct-image-mset-inj[OF inj-on]
    by (force simp del: literal-of-nat.simps simp: distinct-mset-mset-set
        dest: le-neq-implies-less)+
  note - = size-mset-mono[OF this]
  moreover have ‹size (nat-of-lit `# remdups-mset (ℒ_all 𝒜)) = size (remdups-mset (ℒ_all 𝒜))›
    by simp
  ultimately have 2: ‹size (remdups-mset (atm-of `# ℒ_all 𝒜)) ≤ 1 + unat32-max div 2›
    by auto
  from size-mset-mono[OF incl] have 1: ‹length M ≤ size (remdups-mset (atm-of `# ℒ_all 𝒜))›
    unfolding unat32-max-def count-decided-def
    by (auto simp del: length-filter-le)
  with 2 show ?thesis
    by (auto simp: unat32-max-def)
qed


definition last-trail-pol-pre where
  ‹last-trail-pol-pre = (λ(M, xs, lvls, reasons, k). atm-of (last M) < length reasons ∧ M ≠ [])›

definition (in −) last-trail-pol :: ‹trail-pol ⇒ (nat literal × nat option)› where
  ‹last-trail-pol = (λ(M, xs, lvls, reasons, k).
      let r = reasons ! (atm-of (last M)) in
      (last M, if r = DECISION-REASON then None else Some r))›


definition tl-trailt-tr :: ‹trail-pol ⇒ trail-pol› where
  ‹tl-trailt-tr = (λ(M', xs, lvls, reasons, k, cs, zeroed).
    let L = last M' in
    (butlast M',
    let xs = xs[nat-of-lit L := None] in xs[nat-of-lit (−L) := None],
    lvls[atm-of L := 0],
    reasons, if reasons ! atm-of L = DECISION-REASON then k−1 else k,
      if reasons ! atm-of L = DECISION-REASON then butlast cs else cs, zeroed))›

definition tl-trailt-tr-pre where
  ‹tl-trailt-tr-pre = (λ(M, xs, lvls, reason, k, cs, zeroed). M ≠ [] ∧ nat-of-lit(last M) < length xs ∧
      nat-of-lit(−last M) < length xs  ∧ atm-of (last M) < length lvls ∧
      atm-of (last M) < length reason ∧
```

```
    (reason ! atm-of (last M) = DECISION-REASON ⟶ k ≥ 1 ∧ cs ≠ []))›

lemma ann-lits-split-reasons-map-lit-of:
  ‹((M, reasons), M') ∈ ann-lits-split-reasons A ⟹ M = map lit-of (rev M')›
  by (auto simp: ann-lits-split-reasons-def)

lemma control-stack-dec-butlast:
  ‹control-stack b (Decided x1 # M's) ⟹ control-stack (butlast b) M's›
  by (cases b rule: rev-cases) (auto dest: control-stackE)

lemma tl-trail-tr:
  ‹((RETURN o tl-trailt-tr), (RETURN o tl)) ∈
    [λM. M ≠ [] ∧ count-decided M > 0]f trail-pol A → ⟨trail-pol A⟩nres-rel›
proof −
  show ?thesis
    apply (intro frefI nres-relI, rename-tac x y, case-tac ‹y›)
    subgoal by fast
    subgoal for M M' L M's
      unfolding trail-pol-def comp-def RETURN-refine-iff trail-pol-def Let-def
      apply clarify
      apply (intro conjI; clarify?; (intro conjI)?)
      subgoal
        by (auto simp: trail-pol-def polarity-atm-def tl-trailt-tr-def
            ann-lits-split-reasons-def Let-def)
      subgoal by (auto simp: trail-pol-def polarity-atm-def tl-trailt-tr-def)
      subgoal by (auto simp: polarity-atm-def tl-trailt-tr-def Let-def)
      subgoal
        by (cases ‹lit-of L›)
          (auto simp: polarity-def tl-trailt-tr-def Decided-Propagated-in-iff-in-lits-of-l
            uminus-lit-swap Let-def
            dest: ann-lits-split-reasons-map-lit-of)
      subgoal
        by (auto simp: polarity-atm-def tl-trailt-tr-def Let-def
           atm-of-eq-atm-of get-level-cons-if)
      subgoal
        by (auto simp: polarity-atm-def tl-trailt-tr-def
          atm-of-eq-atm-of get-level-cons-if Let-def
           dest!: ann-lits-split-reasons-map-lit-of)
      subgoal
        by (cases ‹L›)
          (auto simp: tl-trailt-tr-def in-Lall-atm-of-in-atms-of-iff ann-lits-split-reasons-def
            dest: no-dup-consistentD)
      subgoal
        by (auto simp: tl-trailt-tr-def)
      subgoal
        by (cases ‹L›)
          (auto simp: tl-trailt-tr-def in-Lall-atm-of-in-atms-of-iff ann-lits-split-reasons-def
            control-stack-dec-butlast
            dest: no-dup-consistentD)
      subgoal
        by (cases ‹L›)
         (auto simp: tl-trailt-tr-def in-Lall-atm-of-in-atms-of-iff ann-lits-split-reasons-def
            control-stack-dec-butlast
            dest: no-dup-consistentD zeroed-trail-consD)
      done
    done
```

qed

lemma *tl-trailt-tr-pre*:
assumes ‹$M \neq []$›
‹$(M', M) \in$ *trail-pol* $\mathcal{A}$›
shows ‹*tl-trailt-tr-pre* M'›
proof −
have [*simp*]: ‹$x \neq [] \Longrightarrow$ *is-decided* (*last* x) $\Longrightarrow$ *Suc* 0 $\leq$ *count-decided* x› **for** x
by (*cases x rule*: *rev-cases*) *auto*
show *?thesis*
using *assms*
by (*cases M*; *cases* ‹*hd M*›)
(*auto simp*: *trail-pol-def ann-lits-split-reasons-def uminus-*$\mathcal{A}_{in}$*-iff*
rev-map[*symmetric*] *hd-append hd-map tl-trailt-tr-pre-def simp del*: *rev-map*
intro!: *ext*)
qed

definition *tl-trail-propedt-tr* :: ‹*trail-pol* ⇒ *trail-pol*› **where**
‹*tl-trail-propedt-tr* = (λ(*M'*, *xs*, *lvls*, *reasons*, *k*, *cs*).
let L = *last M' in*
(*butlast M'*,
let xs = *xs*[*nat-of-lit L* := *None*] *in xs*[*nat-of-lit* (−*L*) := *None*],
lvls[*atm-of L* := 0],
reasons, *k*, *cs*))›

definition *tl-trail-propedt-tr-pre* **where**
‹*tl-trail-propedt-tr-pre* =
(λ(*M*, *xs*, *lvls*, *reason*, *k*, *cs*). *M* ≠ [] ∧ *nat-of-lit*(*last M*) < *length xs* ∧
nat-of-lit(−*last M*) < *length xs* ∧ *atm-of* (*last M*) < *length lvls* ∧
atm-of (*last M*) < *length reason*)›

lemma *tl-trail-propedt-tr-pre*:
assumes ‹$(M', M) \in$ *trail-pol* $\mathcal{A}$› **and**
‹$M \neq []$›
shows ‹*tl-trail-propedt-tr-pre* M'›
using *assms*
unfolding *trail-pol-def comp-def RETURN-refine-iff trail-pol-def Let-def*
tl-trail-propedt-tr-def tl-trail-propedt-tr-pre-def
apply *clarify*
apply (*cases M*; *intro conjI*; *clarify?*; (*intro conjI*)*?*)
subgoal
by (*auto simp*: *trail-pol-def polarity-atm-def tl-trailt-tr-def*
ann-lits-split-reasons-def Let-def)
subgoal
by (*auto simp*: *polarity-atm-def tl-trailt-tr-def*
atm-of-eq-atm-of get-level-cons-if Let-def
dest!: *ann-lits-split-reasons-map-lit-of*)
subgoal
by (*cases* ‹*hd M*›)
(*auto simp*: *tl-trailt-tr-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff ann-lits-split-reasons-def*
dest: *no-dup-consistentD*)
subgoal
by (*cases* ‹*hd M*›)
(*auto simp*: *tl-trailt-tr-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff ann-lits-split-reasons-def*
control-stack-dec-butlast
dest: *no-dup-consistentD*)

subgoal
by (*cases* ‹*hd M*›)
(*auto simp*: *tl-trailt-tr-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff ann-lits-split-reasons-def control-stack-dec-butlast*
dest: *no-dup-consistentD*)
done

definition (**in** −) *lit-of-hd-trail* **where**
‹*lit-of-hd-trail M* = *lit-of* (*hd M*)›

definition (**in** −) *lit-of-last-trail-pol* **where**
‹*lit-of-last-trail-pol* = (λ(*M*, -). *last M*)›

lemma *lit-of-last-trail-pol-lit-of-last-trail*:
‹(*RETURN o lit-of-last-trail-pol*, *RETURN o lit-of-hd-trail*) ∈
[λ*S*. *S* ≠ []]$_f$ *trail-pol* $\mathcal{A}$ → ⟨*Id*⟩*nres-rel*›
by (*auto simp*: *lit-of-hd-trail-def trail-pol-def lit-of-last-trail-pol-def*
ann-lits-split-reasons-def hd-map rev-map[*symmetric*] *last-rev*
intro!: *frefI nres-relI*)

4.5.7 Setting a new literal

definition *cons-trail-Decided* :: ‹*nat literal* ⇒ (*nat*, *nat*) *ann-lits* ⇒ (*nat*, *nat*) *ann-lits*› **where**
‹*cons-trail-Decided L M'* = *Decided L* # *M'*›

definition *cons-trail-Decided-tr* :: ‹*nat literal* ⇒ *trail-pol* ⇒ *trail-pol*› **where**
‹*cons-trail-Decided-tr* = (λ*L* (*M'*, *xs*, *lvls*, *reasons*, *k*, *cs*, *zeroed*). *do*{
let n = *length M' in*
(*M'* @ [*L*], *let xs* = *xs*[*nat-of-lit L* := *SET-TRUE*] *in xs*[*nat-of-lit* (−*L*) := *SET-FALSE*],
lvls[*atm-of L* := *k*+*1*], *reasons*[*atm-of L* := *DECISION-REASON*], *k*+*1*, *cs* @ [*n*], *zeroed*)})›

definition *cons-trail-Decided-tr-pre* **where**
‹*cons-trail-Decided-tr-pre* =
(λ(*L*, (*M*, *xs*, *lvls*, *reason*, *k*, *cs*, *zeroed*)). *nat-of-lit L* < *length xs* ∧ *nat-of-lit* (−*L*) < *length xs* ∧
atm-of L < *length lvls* ∧ *atm-of L* < *length reason* ∧ *length cs* < *unat32-max* ∧
Suc k ≤ *unat32-max* ∧ *length M* < *unat32-max*)›

lemma *length-cons-trail-Decided*[*simp*]:
‹*length* (*cons-trail-Decided L M*) = *Suc* (*length M*)›
by (*auto simp*: *cons-trail-Decided-def*)

lemma *cons-trail-Decided-tr*:
‹(*uncurry* (*RETURN oo cons-trail-Decided-tr*), *uncurry* (*RETURN oo cons-trail-Decided*)) ∈
[λ(*L*, *M*). *undefined-lit M L* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$ *Id* ×$_f$ *trail-pol* $\mathcal{A}$ → ⟨*trail-pol* $\mathcal{A}$⟩*nres-rel*›
by (*intro frefI nres-relI*, *rename-tac x y*, *case-tac* ‹*fst x*›)
(*auto simp*: *trail-pol-def polarity-def cons-trail-Decided-def uminus-lit-swap*
Decided-Propagated-in-iff-in-lits-of-l
cons-trail-Decided-tr-def nth-list-update' ann-lits-split-reasons-def
dest!: *in-list-pos-neg-notD multi-member-split*
intro: *control-stack.cons-dec*
simp del: *nat-of-lit.simps*)

lemma *cons-trail-Decided-tr-pre*:
assumes ‹(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$› **and**
‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and** ‹*undefined-lit M L*›

shows ‹*cons-trail-Decided-tr-pre* (*L*, *M'*)›
using *assms*
by (*auto simp*: *trail-pol-alt-def image-image ann-lits-split-reasons-def uminus-$\mathcal{A}_{in}$-iff*
cons-trail-Decided-tr-pre-def control-stack-length-count-dec
intro!: *ext undefined-lit-count-decided-unat32-max length-trail-unat32-max*)

4.5.8 Polarity: Defined or Undefined

definition (**in** −) *defined-atm-pol-pre* **where**
‹*defined-atm-pol-pre* = (λ(*M*, *xs*, *lvls*, *k*) *L*. *2*∗*L* < *length xs* ∧
2∗*L* ≤ *unat32-max*)›

definition (**in** −) *defined-atm-pol* **where**
‹*defined-atm-pol* = (λ(*M*, *xs*, *lvls*, *k*) *L*. ¬((*xs*!(*2*∗*L*)) = *None*))›

lemma *undefined-atm-code*:
‹(*uncurry* (*RETURN oo defined-atm-pol*), *uncurry* (*RETURN oo defined-atm*)) ∈
[λ(*M*, *L*). *Pos L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$ *trail-pol* $\mathcal{A}$ ×$_r$ *Id* → ⟨*bool-rel*⟩ *nres-rel*› (**is** *?A*) **and**
defined-atm-pol-pre:
‹(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$ ⟹ *L* ∈# $\mathcal{A}$ ⟹ *defined-atm-pol-pre M' L*›
proof −
have *H*: ‹*2*∗*L* < *length xs*› (**is** ‹*?length*›) **and**
none: ‹*defined-atm M L* ⟷ *xs* ! (*2*∗*L*) ≠ *None*› (**is** *?undef*) **and**
le: ‹*2*∗*L* ≤ *unat32-max*› (**is** *?le*)
if *L-N*: ‹*Pos L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and** *tr*: ‹((*M'*, *xs*, *lvls*, *k*), *M*) ∈ *trail-pol* $\mathcal{A}$›
for *M xs lvls k M' L*
proof −
have
‹*M'* = *map lit-of* (*rev M*)› **and**
‹∀ *L*∈#$\mathcal{L}_{all}$ $\mathcal{A}$. *nat-of-lit L* < *length xs* ∧ *xs* ! *nat-of-lit L* = *polarity M L*›
using *tr* **unfolding** *trail-pol-def ann-lits-split-reasons-def* **by** *fast+*
then have *L*: ‹*nat-of-lit* (*Pos L*) < *length xs*› **and**
xsL: ‹*xs* ! (*nat-of-lit* (*Pos L*)) = *polarity M* (*Pos L*)›
using *L-N* **by** (*auto dest*!: *multi-member-split*)
show *?length*
using *L* **by** *simp*
show *?undef*
using *xsL* **by** (*auto simp*: *polarity-def defined-atm-def*
Decided-Propagated-in-iff-in-lits-of-l split: *if-splits*)
show ‹*2*∗*L* ≤ *unat32-max*›
using *tr L-N* **unfolding** *trail-pol-def* **by** *auto*
qed
show *?A*
unfolding *defined-atm-pol-def*
by (*intro frefI nres-relI*) (*auto 5 5 simp*: *none H le intro*!: *ASSERT-leI*)
show ‹(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$ ⟹ *L* ∈# $\mathcal{A}$ ⟹ *defined-atm-pol-pre M' L*›
using *H le* **by** (*auto simp*: *defined-atm-pol-pre-def in-$\mathcal{L}_{all}$-atm-of-$\mathcal{A}_{in}$*)
qed

4.5.9 Reasons

definition *get-propagation-reason-pol* :: ‹*trail-pol* ⇒ *nat literal* ⇒ *nat option nres*› **where**
‹*get-propagation-reason-pol* = (λ(-, -, -, *reasons*, -) *L*. *do* {
ASSERT(*atm-of L* < *length reasons*);
let r = *reasons* ! *atm-of L*;
RETURN (*if r* = *DECISION-REASON then None else Some r*)})›

lemma *get-propagation-reason-pol*:
 ‹(*uncurry get-propagation-reason-pol*, *uncurry get-propagation-reason*) ∈
  [λ(*M*, *L*). *L* ∈ *lits-of-l M*]$_f$ *trail-pol* $\mathcal{A}$ ×$_r$ *Id* → ⟨⟨*nat-rel*⟩*option-rel*⟩ *nres-rel*›
 apply (*intro frefI nres-relI*)
 unfolding *lits-of-def*
 apply *clarify*
 apply (*rename-tac a aa ab ac b ba ad bb x*, *case-tac x*)
 by (*auto simp*: *get-propagation-reason-def get-propagation-reason-pol-def*
  trail-pol-def ann-lits-split-reasons-def lits-of-def assert-bind-spec-conv)

definition *get-propagation-reason-raw-pol* :: ‹*trail-pol* ⇒ *nat literal* ⇒ *nat nres*› **where**
 ‹*get-propagation-reason-raw-pol* = (λ(-, -, -, *reasons*, -) *L*. *do* {
  ASSERT(*atm-of L* < *length reasons*);
  let r = *reasons* ! *atm-of L*;
  RETURN r})›

The version *get-propagation-reason* can return the reason, but does not have to: it can be more suitable for specification (like for the conflict minimisation, where finding the reason is not mandatory).

The following version *always* returns the reasons if there is one. Remark that both functions are linked to the same code (but *get-propagation-reason* can be called first with some additional filtering later).

definition (**in** −) *get-the-propagation-reason*
 :: ‹('*v*, '*mark*) *ann-lits* ⇒ '*v literal* ⇒ '*mark option nres*›
where
 ‹*get-the-propagation-reason M L* = *SPEC*(λ*C*.
  (*C* ≠ *None* ⟷ *Propagated L* (*the C*) ∈ *set M*) ∧
  (*C* = *None* ⟷ *Decided L* ∈ *set M* ∨ *L* ∉ *lits-of-l M*))›

lemma *no-dup-Decided-PropedD*:
 ‹*no-dup ad* ⟹ *Decided L* ∈ *set ad* ⟹ *Propagated L C* ∈ *set ad* ⟹ *False*›
 by (*metis annotated-lit.distinct*(*1*) *in-set-conv-decomp lit-of.simps*(*1*) *lit-of.simps*(*2*)
  no-dup-appendD no-dup-cons undefined-notin xy-in-set-cases)

definition *get-the-propagation-reason-pol* :: ‹*trail-pol* ⇒ *nat literal* ⇒ *nat option nres*› **where**
 ‹*get-the-propagation-reason-pol* = (λ(-, *xs*, -, *reasons*, -, -) *L*. *do* {
  ASSERT(*atm-of L* < *length reasons*);
  ASSERT(*nat-of-lit L* < *length xs*);
  let r = *reasons* ! *atm-of L*;
  RETURN (*if xs* ! *nat-of-lit L* = *SET-TRUE* ∧ *r* ≠ *DECISION-REASON then Some r else None*)})›

lemma *get-the-propagation-reason-pol*:
 ‹(*uncurry get-the-propagation-reason-pol*, *uncurry get-the-propagation-reason*) ∈
  [λ(*M*, *L*). *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$ *trail-pol* $\mathcal{A}$ ×$_r$ *Id* → ⟨⟨*nat-rel*⟩*option-rel*⟩ *nres-rel*›
proof −
 have [*dest*]: ‹*no-dup bb* ⟹
  SET-TRUE = *polarity bb* (*Pos x1*) ⟹ *Pos x1* ∈ *lits-of-l bb* ∧ *Neg x1* ∉ *lits-of-l bb*› **for** *bb x1*
  by (*auto simp*: *polarity-def split*: *if-splits dest*: *no-dup-consistentD*)
 show *?thesis*
  apply (*intro frefI nres-relI*)
  unfolding *lits-of-def get-the-propagation-reason-def uncurry-def get-the-propagation-reason-pol-def*
  apply *clarify*

apply (*refine-vcg*)
subgoal
by (*auto simp*: *get-the-propagation-reason-def get-the-propagation-reason-pol-def Let-def trail-pol-def ann-lits-split-reasons-def assert-bind-spec-conv dest*!: *multi-member-split*[*of* - ‹$\mathcal{L}_{all}$ $\mathcal{A}$›])[]
subgoal
by (*auto simp*: *get-the-propagation-reason-def get-the-propagation-reason-pol-def Let-def trail-pol-def ann-lits-split-reasons-def assert-bind-spec-conv dest*!: *multi-member-split*[*of* - ‹$\mathcal{L}_{all}$ $\mathcal{A}$›])[]
subgoal for *a aa ab ac ad zeroed b ba ae bb*
apply (*cases* ‹*aa* ! *nat-of-lit ba* ≠ *SET-TRUE*›)
apply (*subgoal-tac* ‹*ba* ∉ *lits-of-l ae*›)
prefer *2*
subgoal
by (*auto simp*: *get-the-propagation-reason-def get-the-propagation-reason-pol-def Let-def trail-pol-def ann-lits-split-reasons-def assert-bind-spec-conv polarity-spec'*(*2*) *dest*: *multi-member-split*[*of* - ‹$\mathcal{L}_{all}$ $\mathcal{A}$›])[]
subgoal
by (*auto simp*: *lits-of-def dest*: *imageI*[*of* - - *lit-of*])

apply (*subgoal-tac* ‹*ba* ∈ *lits-of-l ae*›)
prefer *2*
subgoal
by (*auto simp*: *get-the-propagation-reason-def get-the-propagation-reason-pol-def Let-def trail-pol-def ann-lits-split-reasons-def assert-bind-spec-conv polarity-spec'*(*2*) *dest*: *multi-member-split*[*of* - ‹$\mathcal{L}_{all}$ $\mathcal{A}$›])[]
subgoal
apply (*auto simp*: *get-the-propagation-reason-def get-the-propagation-reason-pol-def Let-def trail-pol-def ann-lits-split-reasons-def assert-bind-spec-conv lits-of-def dest*!: *multi-member-split*[*of* - ‹$\mathcal{L}_{all}$ $\mathcal{A}$›])[]
apply (*case-tac x*; *auto*)
apply (*case-tac x*; *auto*)
done
done
done
qed

4.6 Direct access to elements in the trail

definition (**in** −) *rev-trail-nth* **where**
‹*rev-trail-nth M i* = *lit-of* (*rev M* ! *i*)›

definition (**in** −) *isa-trail-nth* :: ‹*trail-pol* ⇒ *nat* ⇒ *nat literal nres*› **where**
‹*isa-trail-nth* = (λ(*M*, -) *i*. *do* {
ASSERT(*i* < *length M*);
RETURN (*M* ! *i*)
})›

lemma *isa-trail-nth-rev-trail-nth*:
‹(*uncurry isa-trail-nth*, *uncurry* (*RETURN oo rev-trail-nth*)) ∈
[λ(*M*, *i*). *i* < *length M*]$_f$ *trail-pol* $\mathcal{A}$ ×$_r$ *nat-rel* → ⟨*Id*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *isa-trail-nth-def rev-trail-nth-def trail-pol-def ann-lits-split-reasons-def intro*!: *ASSERT-leI*)

We here define a variant of the trail representation, where the the control stack is out of sync of the trail (i.e., there are some leftovers at the end). This might make backtracking a little faster.

definition *trail-pol-no-CS* :: ‹*nat multiset* ⇒ (*trail-pol* × (*nat*, *nat*) *ann-lits*) *set*›
where
‹*trail-pol-no-CS* $\mathcal{A}$ =
{((*M'*, *xs*, *lvls*, *reasons*, *k*, *cs*, *zeroed*), *M*). ((*M'*, *reasons*), *M*) ∈ *ann-lits-split-reasons* $\mathcal{A}$ ∧
no-dup M ∧
(∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *nat-of-lit L* < *length xs* ∧ *xs* ! (*nat-of-lit L*) = *polarity M L*) ∧
(∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *atm-of L* < *length lvls* ∧ *lvls* ! (*atm-of L*) = *get-level M L*) ∧
(∀ *L*∈*set M*. *lit-of L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$) ∧
isasat-input-bounded $\mathcal{A}$ ∧
zeroed-trail M zeroed ∧
control-stack (*take* (*count-decided M*) *cs*) *M*
}›

definition *tl-trailt-tr-no-CS* :: ‹*trail-pol* ⇒ *trail-pol*› **where**
‹*tl-trailt-tr-no-CS* = (λ(*M'*, *xs*, *lvls*, *reasons*, *k*, *cs*, *zeroed*).
let L = *last M'* *in*
(*butlast M'*,
let xs = *xs*[*nat-of-lit L* := *None*] *in xs*[*nat-of-lit* (−*L*) := *None*],
lvls[*atm-of L* := *0*],
reasons, *k*, *cs*, *zeroed*))›

definition *tl-trailt-tr-no-CS-pre* **where**
‹*tl-trailt-tr-no-CS-pre* = (λ(*M*, *xs*, *lvls*, *reason*, *k*, *cs*). *M* ≠ [] ∧ *nat-of-lit*(*last M*) < *length xs* ∧
nat-of-lit(−*last M*) < *length xs* ∧ *atm-of* (*last M*) < *length lvls* ∧
atm-of (*last M*) < *length reason*)›

lemma *control-stack-take-Suc-count-dec-unstack*:
‹*control-stack* (*take* (*Suc* (*count-decided M's*)) *cs*) (*Decided x1* # *M's*) ⟹
control-stack (*take* (*count-decided M's*) *cs*) *M's*›
using *control-stack-length-count-dec*[*of* ‹*take* (*Suc* (*count-decided M's*)) *cs*› ‹*Decided x1* # *M's*›]
by (*auto simp*: *min-def take-Suc-conv-app-nth split*: *if-splits elim*: *control-stackE*)

lemma *tl-trailt-tr-no-CS-pre*:
assumes ‹(*M'*, *M*) ∈ *trail-pol-no-CS* $\mathcal{A}$› **and** ‹*M* ≠ []›
shows ‹*tl-trailt-tr-no-CS-pre M'*›
proof −
have [*simp*]: ‹*x* ≠ [] ⟹ *is-decided* (*last x*) ⟹ *Suc 0* ≤ *count-decided x*› **for** *x*
by (*cases x rule*: *rev-cases*) *auto*
show *?thesis*
using *assms*
unfolding *trail-pol-def comp-def RETURN-refine-iff trail-pol-no-CS-def Let-def*
tl-trailt-tr-no-CS-def tl-trailt-tr-no-CS-pre-def
by (*cases M*; *cases* ‹*hd M*›)
(*auto simp*: *trail-pol-no-CS-def ann-lits-split-reasons-def uminus-*$\mathcal{A}_{in}$*-iff*
rev-map[*symmetric*] *hd-append hd-map simp del*: *rev-map*
intro!: *ext*)
qed

lemma *tl-trail-tr-no-CS*:
‹((*RETURN o tl-trailt-tr-no-CS*), (*RETURN o tl*)) ∈
[λ*M*. *M* ≠ [] ∧ *count-decided M* > *0*]$_f$ *trail-pol-no-CS* $\mathcal{A}$ → ⟨*trail-pol-no-CS* $\mathcal{A}$⟩*nres-rel*›
apply (*intro frefI nres-relI*, *rename-tac x y*, *case-tac* ‹*y*›)
subgoal by *fast*

subgoal for *M M′ L M′s*
unfolding *trail-pol-def comp-def RETURN-refine-iff trail-pol-no-CS-def Let-def tl-trailt-tr-no-CS-def*
apply *clarify*
apply (*intro conjI*; *clarify?*; (*intro conjI*)*?*)
subgoal
by (*auto simp*: *trail-pol-def polarity-atm-def tl-trailt-tr-def ann-lits-split-reasons-def Let-def*)
subgoal by (*auto simp*: *trail-pol-def polarity-atm-def tl-trailt-tr-def*)
subgoal by (*auto simp*: *polarity-atm-def tl-trailt-tr-def Let-def*)
subgoal
by (*cases* ‹*lit-of L*›)
(*auto simp*: *polarity-def tl-trailt-tr-def Decided-Propagated-in-iff-in-lits-of-l uminus-lit-swap Let-def dest*: *ann-lits-split-reasons-map-lit-of*)
subgoal
by (*auto simp*: *polarity-atm-def tl-trailt-tr-def Let-def atm-of-eq-atm-of get-level-cons-if*)
subgoal
by (*auto simp*: *polarity-atm-def tl-trailt-tr-def atm-of-eq-atm-of get-level-cons-if Let-def dest*!: *ann-lits-split-reasons-map-lit-of*)
subgoal
by (*cases* ‹*L*›)
(*auto simp*: *tl-trailt-tr-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff ann-lits-split-reasons-def control-stack-dec-butlast dest*: *no-dup-consistentD ann-lits-split-reasons-map-lit-of*)
subgoal
by (*cases* ‹*L*›)
(*auto simp*: *tl-trailt-tr-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff ann-lits-split-reasons-def control-stack-dec-butlast control-stack-take-Suc-count-dec-unstack dest*: *no-dup-consistentD ann-lits-split-reasons-map-lit-of zeroed-trail-consD*)
subgoal
by (*cases* ‹*L*›)
(*auto simp*: *tl-trailt-tr-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff ann-lits-split-reasons-def control-stack-dec-butlast control-stack-take-Suc-count-dec-unstack dest*: *no-dup-consistentD ann-lits-split-reasons-map-lit-of*)
done
done

definition *trail-conv-to-no-CS* :: ‹(*nat*, *nat*) *ann-lits* ⇒ (*nat*, *nat*) *ann-lits*› **where**
‹*trail-conv-to-no-CS M* = *M*›

definition *trail-pol-conv-to-no-CS* :: ‹*trail-pol* ⇒ *trail-pol*› **where**
‹*trail-pol-conv-to-no-CS M* = *M*›

lemma *id-trail-conv-to-no-CS*:
‹(*RETURN o trail-pol-conv-to-no-CS*, *RETURN o trail-conv-to-no-CS*) ∈ *trail-pol* $\mathcal{A}$ →$_f$ ⟨*trail-pol-no-CS* $\mathcal{A}$⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *trail-pol-no-CS-def trail-conv-to-no-CS-def trail-pol-def control-stack-length-count-dec trail-pol-conv-to-no-CS-def intro*: *ext*)

definition *trail-conv-back* :: ‹*nat* ⇒ (*nat*, *nat*) *ann-lits* ⇒ (*nat*, *nat*) *ann-lits*› **where**
‹*trail-conv-back j M* = *M*›

definition (**in** −) *trail-conv-back-imp* :: ‹*nat* ⇒ *trail-pol* ⇒ *trail-pol nres*› **where**
‹*trail-conv-back-imp j* = (λ(*M*, *xs*, *lvls*, *reason*, -, *cs*, *zeroed*). *do* {
ASSERT(*j* ≤ *length cs*); *RETURN* (*M*, *xs*, *lvls*, *reason*, *j*, *take* (*j*) *cs*, *zeroed*)})›

lemma *trail-conv-back*:
‹(*uncurry trail-conv-back-imp*, *uncurry* (*RETURN oo trail-conv-back*))
∈ [λ(*k*, *M*). *k* = *count-decided M*]$_f$ *nat-rel* ×$_f$ *trail-pol-no-CS* $\mathcal{A}$ → ⟨*trail-pol* $\mathcal{A}$⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*force simp*: *trail-pol-no-CS-def trail-conv-to-no-CS-def trail-pol-def*
control-stack-length-count-dec trail-conv-back-def trail-conv-back-imp-def
intro: *ext intro*!: *ASSERT-refine-left*
dest: *control-stack-length-count-dec multi-member-split*)

definition (**in** −)*take-arl* **where**
‹*take-arl* = (λ*i* (*xs*, *j*). (*xs*, *i*))›

lemma *isa-trail-nth-rev-trail-nth-no-CS*:
‹(*uncurry isa-trail-nth*, *uncurry* (*RETURN oo rev-trail-nth*)) ∈
[λ(*M*, *i*). *i* < *length M*]$_f$ *trail-pol-no-CS* $\mathcal{A}$ ×$_r$ *nat-rel* → ⟨*Id*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *isa-trail-nth-def rev-trail-nth-def trail-pol-def ann-lits-split-reasons-def*
trail-pol-no-CS-def
intro!: *ASSERT-leI*)

lemma *trail-pol-no-CS-alt-def*:
‹*trail-pol-no-CS* $\mathcal{A}$ =
{((*M'*, *xs*, *lvls*, *reasons*, *k*, *cs*, *zeroed*), *M*). ((*M'*, *reasons*), *M*) ∈ *ann-lits-split-reasons* $\mathcal{A}$ ∧
no-dup M ∧
(∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *nat-of-lit L* < *length xs* ∧ *xs* ! (*nat-of-lit L*) = *polarity M L*) ∧
(∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *atm-of L* < *length lvls* ∧ *lvls* ! (*atm-of L*) = *get-level M L*) ∧
(∀ *L*∈*set M*. *lit-of L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$) ∧
control-stack (*take* (*count-decided M*) *cs*) *M* ∧ *literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *M* ∧
length M < *unat32-max* ∧
length M ≤ *unat32-max div 2* + *1* ∧
count-decided M < *unat32-max* ∧
length M' = *length M* ∧
zeroed-trail M zeroed ∧
isasat-input-bounded $\mathcal{A}$ ∧
M' = *map lit-of* (*rev M*)
}›
proof −
have [*intro*!]: ‹*length M* < *n* ⟹ *count-decided M* < *n*› **for** *M n*
using *length-filter-le*[*of is-decided M*]
by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-trail-def unat32-max-def count-decided-def*
simp del: *length-filter-le*
dest: *length-trail-unat32-max-div2*)
show *?thesis*
unfolding *trail-pol-no-CS-def*
by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-trail-def unat32-max-def ann-lits-split-reasons-def*
dest: *length-trail-unat32-max-div2*
simp del: *isasat-input-bounded-def*)
qed

lemma *isa-length-trail-length-u-no-CS*:
 ‹(*RETURN o isa-length-trail*, *RETURN o length-uint32-nat*) ∈ *trail-pol-no-CS* $\mathcal{A}$ →$_f$ ⟨*nat-rel*⟩*nres-rel*›
 by (*intro frefI nres-relI*)
  (*auto simp*: *isa-length-trail-def trail-pol-no-CS-alt-def ann-lits-split-reasons-def*
   intro!: *ASSERT-leI*)

lemma *control-stack-is-decided*:
 ‹*control-stack cs M* ⟹ *c*∈*set cs* ⟹ *is-decided* ((*rev M*)!*c*)›
 by (*induction arbitrary*: *c rule*: *control-stack.induct*) (*auto simp*: *nth-append*
   dest: *control-stack-le-length-M*)

lemma *control-stack-distinct*:
 ‹*control-stack cs M* ⟹ *distinct cs*›
 by (*induction rule*: *control-stack.induct*) (*auto simp*: *nth-append*
   dest: *control-stack-le-length-M*)

lemma *control-stack-level-control-stack*:
 assumes
  cs: ‹*control-stack cs M*› **and**
  n-d: ‹*no-dup M*› **and**
  i: ‹*i* < *length cs*›
 shows ‹*get-level M* (*lit-of* (*rev M* ! (*cs* ! *i*))) = *Suc i*›
proof −
 define *L* **where** ‹*L* = *rev M* ! (*cs* ! *i*)›
 have *csi*: ‹*cs* ! *i* < *length M*›
  using *cs i* **by** (*auto intro*: *control-stack-le-length-M*)
 then have *L-M*: ‹*L* ∈ *set M*›
  using *nth-mem*[*of* ‹*cs* !*i*› ‹*rev M*›] **unfolding** *L-def* **by** (*auto simp del*: *nth-mem*)
 have *dec-L*: ‹*is-decided L*›
  using *control-stack-is-decided*[*OF cs*] *i* **unfolding** *L-def* **by** *auto*
 then have ‹*rev M*!(*cs* ! (*get-level M* (*lit-of L*) − *1*)) = *L*›
  using *control-stack-rev-get-lev*[*OF cs n-d L-M*] **by** *auto*
 moreover have ‹*distinct M*›
  using *no-dup-distinct*[*OF n-d*] **unfolding** *mset-map*[*symmetric*] *distinct-mset-mset-distinct*
  by (*rule distinct-mapI*)

 moreover have *lev0*: ‹*get-level M* (*lit-of L*) ≥ *1*›
  using *split-list*[*OF L-M*] *n-d dec-L* **by** (*auto simp*: *get-level-append-if*)
 moreover have ‹*cs* ! (*get-level M* (*lit-of* (*rev M* ! (*cs* ! *i*))) − *Suc 0*) < *length M*›
  using *control-stack-le-length-M*[*OF cs*,
    of ‹*cs* ! (*get-level M* (*lit-of* (*rev M* ! (*cs* ! *i*))) − *Suc 0*)›, *OF nth-mem*]
   control-stack-length-count-dec[*OF cs*] *count-decided-ge-get-level*[*of M*
    ‹*lit-of* (*rev M* ! (*cs* ! *i*))›] *lev0*
  by (*auto simp*: *L-def*)
 ultimately have ‹*cs* ! (*get-level M* (*lit-of L*) − *1*) = *cs* ! *i*›
  using *nth-eq-iff-index-eq*[*of* ‹*rev M*›] *csi* **unfolding** *L-def* **by** *auto*
 then have ‹*i* = *get-level M* (*lit-of L*) − *1*›
  using *nth-eq-iff-index-eq*[*OF control-stack-distinct*[*OF cs*], *of i* ‹*get-level M* (*lit-of L*) − *1*›]
   i lev0 count-decided-ge-get-level[*of M* ‹*lit-of* (*rev M* ! (*cs* ! *i*))›]
   control-stack-length-count-dec[*OF cs*]
  by (*auto simp*: *L-def*)
 then show *?thesis* **using** *lev0* **unfolding** *L-def*[*symmetric*] **by** *auto*
qed

definition *get-pos-of-level-in-trail* **where**
‹*get-pos-of-level-in-trail* M_0 *lev* =
SPEC(λi. $i <$ *length* $M_0 \wedge$ *is-decided* (*rev* $M_0!i$) $\wedge$ *get-level* M_0 (*lit-of* (*rev* $M_0!i$)) = *lev*+*1*)›

definition (**in** −) *get-pos-of-level-in-trail-imp* :: ‹*trail-pol* ⇒ *nat* ⇒ -› **where**
‹*get-pos-of-level-in-trail-imp* = (λ(*M'*, *xs*, *lvls*, *reasons*, *k*, *cs*, *zeroed*) *lev*. *do* {
ASSERT(*lev* < *length cs*);
RETURN (*cs* ! *lev*)
})›
definition *get-pos-of-level-in-trail-pre* **where**
‹*get-pos-of-level-in-trail-pre* = (λ(*M*, *lev*). *lev* < *count-decided M*)›

lemma *get-pos-of-level-in-trail-imp-get-pos-of-level-in-trail*:
‹(*uncurry get-pos-of-level-in-trail-imp*, *uncurry get-pos-of-level-in-trail*) ∈
[*get-pos-of-level-in-trail-pre*]$_f$ *trail-pol-no-CS* $\mathcal{A}$ ×$_f$ *nat-rel* → ⟨*nat-rel*⟩*nres-rel*›
apply (*intro nres-relI frefI*)
unfolding *get-pos-of-level-in-trail-imp-def uncurry-def get-pos-of-level-in-trail-def*
get-pos-of-level-in-trail-pre-def
apply *clarify*
apply (*rule ASSERT-leI*)
subgoal
by (*auto simp*: *trail-pol-no-CS-alt-def dest*!: *control-stack-length-count-dec*)
subgoal for *a aa ab ac ad b ba ae bb*
by (*auto simp*: *trail-pol-no-CS-def control-stack-length-count-dec in-set-take-conv-nth*
intro!: *control-stack-le-length-M control-stack-is-decided*
dest: *control-stack-level-control-stack*)
done

lemma *get-pos-of-level-in-trail-imp-get-pos-of-level-in-trail-CS*:
‹(*uncurry get-pos-of-level-in-trail-imp*, *uncurry get-pos-of-level-in-trail*) ∈
[*get-pos-of-level-in-trail-pre*]$_f$ *trail-pol* $\mathcal{A}$ ×$_f$ *nat-rel* → ⟨*nat-rel*⟩*nres-rel*›
apply (*intro nres-relI frefI*)
unfolding *get-pos-of-level-in-trail-imp-def uncurry-def get-pos-of-level-in-trail-def*
get-pos-of-level-in-trail-pre-def
apply *clarify*
apply (*rule ASSERT-leI*)
subgoal
by (*auto simp*: *trail-pol-def dest*!: *control-stack-length-count-dec*)
subgoal for *a aa ab ac ad b ba ae bb*
by (*auto simp*: *trail-pol-def control-stack-length-count-dec in-set-take-conv-nth*
intro!: *control-stack-le-length-M control-stack-is-decided*
dest: *control-stack-level-control-stack*)
done

lemma *lit-of-last-trail-pol-lit-of-last-trail-no-CS*:
‹(*RETURN o lit-of-last-trail-pol*, *RETURN o lit-of-hd-trail*) ∈
[λ*S*. *S* ≠ []]$_f$ *trail-pol-no-CS* $\mathcal{A}$ → ⟨*Id*⟩*nres-rel*›
by (*auto simp*: *lit-of-hd-trail-def trail-pol-no-CS-def lit-of-last-trail-pol-def*
ann-lits-split-reasons-def hd-map rev-map[*symmetric*] *last-rev*
intro!: *frefI nres-relI*)

definition *trail-height-before-conflict* :: ‹*trail-pol* ⇒ *nat nres*› **where**
‹*trail-height-before-conflict* = (λ(*M'*, *xs*, *lvls*, *reasons*, *k*, *cs*, *zeroed*). *do*{
if k = *0 then RETURN 0 else do* {
let k' = *k*−*1*;

```
      ASSERT (k' < length cs);
      RETURN (cs ! k')
    }
  })⟩
```

definition *trail-height-before-conflict-spec* **where**
⟨*trail-height-before-conflict-spec - = RES* (*UNIV* :: *nat set*)⟩

lemma *trail-height-before-conflict*:
⟨(*trail-height-before-conflict*, *trail-height-before-conflict-spec*) ∈ *trail-pol* $\mathcal{A}$ →$_f$ ⟨*nat-rel*⟩*nres-rel*⟩
unfolding *trail-height-before-conflict-def trail-height-before-conflict-spec-def Let-def*
by (*intro frefI*, *refine-vcg*)
(*auto simp*: *trail-pol-def control-stack-length-count-dec*
intro!: *frefI ASSERT-leI*)

definition *trail-zeroed-until* :: ⟨*trail-pol* ⇒ *nat*⟩ **where**
⟨*trail-zeroed-until* = (λ(*M'*, *xs*, *lvls*, *reasons*, *k*, *cs*, *zeroed*). *zeroed*)⟩

definition *trail-set-zeroed-until* :: ⟨*nat* ⇒ *trail-pol* ⇒ *trail-pol*⟩ **where**
⟨*trail-set-zeroed-until* = (λ*zeroed* (*M'*, *xs*, *lvls*, *reasons*, *k*, *cs*, -). (*M'*, *xs*, *lvls*, *reasons*, *k*, *cs*, *zeroed*))⟩

lemma *trail-set-zeroed-until-rel*:
⟨(*M*,*M'*) ∈ *trail-pol* $\mathcal{A}$ ⟹*zeroed-trail M' z* ⟹ (*trail-set-zeroed-until z M*, *M'*) ∈ (*trail-pol* $\mathcal{A}$)⟩
by (*auto simp*: *trail-pol-def trail-set-zeroed-until-def*)

end
theory *IsaSAT-Options*
imports *IsaSAT-Literals*
begin

4.7 Options

We define the options from our SAT solver. Using options has several advantages: it is much easier to change the value (instead of recompiling everything from scratch the complete Isabelle development) and it is easier to change.

We hide the options inside a datatype to make sure Isabelle does not split the the component to make goals even less readable.

4.7.1 Definition

type-synonym *opts-target* = ⟨*3 word*⟩

datatype *opts* =
IsaOptions (*opts-restart*: *bool*)
(*opts-reduce*: *bool*)
(*opts-unbounded-mode*: *bool*)
(*opts-minimum-between-restart*: ⟨*64 word*⟩)
(*opts-restart-coeff1*: ⟨*64 word*⟩)
(*opts-restart-coeff2*: *nat*)
(*opts-target*: ⟨*opts-target*⟩)
(*opts-fema*: ⟨*64 word*⟩)
(*opts-sema*: ⟨*64 word*⟩)
(*opts-GC-units-lim*: ⟨*64 word*⟩)

(*opts-subsumption*: *bool*)

definition *TARGET-NEVER* :: ‹*opts-target*› **where**
‹*TARGET-NEVER* = *0*›

definition *TARGET-STABLE-ONLY* :: ‹*opts-target*› **where**
‹*TARGET-STABLE-ONLY* = *1*›

definition *TARGET-ALWAYS* :: ‹*opts-target*› **where**
‹*TARGET-ALWAYS* = *2*›

4.7.2 Refinement

type-synonym *opts-ref* =
‹*bool* × *bool* × *bool* × *64 word* × *64 word* × *nat* × *opts-target* × *64 word* × *64 word* × *64 word* × *bool*›

definition *opts-rel* :: ‹(*opts-ref* × *opts*) *set*› **where**
‹*opts-rel* = {(*S*, *T*). *S* = (*opts-restart T*, *opts-reduce T*, *opts-unbounded-mode T*,
opts-minimum-between-restart T, *opts-restart-coeff1 T*, *opts-restart-coeff2 T*,
opts-target T, *opts-fema T*, *opts-sema T*, *opts-GC-units-lim T*, *opts-subsumption T*)}›

fun *opts-rel-restart* :: ‹*opts-ref* ⇒ *bool*› **where**
‹*opts-rel-restart* (*res*, *red*, *unbd*, *mini*, *res1*, *res2*) = *res*›

lemma *opts-rel-restart*:
‹(*opts-rel-restart*, *opts-restart*) ∈ *opts-rel* → *bool-rel*›
by (*auto simp*: *opts-rel-def intro*!: *frefI*)

fun *opts-rel-reduce* :: ‹*opts-ref* ⇒ *bool*› **where**
‹*opts-rel-reduce* (*res*, *red*, *unbd*, *mini*, *res1*, *res2*) = *red*›

lemma *opts-rel-reduce*:
‹(*opts-rel-reduce*, *opts-reduce*) ∈ *opts-rel* → *bool-rel*›
by (*auto simp*: *opts-rel-def intro*!: *frefI*)

fun *opts-rel-unbounded-mode* :: ‹*opts-ref* ⇒ *bool*› **where**
‹*opts-rel-unbounded-mode* (*res*, *red*, *unbd*, *mini*, *res1*, *res2*) = *unbd*›

lemma *opts-rel-unbounded-mode*:
‹(*opts-rel-unbounded-mode*, *opts-unbounded-mode*) ∈ *opts-rel* → *bool-rel*›
by (*auto simp*: *opts-rel-def intro*!: *frefI*)

fun *opts-rel-miminum-between-restart* :: ‹*opts-ref* ⇒ *64 word*› **where**
‹*opts-rel-miminum-between-restart* (*res*, *red*, *unbd*, *mini*, *res1*, *res2*) = *mini*›

lemma *opts-rel-miminum-between-restart*:
‹(*opts-rel-miminum-between-restart*, *opts-minimum-between-restart*) ∈ *opts-rel* → *Id*›
by (*auto simp*: *opts-rel-def intro*!: *frefI*)

fun *opts-rel-restart-coeff1* :: ‹*opts-ref* ⇒ *64 word*› **where**
‹*opts-rel-restart-coeff1* (*res*, *red*, *unbd*, *mini*, *res1*, *res2*) = *res1*›

lemma *opts-rel-restart-coeff1*:
‹(*opts-rel-restart-coeff1*, *opts-restart-coeff1*) ∈ *opts-rel* → *Id*›

by (*auto simp*: *opts-rel-def intro*!: *frefI*)

fun *opts-rel-restart-coeff2* :: ‹*opts-ref* ⇒ *nat*› **where**
‹*opts-rel-restart-coeff2* (*res*, *red*, *unbd*, *mini*, *res1*, *res2*, *target*) = *res2*›

lemma *opts-rel-restart-coeff2*:
‹(*opts-rel-restart-coeff2*, *opts-restart-coeff2*) ∈ *opts-rel* → *Id*›
by (*auto simp*: *opts-rel-def intro*!: *frefI*)

fun *opts-rel-target* :: ‹*opts-ref* ⇒ *3 word*› **where**
‹*opts-rel-target* (*res*, *red*, *unbd*, *mini*, *res1*, *res2*, *target*, *fema*, *sema*) = *target*›

lemma *opts-rel-target*:
‹(*opts-rel-target*, *opts-target*) ∈ *opts-rel* → *Id*›
by (*auto simp*: *opts-rel-def intro*!: *frefI*)

fun *opts-rel-fema* :: ‹*opts-ref* ⇒ *64 word*› **where**
‹*opts-rel-fema* (*res*, *red*, *unbd*, *mini*, *res1*, *res2*, *target*, *fema*, *sema*) = *fema*›

lemma *opts-rel-fema*:
‹(*opts-rel-fema*, *opts-fema*) ∈ *opts-rel* → *Id*›
by (*auto simp*: *opts-rel-def intro*!: *frefI*)

fun *opts-rel-sema* :: ‹*opts-ref* ⇒ *64 word*› **where**
‹*opts-rel-sema* (*res*, *red*, *unbd*, *mini*, *res1*, *res2*, *target*, *fema*, *sema*, *units*) = *sema*›
lemma *opts-rel-sema*:
‹(*opts-rel-sema*, *opts-sema*) ∈ *opts-rel* → *Id*›
by (*auto simp*: *opts-rel-def intro*!: *frefI*)

fun *opts-rel-GC-units-lim* :: ‹*opts-ref* ⇒ *64 word*› **where**
‹*opts-rel-GC-units-lim* (*res*, *red*, *unbd*, *mini*, *res1*, *res2*, *target*, *fema*, *sema*, *units*,-) = *units*›

fun *opts-rel-subsumption* :: ‹*opts-ref* ⇒ *bool*› **where**
‹*opts-rel-subsumption* (*res*, *red*, *unbd*, *mini*, *res1*, *res2*, *target*, *fema*, *sema*, *units*,*subsume*) = *subsume*›

lemma *opts-GC-units-lim*:
‹(*opts-rel-GC-units-lim*, *opts-GC-units-lim*) ∈ *opts-rel* → *Id*› **and**
opts-subsumption:
‹(*opts-rel-subsumption*, *opts-subsumption*) ∈ *opts-rel* → *Id*›
by (*auto simp*: *opts-rel-def opts-GC-units-lim-def intro*!: *frefI*)

lemma *opts-rel-alt-defs*:
‹*RETURN o opts-rel-restart* = (λ(*res*, *red*, *unbd*, *mini*, *res1*, *res2*). *RETURN res*)›
‹*RETURN o opts-rel-reduce* = (λ(*res*, *red*, *unbd*, *mini*, *res1*, *res2*). *RETURN red*)›
‹*RETURN o opts-rel-unbounded-mode* = (λ(*res*, *red*, *unbd*, *mini*, *res1*, *res2*). *RETURN unbd*)›
‹*RETURN o opts-rel-miminum-between-restart* = (λ(*res*, *red*, *unbd*, *mini*, *res1*, *res2*). *RETURN mini*)›
‹*RETURN o opts-rel-restart-coeff1* = (λ(*res*, *red*, *unbd*, *mini*, *res1*, *res2*). *RETURN res1*)›
‹*RETURN o opts-rel-restart-coeff2* = (λ(*res*, *red*, *unbd*, *mini*, *res1*, *res2*, -). *RETURN res2*)›
‹*RETURN o opts-rel-target* = (λ(*res*, *red*, *unbd*, *mini*, *res1*, *res2*, *target*, *fema*, *semax*). *RETURN target*)›
‹*RETURN o opts-rel-fema* = (λ(*res*, *red*, *unbd*, *mini*, *res1*, *res2*, *target*, *fema*, *sema*). *RETURN fema*)›
‹*RETURN o opts-rel-sema* = (λ(*res*, *red*, *unbd*, *mini*, *res1*, *res2*, *target*, *fema*, *sema*, *units*). *RETURN sema*)›
‹*RETURN o opts-rel-GC-units-lim* = (λ(*res*, *red*, *unbd*, *mini*, *res1*, *res2*, *target*, *fema*, *sema*, *units*, *subsume*). *RETURN units*)›
‹*RETURN o opts-rel-subsumption* = (λ(*res*, *red*, *unbd*, *mini*, *res1*, *res2*, *target*, *fema*, *sema*, *units*,

subsume). *RETURN subsume*)›
by (*auto intro*!: *ext*)

end
theory *IsaSAT-EMA*
imports *IsaSAT-Literals*
begin

4.8 Moving averages

definition *EMA-FIXPOINT-SIZE* :: ‹*nat*› **where**
‹*EMA-FIXPOINT-SIZE* = *32*›

We use (at least hopefully) the variant of EMA-14 implemented in Cadical, but with fixed-point calculations (*1* is *1* >> *EMA-FIXPOINT-SIZE*).
Remark that the coefficient β already should take care of the fixed-point conversion of the glue. Otherwise, *value* is wrongly updated.

type-synonym *ema* = ‹*64 word* × *64 word* × *64 word* × *64 word* × *64 word*›

definition *ema-bitshifting* **where**
‹*ema-bitshifting* = (*1* << *EMA-FIXPOINT-SIZE*)›

definition *EMA-MULT-SHIFT* :: ‹*nat*› **where**
‹*EMA-MULT-SHIFT* = *8*›

TODO: some precision is lost here in the difference calculation.

definition (**in** −) *ema-update* :: ‹*nat* ⇒ *ema* ⇒ *ema*› **where**
‹*ema-update* = (λ*lbd* (*value*, α, β, *wait*, *period*).
let lbd = (*of-nat lbd*) ∗ *ema-bitshifting in*
let value = *if lbd* > *value*
then value + ((β >> (*EMA-FIXPOINT-SIZE* − *EMA-MULT-SHIFT*)) ∗ ((*lbd* − *value*) >> *EMA-MULT-SHIFT*))
else value − ((β >> (*EMA-FIXPOINT-SIZE* − *EMA-MULT-SHIFT*)) ∗ ((*value* − *lbd*) >> *EMA-MULT-SHIFT*))
in
let wait = *wait* − *1 in*
if β ≤ α ∨ *wait* > *0 then* (*value*, α, β, *wait*, *period*)
else
let wait = *2* ∗ (*period*+*1*)−*1 in*
let period = *wait in*
let β = β >> *1 in*
let β = *if* β < α *then* α *else* β *in*
(*value*, α, β, *wait*, *period*))›

definition (**in** −) *ema-init* :: ‹*64 word* ⇒ *ema*› **where**
‹*ema-init* α = (*0*, α >> (*EMA-FIXPOINT-SIZE* − *32*), *ema-bitshifting*, *1*, *0*)›

fun *ema-reinit* **where**
‹*ema-reinit* (*value*, α, β, *wait*, *period*) = (*value*, α, *ema-bitshifting*, *1*, *0*)›

fun *ema-get-value* :: ‹*ema* ⇒ *64 word*› **where**
‹*ema-get-value* (*v*, -) = *v*›

fun *ema-extract-value* :: ‹*ema* ⇒ *64 word*› **where**
‹*ema-extract-value* (*v*, -) = *v* >> *EMA-FIXPOINT-SIZE*›

We use the default values for Cadical: $(3::'a)$ / $(10::'a)^2$ and $(1::'a)$ / $(10::'a)^5$ in our fixed-point version.

value ‹((*3* :: *64 word*) << *EMA-FIXPOINT-SIZE*) >> (*15*)›
value ‹((*4* :: *64 word*) << *EMA-FIXPOINT-SIZE* >> *7*)›
abbreviation *ema-fast-init* :: *ema* **where**
‹*ema-fast-init* ≡ *ema-init* (*128849010*)› — 5629499534213

abbreviation *ema-slow-init* :: *ema* **where**
‹*ema-slow-init* ≡ *ema-init* (*429450*)› — 2814749767

Small test below. It was useful once to detect an overflow that lead to very bad initialisation behaviour.

value ‹*let* α = *shiftr 128849010* (*EMA-FIXPOINT-SIZE* − *32*);
x = (((*ema-update 10*) ^^ *400*) (*7* ∗ *ema-bitshifting*, α, *ema-bitshifting*, *1*, *0*))
in (*ema-extract-value x*, *ema-get-value x*, *x*)›

end
theory *IsaSAT-Phasing*
imports *IsaSAT-Literals*
begin

4.8.1 Phase saving

type-synonym *phase-saver* = ‹*bool list*›

definition *phase-saving* :: ‹*nat multiset* ⇒ *phase-saver* ⇒ *bool*› **where**
‹*phase-saving* $\mathcal{A}$ φ ⟷ (∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length* φ)›

Save phase as given (e.g. for literals in the trail):

definition *save-phase* :: ‹*nat literal* ⇒ *phase-saver* ⇒ *phase-saver*› **where**
‹*save-phase L* φ = φ[*atm-of L* := *is-pos L*]›

lemma *phase-saving-save-phase*[*simp*]:
‹*phase-saving* $\mathcal{A}$ (*save-phase L* φ) ⟷ *phase-saving* $\mathcal{A}$ φ›
by (*auto simp*: *phase-saving-def save-phase-def*)

Save opposite of the phase (e.g. for literals in the conflict clause):

definition *save-phase-inv* :: ‹*nat literal* ⇒ *phase-saver* ⇒ *phase-saver*› **where**
‹*save-phase-inv L* φ = φ[*atm-of L* := ¬*is-pos L*]›

end
theory *IsaSAT-Rephase*
imports *IsaSAT-Phasing*
begin

Chapter 5

Phase Saving

5.1 Rephasing

type-synonym *phase-save-heur* = ‹*phase-saver* × *64 word* × *phase-saver* × *64 word* × *phase-saver* × *64 word* × *64 word* × *64 word*›

definition *phase-save-heur-rel* :: ‹*nat multiset* ⇒ *phase-save-heur* ⇒ *bool*› **where**
‹*phase-save-heur-rel* $\mathcal{A}$ = (λ(φ, *target-assigned*, *target*, *best-assigned*, *best*,
end-of-phase, *curr-phase*). *phase-saving* $\mathcal{A}$ φ ∧
phase-saving $\mathcal{A}$ *target* ∧ *phase-saving* $\mathcal{A}$ *best* ∧ *length* φ = *length best* ∧
length target = *length best*)›

definition *end-of-rephasing-phase* :: ‹*phase-save-heur* ⇒ *64 word*› **where**
‹*end-of-rephasing-phase* = (λ(φ, *target-assigned*, *target*, *best-assigned*, *best*, *end-of-phase*, *curr-phase*,
length-phase). *end-of-phase*)›

definition *phase-current-rephasing-phase* :: ‹*phase-save-heur* ⇒ *64 word*› **where**
‹*phase-current-rephasing-phase* =
(λ(φ, *target-assigned*, *target*, *best-assigned*, *best*, *end-of-phase*, *curr-phase*, *length-phase*). *curr-phase*)›

We implement the idea in CaDiCaL of rephasing:

- We remember the best model found so far. It is used as base.
- We flip the phase saving heuristics between *True*, *False*, and random.

definition *rephase-init* :: ‹*bool* ⇒ *bool list* ⇒ *bool list nres*› **where**
‹*rephase-init b* φ = *do* {
let n = *length* φ;
nfoldli [*0*..<*n*]
(λ-. *True*)
(λ *a* φ. *do* {
ASSERT(*a* < *length* φ);
RETURN (φ[*a* := *b*])
})
φ
}›

lemma *rephase-init-spec*:
‹*rephase-init b* φ ≤ *SPEC*(λψ. *length* ψ = *length* φ)›
proof −

```
  show ?thesis
  unfolding rephase-init-def Let-def
  apply (rule nfoldli-rule[where I = ‹λ- - ψ. length φ = length ψ›])
  apply (auto dest: in-list-in-setD)
  done
qed


definition copy-phase :: ‹bool list ⇒ bool list ⇒ bool list nres› where
‹copy-phase φ φ' = do {
  ASSERT(length φ = length φ');
  let n = length φ';
  nfoldli [0..<n]
    (λ-. True)
    (λ a φ'. do {
       ASSERT(a < length φ);
       ASSERT(a < length φ');
       RETURN (φ'[a := φ!a])
   })
   φ'
 }›

lemma copy-phase-alt-def:
‹copy-phase φ φ' = do {
  ASSERT(length φ = length φ');
  let n = length φ;
  nfoldli [0..<n]
    (λ-. True)
    (λ a φ'. do {
       ASSERT(a < length φ);
       ASSERT(a < length φ');
       RETURN (φ'[a := φ!a])
   })
   φ'
 }›
  unfolding copy-phase-def
  by (auto simp: ASSERT-same-eq-conv)

lemma copy-phase-spec:
  ‹length φ = length φ' ⟹ copy-phase φ φ' ≤ SPEC(λψ. length ψ = length φ)›
  unfolding copy-phase-def Let-def
  apply (intro ASSERT-leI)
  subgoal by auto
  apply (rule nfoldli-rule[where I = ‹λ- - ψ. length φ = length ψ›])
  apply (auto dest: in-list-in-setD)
  done


definition rephase-random :: ‹64 word ⇒ bool list ⇒ bool list nres› where
‹rephase-random b φ = do {
  let n = length φ;
  (-, φ) ← nfoldli [0..<n]
      (λ-. True)
      (λa (state, φ). do {
        ASSERT(a < length φ);
       let state = state * 6364136223846793005 + 1442695040888963407;
```

```
      RETURN (state, φ[a := (state < 2147483648)])
    })
    (b, φ);
  RETURN φ
 }›
```

lemma *rephase-random-spec*:
```
  ‹rephase-random b φ ≤ SPEC(λψ. length ψ = length φ)›
  unfolding rephase-random-def Let-def
  apply (refine-vcg nfoldli-rule[where I = ‹λ- - (-, ψ). length φ = length ψ›])
  apply (auto dest: in-list-in-setD)
  done
```

definition *rephase-flipped* :: ‹*bool list* ⇒ *bool list nres*› **where**
```
‹rephase-flipped φ = do {
  let n = length φ;
  (φ) ← nfoldli [0..<n]
      (λ-. True)
      (λa  φ. do {
        ASSERT(a < length φ);
       let v = φ ! a;
       RETURN (φ[a := ¬v])
     })
     φ;
  RETURN φ
 }›
```

lemma *rephase-flipped-spec*:
```
  ‹rephase-flipped φ ≤ SPEC(λψ. length ψ = length φ)›
  unfolding rephase-flipped-def Let-def
  apply (refine-vcg nfoldli-rule[where I = ‹λ- - ψ. length φ = length ψ›])
  apply (auto dest: in-list-in-setD)
  done
```

definition *reset-target-phase* :: ‹*phase-save-heur* ⇒ *phase-save-heur nres*› **where**
```
‹reset-target-phase = (λ(φ, target-assigned, target, best-assigned, best, end-of-phase, curr-phase).
      RETURN (φ, 0, target, best-assigned, best, end-of-phase, curr-phase)
   )›
```

definition *reset-best-phase* :: ‹*phase-save-heur* ⇒ *phase-save-heur nres*› **where**
```
‹reset-best-phase = (λ(φ, target-assigned, target, best-assigned, best, end-of-phase, curr-phase).
      RETURN (φ, target-assigned, target, 0, best, end-of-phase, curr-phase)
   )›
```

lemma *reset-target-phase-spec*:
```
  assumes ‹phase-save-heur-rel A φ›
  shows ‹reset-target-phase φ ≤ ⇓Id (SPEC(phase-save-heur-rel A))›
  using assms by (cases φ) (auto simp: reset-target-phase-def phase-save-heur-rel-def)
```

lemma *reset-best-phase-spec*:

assumes ‹*phase-save-heur-rel* $\mathcal{A}$ φ›
shows ‹*reset-best-phase* $\varphi \leq \Downarrow Id$ (*SPEC*(*phase-save-heur-rel* $\mathcal{A}$))›
using *assms* **by** (*cases* φ) (*auto simp*: *reset-best-phase-def phase-save-heur-rel-def*)

definition *current-phase-letter* :: ‹*64 word* ⇒ *64 word*› **where**
‹*current-phase-letter curr-phase* =
(*if curr-phase* = *0* ∨ *curr-phase* = *2* ∨ *curr-phase* = *4* ∨ *curr-phase* = *6*
then 66
else if curr-phase = *1*
then 73
else if curr-phase = *3*
then 35
else if curr-phase = *5*
then 79
else 70)
›

The phases are: BOBIBF independantly of the mode of the solver unlike CaDiCaL where this is independent and kissat where no flipping is used in unsat mode. We schedule in log interval. In the last phase, we increase the length of the phase in Θ (*log 10 n* ∗ *log 10 n*).

definition *phase-rephase* :: ‹*64 word* ⇒ *64 word* ⇒ *64 word* ⇒ *phase-save-heur* ⇒ *phase-save-heur nres*› **where**
‹*phase-rephase* = (λ*end-of-phase lrephase* (*b*::*64 word*) (φ, *target-assigned*, *target*, *best-assigned*, *best*, -, *curr-phase*,
length-phase).
do {
let rephaselength = (*500* :: *64 word*);
let target-assigned = (*0*::*64 word*);
target ← *copy-phase* φ *target*;
if curr-phase = *0* ∨ *curr-phase* = *2* ∨ *curr-phase* = *4* ∨ *curr-phase* = *6* — reset the best best phase
then do {
φ ← *copy-phase best* φ;
RETURN (φ, *target-assigned*, *target*, *0*, *best*, *length-phase*∗*rephaselength*+*end-of-phase*, *curr-phase* + *1*, *length-phase*)
}
else if curr-phase = *1*
then do {
φ ← *rephase-init True* φ;
RETURN (φ, *target-assigned*, *target*, *best-assigned*, *best*, *length-phase*∗*rephaselength*+*end-of-phase*, *curr-phase* + *1*, *length-phase*)
}
else if curr-phase = *3*
then do {
φ ← *rephase-random end-of-phase* φ;
RETURN (φ, *target-assigned*, *target*, *best-assigned*, *best*, *length-phase*∗*rephaselength*+*end-of-phase*, *curr-phase* + *1*, *length-phase*)
}
else if curr-phase = *5*
then do {
φ ← *rephase-init False* φ;
RETURN (φ, *target-assigned*, *target*, *best-assigned*, *best*, *length-phase*∗*rephaselength*+*end-of-phase*, *curr-phase* + *1*, *length-phase*)
}
else do {

```
        φ ← rephase-flipped φ;
        let loglength = (if lrephase+10 > 0 then of-nat (word-log2 (lrephase+10)) else 1);
        let new-length = (lrephase+1)*loglength*loglength;
       RETURN (φ, target-assigned, target, best-assigned, best, new-length*rephaselength+end-of-phase,
0,
           new-length)
     }
   })›
```

lemma *phase-rephase-spec*:
assumes ‹*phase-save-heur-rel* $\mathcal{A}$ φ›
shows ‹*phase-rephase end-of-phase lrephase b* $\varphi \leq \Downarrow$*Id* (*SPEC*(*phase-save-heur-rel* $\mathcal{A}$))›
proof −
obtain φ' *target-assigned target best-assigned best end-of-phase curr-phase* **where**
φ: ‹$\varphi = (\varphi'$, *target-assigned*, *target*, *best-assigned*, *best*, *end-of-phase*, *curr-phase*)›
by (*cases* φ) *auto*
then have [*simp*]: ‹*length* φ' = *length best*›
using *assms* **by** (*auto simp*: *phase-save-heur-rel-def*)
show *?thesis*
using *assms*
unfolding *phase-rephase-def*
by (*refine-vcg lhs-step-If rephase-init-spec*[*THEN order-trans*]
copy-phase-spec[*THEN order-trans*] *rephase-random-spec*[*THEN order-trans*]
rephase-flipped-spec[*THEN order-trans*])
(*auto simp*: *phase-save-heur-rel-def phase-saving-def*)
qed

definition *phase-save-phase* :: ‹*64 word* ⇒ *phase-save-heur* ⇒ *phase-save-heur nres*› **where**
‹*phase-save-phase* = (λ*n* (φ, *target-assigned*, *target*, *best-assigned*, *best*, *end-of-phase*, *curr-phase*). *do* {
target ← (*if n* > *target-assigned*
then copy-phase φ *target else RETURN target*);
target-assigned ← (*if n* > *target-assigned*
then RETURN n else RETURN target-assigned);
best ← (*if n* > *best-assigned*
then copy-phase φ *best else RETURN best*);
best-assigned ← (*if n* > *best-assigned*
then RETURN n else RETURN best-assigned);
RETURN (φ, *target-assigned*, *target*, *best-assigned*, *best*, *end-of-phase*, *curr-phase*)
})›

lemma *phase-save-phase-spec*:
assumes ‹*phase-save-heur-rel* $\mathcal{A}$ φ›
shows ‹*phase-save-phase n* $\varphi \leq \Downarrow$*Id* (*SPEC*(*phase-save-heur-rel* $\mathcal{A}$))›
proof −
obtain φ' *target-assigned target best-assigned best end-of-phase curr-phase* **where**
φ: ‹$\varphi = (\varphi'$, *target-assigned*, *target*, *best-assigned*, *best*, *end-of-phase*, *curr-phase*)›
by (*cases* φ) *auto*
then have [*simp*]: ‹*length* φ' = *length best*› ‹*length target* = *length best*›
using *assms* **by** (*auto simp*: *phase-save-heur-rel-def*)
have *1*: ‹$\Downarrow$*Id* (*SPEC*(*phase-save-heur-rel* $\mathcal{A}$)) ≥
$\Downarrow$*Id*((λ(φ, *target-assigned*, *target*, *best-assigned*, *best*, *end-of-phase*, *curr-phase*). *do* {
target ← (*if n* > *target-assigned*
then SPEC (λφ'. *length* φ = *length* φ') *else RETURN target*);
target-assigned ← (*if n* > *target-assigned*

```
        then RETURN n else RETURN target-assigned);
      best ← (if n > best-assigned
        then SPEC (λφ'. length φ = length φ') else RETURN best);
      best-assigned ← (if n > best-assigned
        then RETURN n else RETURN best-assigned);
      RETURN (φ', target-assigned, target, best-assigned, best, end-of-phase, curr-phase)
    }) φ)›
  using assms
 by (auto simp: phase-save-heur-rel-def phase-saving-def RES-RETURN-RES φ RES-RES-RETURN-RES)

 show ?thesis
  unfolding phase-save-phase-def φ
  apply (simp only: prod.case)
  apply (rule order-trans)
  defer
  apply (rule 1)
  apply (simp only: prod.case φ)
  apply (refine-vcg if-mono rephase-init-spec copy-phase-spec rephase-random-spec)
  apply (auto simp: phase-rephase-def)
  done
qed

definition get-next-phase-pre :: ‹bool ⇒ nat ⇒ phase-save-heur ⇒ bool› where
 ‹get-next-phase-pre = (λuse-target-phasing L (φ, target-assigned, target, best-assigned, best, end-of-phase,
curr-phase).
   L < length φ ∧ L < length target)›

definition get-next-phase-stats :: ‹bool ⇒ nat ⇒ phase-save-heur ⇒ bool nres› where
 ‹get-next-phase-stats = (λuse-target-phasing L (φ, target-assigned, target, best-assigned, best, end-of-phase,
curr-phase).
  if ¬use-target-phasing then do {
   ASSERT(L < length φ);
   RETURN(φ ! L)
  } else do {
   ASSERT(L < length target);
   RETURN(target ! L)
  })›

end
theory IsaSAT-Reluctant
 imports More-Sepref.WB-More-Refinement
   Isabelle-LLVM.Bits-Natural
begin
```

In this file, we define the Luby sequence, based on the implementation from CaDiCaL.

```
datatype reluctant =
  Reluctant
    (reluctant-limited: bool)
    (reluctant-trigger: bool)
    (reluctant-u: ‹64 word›)
    (reluctant-v: ‹64 word›)
    (reluctant-period: ‹64 word›)
    (reluctant-wait: ‹64 word›)
    (reluctant-limit: ‹64 word›)

definition reluctant-set-trigger :: ‹bool ⇒ reluctant ⇒ reluctant› where
```

```
‹reluctant-set-trigger trigger r =
   (let
  limited = reluctant-limited r;
  u = reluctant-u r;
  v = reluctant-v r;
  period = reluctant-period r;
  wait = reluctant-wait r;
  limit = reluctant-limit r in Reluctant limited trigger u v period wait limit)›

definition reluctant-disable :: ‹reluctant ⇒ reluctant› where
 ‹reluctant-disable r =
 (let
  limited = reluctant-limited r;
  trigger = reluctant-trigger r;
  u = reluctant-u r;
  v = reluctant-v r;
  period = reluctant-period r;
  wait = reluctant-wait r;
 limit = reluctant-limit r in
 (Reluctant limited trigger u v 0 wait limit))›

definition reluctant-untrigger :: ‹reluctant ⇒ reluctant› where
 ‹reluctant-untrigger r = (reluctant-set-trigger False r)›

definition reluctant-triggered :: ‹reluctant ⇒ reluctant × bool› where
 ‹reluctant-triggered r = (reluctant-set-trigger False r, reluctant-trigger r)›

definition reluctant-triggered2 :: ‹reluctant ⇒ bool› where
 ‹reluctant-triggered2 r = (reluctant-trigger r)›

definition reluctant-tick :: ‹reluctant ⇒ reluctant› where
 ‹reluctant-tick  r =
 (let
  limited = reluctant-limited r;
  trigger = reluctant-trigger r;
  u = reluctant-u r;
  v = reluctant-v r;
  period = reluctant-period r;
  wait = reluctant-wait r;
  limit = reluctant-limit r in
 (if period = 0 ∨ trigger then Reluctant limited trigger u v period (wait) limit
  else if wait > 1 then Reluctant limited trigger u v period (wait − 1) limit
  else let (u, v) = (if u AND (0−u) = v then (u+1, 1) else (u, 2 ∗ v));
     (u, v) = (if limited ∧ wait > limit then (1,1) else (u, v));
     wait = v ∗ period in
  Reluctant limited True u v period wait limit))›

definition reluctant-enable :: ‹64 word ⇒ 64 word ⇒ reluctant› where
 ‹reluctant-enable period limit =
 (let limited = limit ≠ 0;
      period = (if limited ∧ period > limit then limit else period)
   in
  Reluctant limited False 1 1 period period limit)›

definition reluctant-init :: ‹reluctant› where
 ‹reluctant-init = reluctant-enable (1<< 10) (1<<20)›
```

value
‹*let p0* = (*reluctant-tick* ^^ *1024*) (*reluctant-enable* (*1*<< *10*) (*1*<<*20*));
p1 = *reluctant-untrigger p0*;
p2 = (*reluctant-tick* ^^ *1024*) *p1*;
p3 = *reluctant-untrigger p2*;
p4 = (*reluctant-tick* ^^ *2048*) *p3*;
p5 = *reluctant-untrigger p3*;
p6 = (*reluctant-tick* ^^ *2048*) *p5 in*
(*p0*, *p1*, *p2*, *p3*, *p4*, *p5*, *p6*, *reluctant-triggered2 p0* = *True*, *reluctant-triggered2 p2* = *False*)›
end
theory *Tuple16*
imports *More-Sepref.WB-More-Refinement IsaSAT-Literals*
begin

datatype ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*, '*p*) *tuple16* = *Tuple16*
(*Tuple16-get-a*: '*a*)
(*Tuple16-get-b*: '*b*)
(*Tuple16-get-c*: '*c*)
(*Tuple16-get-d*: '*d*)
(*Tuple16-get-e*: '*e*)
(*Tuple16-get-f*: '*f*)
(*Tuple16-get-g*: '*g*)
(*Tuple16-get-h*: '*h*)
(*Tuple16-get-i*: '*i*)
(*Tuple16-get-j*: '*j*)
(*Tuple16-get-k*: '*k*)
(*Tuple16-get-l*: '*l*)
(*Tuple16-get-m*: '*m*)
(*Tuple16-get-n*: '*n*)
(*Tuple16-get-o*: '*o*)
(*Tuple16-get-p*: '*p*)

Accessors context
begin
qualified fun *set-a* :: ‹'*a* ⇒ ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*, '*p*) *tuple16* ⇒ -› **where**
‹*set-a M* (*Tuple16* - *N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena*) = (*Tuple16 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena*)›

fun *set-b* :: ‹'*b* ⇒('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*, '*p*) *tuple16* ⇒ -› **where**
‹*set-b N* (*Tuple16 M* - *D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena*) = (*Tuple16 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena*)›

fun *set-c* :: ‹'*c* ⇒('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*, '*p*) *tuple16* ⇒ -› **where**
‹*set-c D* (*Tuple16 M N* - *i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena*) = (*Tuple16 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena*)›

fun *set-d* :: ‹'*d* ⇒('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*, '*p*) *tuple16* ⇒ -› **where**
‹*set-d i* (*Tuple16 M N D* - *W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena*) = (*Tuple16 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena*)›

fun *set-e* :: ‹'*e* ⇒('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*, '*p*) *tuple16* ⇒ -› **where**
‹*set-e W* (*Tuple16 M N D i* - *ivmtf icount ccach lbd outl heur stats aivdom clss opts arena*) = (*Tuple16 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena*)›

fun *set-f* :: ‹'f ⇒('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j, 'k, 'l, 'm, 'n, 'o, 'p) *tuple16* ⇒ -› **where**
‹*set-f ivmtf* (*Tuple16 M N D i W* - *icount ccach lbd outl heur stats aivdom clss opts arena*) = (*Tuple16 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena*)›

fun *set-g* :: ‹'g ⇒('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j, 'k, 'l, 'm, 'n, 'o, 'p) *tuple16* ⇒ -› **where**
‹*set-g icount* (*Tuple16 M N D i W ivmtf* - *ccach lbd outl heur stats aivdom clss opts arena*) = (*Tuple16 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena*)›

fun *set-h* :: ‹'h ⇒('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j, 'k, 'l, 'm, 'n, 'o, 'p) *tuple16* ⇒ -› **where**
‹*set-h ccach* (*Tuple16 M N D i W ivmtf icount* - *lbd outl heur stats aivdom clss opts arena*) = (*Tuple16 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena*)›

fun *set-i* :: ‹'i ⇒('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j, 'k, 'l, 'm, 'n, 'o, 'p) *tuple16* ⇒ -› **where**
‹*set-i lbd* (*Tuple16 M N D i W ivmtf icount ccach* - *outl heur stats aivdom clss opts arena*) = (*Tuple16 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena*)›

fun *set-j* :: ‹'j ⇒('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j, 'k, 'l, 'm, 'n, 'o, 'p) *tuple16* ⇒ -› **where**
‹*set-j outl* (*Tuple16 M N D i W ivmtf icount ccach lbd* - *heur stats aivdom clss opts arena*) = (*Tuple16 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena*)›

fun *set-k* :: ‹'k ⇒('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j, 'k, 'l, 'm, 'n, 'o, 'p) *tuple16* ⇒ -› **where**
‹*set-k stats* (*Tuple16 M N D i W ivmtf icount ccach lbd outl* - *heur aivdom clss opts arena*) = (*Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena*)›

fun *set-l* :: ‹'l ⇒('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j, 'k, 'l, 'm, 'n, 'o, 'p) *tuple16* ⇒ -› **where**
‹*set-l heur* (*Tuple16 M N D i W ivmtf icount ccach lbd outl stats* - *aivdom clss opts arena*) = (*Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena*)›

fun *set-m* :: ‹'m ⇒('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j, 'k, 'l, 'm, 'n, 'o, 'p) *tuple16* ⇒ -› **where**
‹*set-m aivdom* (*Tuple16 M N D i W ivmtf icount ccach lbd outl heur stats* - *clss opts arena*) = (*Tuple16 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena*)›

fun *set-n* :: ‹'n ⇒('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j, 'k, 'l, 'm, 'n, 'o, 'p) *tuple16* ⇒ -› **where**
‹*set-n clss* (*Tuple16 M N D i W ivmtf icount ccach lbd outl heur stats aivdom* - *opts arena*) = (*Tuple16 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena*)›

fun *set-o* :: ‹'o ⇒('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j, 'k, 'l, 'm, 'n, 'o, 'p) *tuple16* ⇒ -› **where**
‹*set-o opts* (*Tuple16 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss* - *arena*) = (*Tuple16 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena*)›

fun *set-p* :: ‹'p ⇒('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j, 'k, 'l, 'm, 'n, 'o, 'p) *tuple16* ⇒ -› **where**
‹*set-p arena* (*Tuple16 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts* -) = (*Tuple16 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena*)›

end

named-theorems *tuple16-state-simp*
lemma [*tuple16-state-simp*]:
‹*Tuple16-get-a* (*Tuple16.set-a M S*) = *M*›
‹*Tuple16-get-b* (*Tuple16.set-a M S*) = *Tuple16-get-b S*›
‹*Tuple16-get-c*(*Tuple16.set-a M S*) = *Tuple16-get-c S*›
‹*Tuple16-get-d* (*Tuple16.set-a M S*) = *Tuple16-get-d S*›
‹*Tuple16-get-e* (*Tuple16.set-a M S*) = *Tuple16-get-e S*›
‹*Tuple16-get-f* (*Tuple16.set-a M S*) = *Tuple16-get-f S*›
‹*Tuple16-get-g* (*Tuple16.set-a M S*) = *Tuple16-get-g S*›

‹*Tuple16-get-h* (*Tuple16.set-a M S*) = *Tuple16-get-h S*›
‹*Tuple16-get-i* (*Tuple16.set-a M S*) = *Tuple16-get-i S*›
‹*Tuple16-get-j* (*Tuple16.set-a M S*) = *Tuple16-get-j S*›
‹*Tuple16-get-k* (*Tuple16.set-a M S*) = *Tuple16-get-k S*›
‹*Tuple16-get-l* (*Tuple16.set-a M S*) = *Tuple16-get-l S*›
‹*Tuple16-get-m* (*Tuple16.set-a M S*) = *Tuple16-get-m S*›
‹*Tuple16-get-n* (*Tuple16.set-a M S*) = *Tuple16-get-n S*›
‹*Tuple16-get-o* (*Tuple16.set-a M S*) = *Tuple16-get-o S*›
‹*Tuple16-get-p* (*Tuple16.set-a M S*) = *Tuple16-get-p S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple16-state-simp*]:
‹*Tuple16-get-a* (*Tuple16.set-b N S*) = *Tuple16-get-a S*›
‹*Tuple16-get-b* (*Tuple16.set-b N S*) = *N*›
‹*Tuple16-get-c*(*Tuple16.set-b N S*) = *Tuple16-get-c S*›
‹*Tuple16-get-d* (*Tuple16.set-b N S*) = *Tuple16-get-d S*›
‹*Tuple16-get-e* (*Tuple16.set-b N S*) = *Tuple16-get-e S*›
‹*Tuple16-get-f* (*Tuple16.set-b N S*) = *Tuple16-get-f S*›
‹*Tuple16-get-g* (*Tuple16.set-b N S*) = *Tuple16-get-g S*›
‹*Tuple16-get-h* (*Tuple16.set-b N S*) = *Tuple16-get-h S*›
‹*Tuple16-get-i* (*Tuple16.set-b N S*) = *Tuple16-get-i S*›
‹*Tuple16-get-j* (*Tuple16.set-b N S*) = *Tuple16-get-j S*›
‹*Tuple16-get-k* (*Tuple16.set-b N S*) = *Tuple16-get-k S*›
‹*Tuple16-get-l* (*Tuple16.set-b N S*) = *Tuple16-get-l S*›
‹*Tuple16-get-m* (*Tuple16.set-b N S*) = *Tuple16-get-m S*›
‹*Tuple16-get-n* (*Tuple16.set-b N S*) = *Tuple16-get-n S*›
‹*Tuple16-get-o* (*Tuple16.set-b N S*) = *Tuple16-get-o S*›
‹*Tuple16-get-p* (*Tuple16.set-b N S*) = *Tuple16-get-p S*›

‹*Tuple16-get-a* (*Tuple16.set-c D S*) = *Tuple16-get-a S*›
‹*Tuple16-get-b* (*Tuple16.set-c D S*) = *Tuple16-get-b S*›
‹*Tuple16-get-c*(*Tuple16.set-c D S*) = *D*›
‹*Tuple16-get-d* (*Tuple16.set-c D S*) = *Tuple16-get-d S*›
‹*Tuple16-get-e* (*Tuple16.set-c D S*) = *Tuple16-get-e S*›
‹*Tuple16-get-f* (*Tuple16.set-c D S*) = *Tuple16-get-f S*›
‹*Tuple16-get-g* (*Tuple16.set-c D S*) = *Tuple16-get-g S*›
‹*Tuple16-get-h* (*Tuple16.set-c D S*) = *Tuple16-get-h S*›
‹*Tuple16-get-i* (*Tuple16.set-c D S*) = *Tuple16-get-i S*›
‹*Tuple16-get-j* (*Tuple16.set-c D S*) = *Tuple16-get-j S*›
‹*Tuple16-get-k* (*Tuple16.set-c D S*) = *Tuple16-get-k S*›
‹*Tuple16-get-l* (*Tuple16.set-c D S*) = *Tuple16-get-l S*›
‹*Tuple16-get-m* (*Tuple16.set-c D S*) = *Tuple16-get-m S*›
‹*Tuple16-get-n* (*Tuple16.set-c D S*) = *Tuple16-get-n S*›
‹*Tuple16-get-o* (*Tuple16.set-c D S*) = *Tuple16-get-o S*›
‹*Tuple16-get-p* (*Tuple16.set-c D S*) = *Tuple16-get-p S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple16-state-simp*]:
‹*Tuple16-get-a* (*Tuple16.set-d j S*) = *Tuple16-get-a S*›
‹*Tuple16-get-b* (*Tuple16.set-d j S*) = *Tuple16-get-b S*›
‹*Tuple16-get-c*(*Tuple16.set-d j S*) = *Tuple16-get-c S*›
‹*Tuple16-get-d* (*Tuple16.set-d j S*) = *j*›
‹*Tuple16-get-e* (*Tuple16.set-d j S*) = *Tuple16-get-e S*›
‹*Tuple16-get-f* (*Tuple16.set-d j S*) = *Tuple16-get-f S*›
‹*Tuple16-get-g* (*Tuple16.set-d j S*) = *Tuple16-get-g S*›
‹*Tuple16-get-h* (*Tuple16.set-d j S*) = *Tuple16-get-h S*›

‹*Tuple16-get-i* (*Tuple16.set-d j S*) = *Tuple16-get-i S*›
‹*Tuple16-get-j* (*Tuple16.set-d j S*) = *Tuple16-get-j S*›
‹*Tuple16-get-k* (*Tuple16.set-d j S*) = *Tuple16-get-k S*›
‹*Tuple16-get-l* (*Tuple16.set-d j S*) = *Tuple16-get-l S*›
‹*Tuple16-get-m* (*Tuple16.set-d j S*) = *Tuple16-get-m S*›
‹*Tuple16-get-n* (*Tuple16.set-d j S*) = *Tuple16-get-n S*›
‹*Tuple16-get-o* (*Tuple16.set-d j S*) = *Tuple16-get-o S*›
‹*Tuple16-get-p* (*Tuple16.set-d j S*) = *Tuple16-get-p S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple16-state-simp*]:
‹*Tuple16-get-a* (*Tuple16.set-e W S*) = *Tuple16-get-a S*›
‹*Tuple16-get-b* (*Tuple16.set-e W S*) = *Tuple16-get-b S*›
‹*Tuple16-get-c*(*Tuple16.set-e W S*) = *Tuple16-get-c S*›
‹*Tuple16-get-d* (*Tuple16.set-e W S*) = *Tuple16-get-d S*›
‹*Tuple16-get-e* (*Tuple16.set-e W S*) = *W*›
‹*Tuple16-get-f* (*Tuple16.set-e W S*) = *Tuple16-get-f S*›
‹*Tuple16-get-g* (*Tuple16.set-e W S*) = *Tuple16-get-g S*›
‹*Tuple16-get-h* (*Tuple16.set-e W S*) = *Tuple16-get-h S*›
‹*Tuple16-get-i* (*Tuple16.set-e W S*) = *Tuple16-get-i S*›
‹*Tuple16-get-j* (*Tuple16.set-e W S*) = *Tuple16-get-j S*›
‹*Tuple16-get-k* (*Tuple16.set-e W S*) = *Tuple16-get-k S*›
‹*Tuple16-get-l* (*Tuple16.set-e W S*) = *Tuple16-get-l S*›
‹*Tuple16-get-m* (*Tuple16.set-e W S*) = *Tuple16-get-m S*›
‹*Tuple16-get-n* (*Tuple16.set-e W S*) = *Tuple16-get-n S*›
‹*Tuple16-get-o* (*Tuple16.set-e W S*) = *Tuple16-get-o S*›
‹*Tuple16-get-p* (*Tuple16.set-e W S*) = *Tuple16-get-p S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple16-state-simp*]:
‹*Tuple16-get-a* (*Tuple16.set-f vmtf' S*) = *Tuple16-get-a S*›
‹*Tuple16-get-b* (*Tuple16.set-f vmtf' S*) = *Tuple16-get-b S*›
‹*Tuple16-get-c*(*Tuple16.set-f vmtf' S*) = *Tuple16-get-c S*›
‹*Tuple16-get-d* (*Tuple16.set-f vmtf' S*) = *Tuple16-get-d S*›
‹*Tuple16-get-e* (*Tuple16.set-f vmtf' S*) = *Tuple16-get-e S*›
‹*Tuple16-get-f* (*Tuple16.set-f vmtf' S*) = *vmtf'*›
‹*Tuple16-get-g* (*Tuple16.set-f vmtf' S*) = *Tuple16-get-g S*›
‹*Tuple16-get-h* (*Tuple16.set-f vmtf' S*) = *Tuple16-get-h S*›
‹*Tuple16-get-i* (*Tuple16.set-f vmtf' S*) = *Tuple16-get-i S*›
‹*Tuple16-get-j* (*Tuple16.set-f vmtf' S*) = *Tuple16-get-j S*›
‹*Tuple16-get-k* (*Tuple16.set-f vmtf' S*) = *Tuple16-get-k S*›
‹*Tuple16-get-l* (*Tuple16.set-f vmtf' S*) = *Tuple16-get-l S*›
‹*Tuple16-get-m* (*Tuple16.set-f vmtf' S*) = *Tuple16-get-m S*›
‹*Tuple16-get-n* (*Tuple16.set-f vmtf' S*) = *Tuple16-get-n S*›
‹*Tuple16-get-o* (*Tuple16.set-f vmtf' S*) = *Tuple16-get-o S*›
‹*Tuple16-get-p* (*Tuple16.set-f vmtf' S*) = *Tuple16-get-p S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple16-state-simp*]:
‹*Tuple16-get-a* (*Tuple16.set-g count' S*) = *Tuple16-get-a S*›
‹*Tuple16-get-b* (*Tuple16.set-g count' S*) = *Tuple16-get-b S*›
‹*Tuple16-get-c*(*Tuple16.set-g count' S*) = *Tuple16-get-c S*›
‹*Tuple16-get-d* (*Tuple16.set-g count' S*) = *Tuple16-get-d S*›
‹*Tuple16-get-e* (*Tuple16.set-g count' S*) = *Tuple16-get-e S*›
‹*Tuple16-get-f* (*Tuple16.set-g count' S*) = *Tuple16-get-f S*›
‹*Tuple16-get-g* (*Tuple16.set-g count' S*) = *count'*›

‹*Tuple16-get-h* (*Tuple16.set-g count′ S*) = *Tuple16-get-h S*›
‹*Tuple16-get-i* (*Tuple16.set-g count′ S*) = *Tuple16-get-i S*›
‹*Tuple16-get-j* (*Tuple16.set-g count′ S*) = *Tuple16-get-j S*›
‹*Tuple16-get-k* (*Tuple16.set-g count′ S*) = *Tuple16-get-k S*›
‹*Tuple16-get-l* (*Tuple16.set-g count′ S*) = *Tuple16-get-l S*›
‹*Tuple16-get-m* (*Tuple16.set-g count′ S*) = *Tuple16-get-m S*›
‹*Tuple16-get-n* (*Tuple16.set-g count′ S*) = *Tuple16-get-n S*›
‹*Tuple16-get-o* (*Tuple16.set-g count′ S*) = *Tuple16-get-o S*›
‹*Tuple16-get-p* (*Tuple16.set-g count′ S*) = *Tuple16-get-p S*›
by (*solves* ‹*cases S*; *auto simp*: ›)+

lemma [*tuple16-state-simp*]:
‹*Tuple16-get-a* (*Tuple16.set-h ccach S*) = *Tuple16-get-a S*›
‹*Tuple16-get-b* (*Tuple16.set-h ccach S*) = *Tuple16-get-b S*›
‹*Tuple16-get-c*(*Tuple16.set-h ccach S*) = *Tuple16-get-c S*›
‹*Tuple16-get-d* (*Tuple16.set-h ccach S*) = *Tuple16-get-d S*›
‹*Tuple16-get-e* (*Tuple16.set-h ccach S*) = *Tuple16-get-e S*›
‹*Tuple16-get-f* (*Tuple16.set-h ccach S*) = *Tuple16-get-f S*›
‹*Tuple16-get-g* (*Tuple16.set-h ccach S*) = *Tuple16-get-g S*›
‹*Tuple16-get-h* (*Tuple16.set-h ccach S*) = *ccach*›
‹*Tuple16-get-i* (*Tuple16.set-h ccach S*) = *Tuple16-get-i S*›
‹*Tuple16-get-j* (*Tuple16.set-h ccach S*) = *Tuple16-get-j S*›
‹*Tuple16-get-k* (*Tuple16.set-h ccach S*) = *Tuple16-get-k S*›
‹*Tuple16-get-l* (*Tuple16.set-h ccach S*) = *Tuple16-get-l S*›
‹*Tuple16-get-m* (*Tuple16.set-h ccach S*) = *Tuple16-get-m S*›
‹*Tuple16-get-n* (*Tuple16.set-h ccach S*) = *Tuple16-get-n S*›
‹*Tuple16-get-o* (*Tuple16.set-h ccach S*) = *Tuple16-get-o S*›
‹*Tuple16-get-p* (*Tuple16.set-h ccach S*) = *Tuple16-get-p S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple16-state-simp*]:
‹*Tuple16-get-a* (*Tuple16.set-i lbd S*) = *Tuple16-get-a S*›
‹*Tuple16-get-b* (*Tuple16.set-i lbd S*) = *Tuple16-get-b S*›
‹*Tuple16-get-c*(*Tuple16.set-i lbd S*) = *Tuple16-get-c S*›
‹*Tuple16-get-d* (*Tuple16.set-i lbd S*) = *Tuple16-get-d S*›
‹*Tuple16-get-e* (*Tuple16.set-i lbd S*) = *Tuple16-get-e S*›
‹*Tuple16-get-f* (*Tuple16.set-i lbd S*) = *Tuple16-get-f S*›
‹*Tuple16-get-g* (*Tuple16.set-i lbd S*) = *Tuple16-get-g S*›
‹*Tuple16-get-h* (*Tuple16.set-i lbd S*) = *Tuple16-get-h S*›
‹*Tuple16-get-i* (*Tuple16.set-i lbd S*) = *lbd*›
‹*Tuple16-get-j* (*Tuple16.set-i lbd S*) = *Tuple16-get-j S*›
‹*Tuple16-get-k* (*Tuple16.set-i lbd S*) = *Tuple16-get-k S*›
‹*Tuple16-get-l* (*Tuple16.set-i lbd S*) = *Tuple16-get-l S*›
‹*Tuple16-get-m* (*Tuple16.set-i lbd S*) = *Tuple16-get-m S*›
‹*Tuple16-get-n* (*Tuple16.set-i lbd S*) = *Tuple16-get-n S*›
‹*Tuple16-get-o* (*Tuple16.set-i lbd S*) = *Tuple16-get-o S*›
‹*Tuple16-get-p* (*Tuple16.set-i lbd S*) = *Tuple16-get-p S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple16-state-simp*]:
‹*Tuple16-get-a* (*Tuple16.set-j outl S*) = *Tuple16-get-a S*›
‹*Tuple16-get-b* (*Tuple16.set-j outl S*) = *Tuple16-get-b S*›
‹*Tuple16-get-c*(*Tuple16.set-j outl S*) = *Tuple16-get-c S*›
‹*Tuple16-get-d* (*Tuple16.set-j outl S*) = *Tuple16-get-d S*›
‹*Tuple16-get-e* (*Tuple16.set-j outl S*) = *Tuple16-get-e S*›
‹*Tuple16-get-f* (*Tuple16.set-j outl S*) = *Tuple16-get-f S*›

‹*Tuple16-get-g* (*Tuple16.set-j outl S*) = *Tuple16-get-g S*›
‹*Tuple16-get-h* (*Tuple16.set-j outl S*) = *Tuple16-get-h S*›
‹*Tuple16-get-i* (*Tuple16.set-j outl S*) = *Tuple16-get-i S*›
‹*Tuple16-get-j* (*Tuple16.set-j outl S*) = *outl*›
‹*Tuple16-get-k* (*Tuple16.set-j outl S*) = *Tuple16-get-k S*›
‹*Tuple16-get-l* (*Tuple16.set-j outl S*) = *Tuple16-get-l S*›
‹*Tuple16-get-m* (*Tuple16.set-j outl S*) = *Tuple16-get-m S*›
‹*Tuple16-get-n* (*Tuple16.set-j outl S*) = *Tuple16-get-n S*›
‹*Tuple16-get-o* (*Tuple16.set-j outl S*) = *Tuple16-get-o S*›
‹*Tuple16-get-p* (*Tuple16.set-j outl S*) = *Tuple16-get-p S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple16-state-simp*]:
‹*Tuple16-get-a* (*Tuple16.set-k stats S*) = *Tuple16-get-a S*›
‹*Tuple16-get-b* (*Tuple16.set-k stats S*) = *Tuple16-get-b S*›
‹*Tuple16-get-c*(*Tuple16.set-k stats S*) = *Tuple16-get-c S*›
‹*Tuple16-get-d* (*Tuple16.set-k stats S*) = *Tuple16-get-d S*›
‹*Tuple16-get-e* (*Tuple16.set-k stats S*) = *Tuple16-get-e S*›
‹*Tuple16-get-f* (*Tuple16.set-k stats S*) = *Tuple16-get-f S*›
‹*Tuple16-get-g* (*Tuple16.set-k stats S*) = *Tuple16-get-g S*›
‹*Tuple16-get-h* (*Tuple16.set-k stats S*) = *Tuple16-get-h S*›
‹*Tuple16-get-i* (*Tuple16.set-k stats S*) = *Tuple16-get-i S*›
‹*Tuple16-get-j* (*Tuple16.set-k stats S*) = *Tuple16-get-j S*›
‹*Tuple16-get-k* (*Tuple16.set-k stats S*) = *stats*›
‹*Tuple16-get-l* (*Tuple16.set-k stats S*) = *Tuple16-get-l S*›
‹*Tuple16-get-m* (*Tuple16.set-k stats S*) = *Tuple16-get-m S*›
‹*Tuple16-get-n* (*Tuple16.set-k stats S*) = *Tuple16-get-n S*›
‹*Tuple16-get-o* (*Tuple16.set-k stats S*) = *Tuple16-get-o S*›
‹*Tuple16-get-p* (*Tuple16.set-k stats S*) = *Tuple16-get-p S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple16-state-simp*]:
‹*Tuple16-get-a* (*Tuple16.set-l heur S*) = *Tuple16-get-a S*›
‹*Tuple16-get-b* (*Tuple16.set-l heur S*) = *Tuple16-get-b S*›
‹*Tuple16-get-c*(*Tuple16.set-l heur S*) = *Tuple16-get-c S*›
‹*Tuple16-get-d* (*Tuple16.set-l heur S*) = *Tuple16-get-d S*›
‹*Tuple16-get-e* (*Tuple16.set-l heur S*) = *Tuple16-get-e S*›
‹*Tuple16-get-f* (*Tuple16.set-l heur S*) = *Tuple16-get-f S*›
‹*Tuple16-get-g* (*Tuple16.set-l heur S*) = *Tuple16-get-g S*›
‹*Tuple16-get-h* (*Tuple16.set-l heur S*) = *Tuple16-get-h S*›
‹*Tuple16-get-i* (*Tuple16.set-l heur S*) = *Tuple16-get-i S*›
‹*Tuple16-get-j* (*Tuple16.set-l heur S*) = *Tuple16-get-j S*›
‹*Tuple16-get-k* (*Tuple16.set-l heur S*) = *Tuple16-get-k S*›
‹*Tuple16-get-l* (*Tuple16.set-l heur S*) = *heur*›
‹*Tuple16-get-m* (*Tuple16.set-l heur S*) = *Tuple16-get-m S*›
‹*Tuple16-get-n* (*Tuple16.set-l heur S*) = *Tuple16-get-n S*›
‹*Tuple16-get-o* (*Tuple16.set-l heur S*) = *Tuple16-get-o S*›
‹*Tuple16-get-p* (*Tuple16.set-l heur S*) = *Tuple16-get-p S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple16-state-simp*]:
‹*Tuple16-get-a* (*Tuple16.set-m aivdom S*) = *Tuple16-get-a S*›
‹*Tuple16-get-b* (*Tuple16.set-m aivdom S*) = *Tuple16-get-b S*›
‹*Tuple16-get-c*(*Tuple16.set-m aivdom S*) = *Tuple16-get-c S*›
‹*Tuple16-get-d* (*Tuple16.set-m aivdom S*) = *Tuple16-get-d S*›
‹*Tuple16-get-e* (*Tuple16.set-m aivdom S*) = *Tuple16-get-e S*›

‹*Tuple16-get-f* (*Tuple16.set-m aivdom S*) = *Tuple16-get-f S*›
‹*Tuple16-get-g* (*Tuple16.set-m aivdom S*) = *Tuple16-get-g S*›
‹*Tuple16-get-h* (*Tuple16.set-m aivdom S*) = *Tuple16-get-h S*›
‹*Tuple16-get-i* (*Tuple16.set-m aivdom S*) = *Tuple16-get-i S*›
‹*Tuple16-get-j* (*Tuple16.set-m aivdom S*) = *Tuple16-get-j S*›
‹*Tuple16-get-k* (*Tuple16.set-m aivdom S*) = *Tuple16-get-k S*›
‹*Tuple16-get-l* (*Tuple16.set-m aivdom S*) = *Tuple16-get-l S*›
‹*Tuple16-get-m* (*Tuple16.set-m aivdom S*) = *aivdom*›
‹*Tuple16-get-n* (*Tuple16.set-m aivdom S*) = *Tuple16-get-n S*›
‹*Tuple16-get-o* (*Tuple16.set-m aivdom S*) = *Tuple16-get-o S*›
‹*Tuple16-get-p* (*Tuple16.set-m aivdom S*) = *Tuple16-get-p S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple16-state-simp*]:
‹*Tuple16-get-a* (*Tuple16.set-n lcount S*) = *Tuple16-get-a S*›
‹*Tuple16-get-b* (*Tuple16.set-n lcount S*) = *Tuple16-get-b S*›
‹*Tuple16-get-c*(*Tuple16.set-n lcount S*) = *Tuple16-get-c S*›
‹*Tuple16-get-d* (*Tuple16.set-n lcount S*) = *Tuple16-get-d S*›
‹*Tuple16-get-e* (*Tuple16.set-n lcount S*) = *Tuple16-get-e S*›
‹*Tuple16-get-f* (*Tuple16.set-n lcount S*) = *Tuple16-get-f S*›
‹*Tuple16-get-g* (*Tuple16.set-n lcount S*) = *Tuple16-get-g S*›
‹*Tuple16-get-h* (*Tuple16.set-n lcount S*) = *Tuple16-get-h S*›
‹*Tuple16-get-i* (*Tuple16.set-n lcount S*) = *Tuple16-get-i S*›
‹*Tuple16-get-j* (*Tuple16.set-n lcount S*) = *Tuple16-get-j S*›
‹*Tuple16-get-k* (*Tuple16.set-n lcount S*) = *Tuple16-get-k S*›
‹*Tuple16-get-l* (*Tuple16.set-n lcount S*) = *Tuple16-get-l S*›
‹*Tuple16-get-m* (*Tuple16.set-n lcount S*) = *Tuple16-get-m S*›
‹*Tuple16-get-o* (*Tuple16.set-n lcount S*) = *Tuple16-get-o S*›
‹*Tuple16-get-p* (*Tuple16.set-n lcount S*) = *Tuple16-get-p S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple16-state-simp*]:
‹*Tuple16-get-a* (*set-o lcount S*) = *Tuple16-get-a S*›
‹*Tuple16-get-b* (*set-o lcount S*) = *Tuple16-get-b S*›
‹*Tuple16-get-c*(*set-o lcount S*) = *Tuple16-get-c S*›
‹*Tuple16-get-d* (*set-o lcount S*) = *Tuple16-get-d S*›
‹*Tuple16-get-e* (*set-o lcount S*) = *Tuple16-get-e S*›
‹*Tuple16-get-f* (*set-o lcount S*) = *Tuple16-get-f S*›
‹*Tuple16-get-g* (*set-o lcount S*) = *Tuple16-get-g S*›
‹*Tuple16-get-h* (*set-o lcount S*) = *Tuple16-get-h S*›
‹*Tuple16-get-i* (*set-o lcount S*) = *Tuple16-get-i S*›
‹*Tuple16-get-j* (*set-o lcount S*) = *Tuple16-get-j S*›
‹*Tuple16-get-k* (*set-o lcount S*) = *Tuple16-get-k S*›
‹*Tuple16-get-l* (*set-o lcount S*) = *Tuple16-get-l S*›
‹*Tuple16-get-m* (*set-o lcount S*) = *Tuple16-get-m S*›
‹*Tuple16-get-n* (*set-o lcount S*) = *Tuple16-get-n S*›
‹*Tuple16-get-o* (*set-o lcount S*) = *lcount*›
‹*Tuple16-get-p* (*set-o lcount S*) = *Tuple16-get-p S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple16-state-simp*]:
‹*Tuple16-get-a* (*set-p old-arena S*) = *Tuple16-get-a S*›
‹*Tuple16-get-b* (*set-p old-arena S*) = *Tuple16-get-b S*›
‹*Tuple16-get-c*(*set-p old-arena S*) = *Tuple16-get-c S*›
‹*Tuple16-get-d* (*set-p old-arena S*) = *Tuple16-get-d S*›
‹*Tuple16-get-e* (*set-p old-arena S*) = *Tuple16-get-e S*›

‹*Tuple16-get-f* (*set-p old-arena S*) = *Tuple16-get-f S*›
‹*Tuple16-get-g* (*set-p old-arena S*) = *Tuple16-get-g S*›
‹*Tuple16-get-h* (*set-p old-arena S*) = *Tuple16-get-h S*›
‹*Tuple16-get-i* (*set-p old-arena S*) = *Tuple16-get-i S*›
‹*Tuple16-get-j* (*set-p old-arena S*) = *Tuple16-get-j S*›
‹*Tuple16-get-k* (*set-p old-arena S*) = *Tuple16-get-k S*›
‹*Tuple16-get-l* (*set-p old-arena S*) = *Tuple16-get-l S*›
‹*Tuple16-get-m* (*set-p old-arena S*) = *Tuple16-get-m S*›
‹*Tuple16-get-n* (*set-p old-arena S*) = *Tuple16-get-n S*›
‹*Tuple16-get-o* (*set-p old-arena S*) = *Tuple16-get-o S*›
‹*Tuple16-get-p* (*set-p old-arena S*) = *old-arena*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemmas [*simp*] = *tuple16-state-simp*

named-theorems *tuple16-getters-setters* ‹*Definition of getters and setters*›

end
theory *IsaSAT-Stats*
imports *IsaSAT-Literals IsaSAT-EMA IsaSAT-Rephase IsaSAT-Reluctant Tuple16*
begin

5.2 Statistics

We do some statistics on the run.

NB: the statistics are not proven correct (especially they might overflow), there are just there to look for regressions, do some comparisons (e.g., to conclude that we are propagating slower than the other solvers), or to test different option combinations.

type-synonym *limit* = ‹*64 word* × *64 word*›

The statistics have the following meaning:

1. search information (propagations, conflicts, decision, restarts, reductions, fixed variables, GCs, units since last GC, fixed irredundant clss),
2. binary simplification (binary unit, binary red removed),
3. pure literals (purelit removed, purelit rounds),
4. forward subsumption (forward rounds, forward strengthen, forward subsumed)
5. other: max kept lbd,
6. ticks
7. average lbds
8. heuristics

At first we used a tuple that became longer and longer. We even had statistics bug because we changed the wrong element of the tuple. Therefore, we changed to a structure and kept some free spots.

type-synonym *search-stats* = ‹*64 word* × *64 word* ×*64 word* × *64 word* × *64 word* × *64 word* ×*64 word* × *64 word* × *64 word* × *64 word*›

definition *Search-Stats-propagations* :: ‹*search-stats* ⇒ *64 word*› **where**
‹*Search-Stats-propagations* = (λ(*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). *propa*)›

definition *Search-Stats-incr-propagation* :: ‹*search-stats* ⇒ *search-stats*› **where**
‹*Search-Stats-incr-propagation* = (λ(*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). (*propa*+*1*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*))›

definition *Search-Stats-incr-propagation-by* :: ‹*64 word* ⇒ *search-stats* ⇒ *search-stats*› **where**
‹*Search-Stats-incr-propagation-by* = (λ*p* (*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). (*propa*+*p*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*))›

definition *Search-Stats-conflicts* :: ‹*search-stats* ⇒ *64 word*› **where**
‹*Search-Stats-conflicts* = (λ(*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). *confl*)›

definition *Search-Stats-incr-conflicts* :: ‹*search-stats* ⇒ *search-stats*› **where**
‹*Search-Stats-incr-conflicts* = (λ(*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). (*propa*, *confl*+*1*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*))›

definition *Search-Stats-decisions* :: ‹*search-stats* ⇒ *64 word*› **where**
‹*Search-Stats-decisions* = (λ(*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). *dec*)›

definition *Search-Stats-incr-decisions* :: ‹*search-stats* ⇒ *search-stats*› **where**
‹*Search-Stats-incr-decisions* = (λ(*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). (*propa*, *confl*, *dec*+*1*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*))›

definition *Search-Stats-restarts* :: ‹*search-stats* ⇒ *64 word*› **where**
‹*Search-Stats-restarts* = (λ(*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). *res*)›

definition *Search-Stats-incr-restarts* :: ‹*search-stats* ⇒ *search-stats*› **where**
‹*Search-Stats-incr-restarts* = (λ(*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). (*propa*, *confl*, *dec*, *res*+*1*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*))›

definition *Search-Stats-reductions* :: ‹*search-stats* ⇒ *64 word*› **where**
‹*Search-Stats-reductions* = (λ(*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). *reduction*)›

definition *Search-Stats-incr-reductions* :: ‹*search-stats* ⇒ *search-stats*› **where**
‹*Search-Stats-incr-reductions* = (λ(*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). (*propa*, *confl*, *dec*, *res*, *reduction*+*1*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*))›

definition *Search-Stats-fixed* :: ‹*search-stats* ⇒ *64 word*› **where**
‹*Search-Stats-fixed* = (λ(*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). *uset*)›

definition *Search-Stats-incr-fixed* :: ‹*search-stats* ⇒ *search-stats*› **where**
‹*Search-Stats-incr-fixed* = (λ(*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). (*propa*, *confl*, *dec*, *res*, *reduction*, *uset*+*1*, *gcs*, *units*, *irred-cls*, *no-conflict-until*))›

definition *Search-Stats-incr-fixed-by* :: ‹*64 word* ⇒ *search-stats* ⇒ *search-stats*› **where**
‹*Search-Stats-incr-fixed-by* = (λ*p* (*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). (*propa*, *confl*, *dec*, *res*, *reduction*, *uset*+*p*, *gcs*, *units*, *irred-cls*, *no-conflict-until*))›

definition *Search-Stats-gcs* :: ‹*search-stats* ⇒ *64 word*› **where**
‹*Search-Stats-gcs* = (λ(*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). *gcs*)›

definition *Search-Stats-incr-gcs* :: ‹*search-stats* ⇒ *search-stats*› **where**
‹*Search-Stats-incr-gcs* = (λ(*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). (*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*+*1*, *units*, *irred-cls*, *no-conflict-until*))›

definition *Search-Stats-reset-units-since-gc* :: ‹*search-stats* ⇒ *search-stats*› **where**
‹*Search-Stats-reset-units-since-gc* = (λ(*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). (*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *0*, *irred-cls*, *no-conflict-until*))›

definition *Search-Stats-units-since-gcs* :: ‹*search-stats* ⇒ *64 word*› **where**
‹*Search-Stats-units-since-gcs* = (λ(*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *units-since-gcs*, *units*, *irred-cls*, *no-conflict-until*). *units-since-gcs*)›

definition *Search-Stats-incr-units-since-gc* :: ‹*search-stats* ⇒ *search-stats*› **where**
‹*Search-Stats-incr-units-since-gc* = (λ(*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). (*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*+*1*, *irred-cls*, *no-conflict-until*))›

definition *Search-Stats-incr-units-since-gc-by* :: ‹*64 word* ⇒ *search-stats* ⇒ *search-stats*› **where**
‹*Search-Stats-incr-units-since-gc-by* = (λ*p* (*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). (*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*+*p*, *irred-cls*, *no-conflict-until*))›

definition *Search-Stats-incr-irred* :: ‹*search-stats* ⇒ *search-stats*› **where**
‹*Search-Stats-incr-irred* = (λ(*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). (*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*+*1*, *no-conflict-until*))›

definition *Search-Stats-decr-irred* :: ‹*search-stats* ⇒ *search-stats*› **where**
‹*Search-Stats-decr-irred* = (λ(*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). (*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*−*1*, *no-conflict-until*))›

definition *Search-Stats-irred* :: ‹*search-stats* ⇒ *64 word*› **where**
‹*Search-Stats-irred* = (λ(*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). *irred-cls*)›

definition *Search-Stats-no-conflict-until* :: ‹*search-stats* ⇒ *64 word*› **where**
‹*Search-Stats-no-conflict-until* = (λ(*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). *no-conflict-until*)›

definition *Search-Stats-set-no-conflict-until* :: ‹*64 word* ⇒ *search-stats* ⇒ *search-stats*› **where**
‹*Search-Stats-set-no-conflict-until* = (λ*p* (*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *no-conflict-until*). (*propa*, *confl*, *dec*, *res*, *reduction*, *uset*, *gcs*, *units*, *irred-cls*, *p*))›

type-synonym *inprocessing-binary-stats* = ‹*64 word* × *64 word* × *64 word*›

definition *Binary-Stats-incr-rounds* :: ‹*inprocessing-binary-stats* ⇒ *inprocessing-binary-stats*› **where**
‹*Binary-Stats-incr-rounds* = (λ(*rounds*, *units*, *removed*). (*rounds* + *1*, *units*, *removed*))›

definition *Binary-Stats-incr-units* :: ‹*inprocessing-binary-stats* ⇒ *inprocessing-binary-stats*› **where**
‹*Binary-Stats-incr-units* = (λ(*rounds*, *units*, *removed*). (*rounds*, *units*+*1*, *removed*))›

definition *Binary-Stats-incr-removed* :: ‹*inprocessing-binary-stats* ⇒ *inprocessing-binary-stats*› **where**
‹*Binary-Stats-incr-removed* = (λ(*rounds*, *units*, *removed*). (*rounds*, *units*, *removed*+*1*))›

ticks is currently unused: it is supposed to be used as to limit the number of clauses to try.

type-synonym *inprocessing-subsumption-stats* = ‹*64 word* × *64 word* × *64 word* × *64 word* × *64 word*›

definition *Subsumption-Stats-rounds* **where**
‹*Subsumption-Stats-rounds* = (λ(*rounds*, *strengthened*, *subsumed*, -). *rounds*)›

definition *Subsumption-Stats-subsumed* **where**
‹*Subsumption-Stats-subsumed* = (λ(*subsumed*, *strengthened*, *subsumed*, -). *subsumed*)›

definition *Subsumption-Stats-tried* **where**
‹*Subsumption-Stats-tried* = (λ(*subsumed*, *strengthened*, *subsumed*, -, *tried*). *tried*)›

definition *Subsumption-Stats-strengthened* **where**
‹*Subsumption-Stats-strengthened* = (λ(*rounds*, *strengethened*, *subsumed*, -). *strengethened*)›

definition *Subsumption-Stats-incr-rounds* :: ‹*inprocessing-subsumption-stats* ⇒ *inprocessing-subsumption-stats*›
where
‹*Subsumption-Stats-incr-rounds* = (λ(*rounds*, *units*, *removed*, *ticks*, *tried*). (*rounds* + *1*, *units*, *removed*, *ticks*, *tried*))›

definition *Subsumption-Stats-incr-strengthening* :: ‹*inprocessing-subsumption-stats* ⇒ *inprocessing-subsumption-stats*›
where
‹*Subsumption-Stats-incr-strengthening* = (λ(*rounds*, *units*, *removed*). (*rounds*, *units*+*1*, *removed*))›

definition *Subsumption-Stats-incr-subsumed* :: ‹*inprocessing-subsumption-stats* ⇒ *inprocessing-subsumption-stats*›
where
‹*Subsumption-Stats-incr-subsumed* = (λ(*rounds*, *units*, *removed*, *ticks*). (*rounds*, *units*, *removed*+*1*, *ticks*))›

definition *Subsumption-Stats-incr-tried* :: ‹*inprocessing-subsumption-stats* ⇒ *inprocessing-subsumption-stats*›
where
‹*Subsumption-Stats-incr-tried* = (λ(*rounds*, *units*, *removed*, *ticks*, *tried*). (*rounds*, *units*, *removed*, *ticks*, *tried*+*1*))›

definition *Subsumption-Stats-set-ticks-limit* :: ‹*64 word* ⇒ *inprocessing-subsumption-stats* ⇒ *inprocessing-subsumption-stats*› **where**
‹*Subsumption-Stats-set-ticks-limit* = (λ*ticks* (*rounds*, *units*, *removed*, -, *tried*). (*rounds*, *units*, *removed*+*1*, *ticks*, *tried*))›

definition *Subsumption-Stats-ticks-limit* :: ‹*inprocessing-subsumption-stats* ⇒ *64 word*› **where**
‹*Subsumption-Stats-ticks-limit* = (λ(*rounds*, *units*, *removed*, *ticks*, *tried*). *ticks*)›

definition *Subsumption-Stats-ticks-tried* :: ‹*inprocessing-subsumption-stats* ⇒ *64 word*› **where**
‹*Subsumption-Stats-ticks-tried* = (λ(*rounds*, *units*, *removed*, *ticks*, *tried*). *tried*)›

type-synonym *inprocessing-pure-lits-stats* = ‹*64 word* × *64 word*›

definition *Pure-lits-Stats-incr-rounds* :: ‹*inprocessing-pure-lits-stats* ⇒ *inprocessing-pure-lits-stats*› **where**
‹*Pure-lits-Stats-incr-rounds* = (λ(*rounds*, *removed*). (*rounds* + *1*, *removed*))›

definition *Pure-lits-Stats-incr-removed* :: ‹*inprocessing-pure-lits-stats* ⇒ *inprocessing-pure-lits-stats*› **where**
‹*Pure-lits-Stats-incr-removed* = (λ(*rounds*, *removed*). (*rounds*, *removed*+*1*))›

type-synonym *lbd-size-limit-stats* = ‹*nat* × *nat*›

definition *LSize-Stats-lbd* **where**
‹*LSize-Stats-lbd* = (λ(*lbd*, *size*). *lbd*)›

definition *LSize-Stats-size* **where**
‹*LSize-Stats-size* = (λ(*lbd*, *size*). *size*)›

definition *LSize-Stats* **where**
‹*LSize-Stats lbd size'* = (*lbd*, *size'*)›

type-synonym *rephase-stats* = ‹*64 word* × *64 word* × *64 word* × *64 word* × *64 word* × *64 word*›

definition *Rephase-Stats-total* :: ‹*rephase-stats* ⇒ -› **where**
‹*Rephase-Stats-total* = (λ(*rephased*, *original*, *best*, *invert*, *flipped*, *random*). *rephased*)›

definition *Rephase-Stats-incr-total* :: ‹*rephase-stats* ⇒ *rephase-stats*› **where**
‹*Rephase-Stats-incr-total* = (λ(*rephased*, *original*, *best*, *invert*, *flipped*, *random*). (*rephased+1*, *original*, *best*, *invert*, *flipped*, *random*))›

type-synonym *isasat-stats* = ‹(*search-stats*, *inprocessing-binary-stats*, *inprocessing-subsumption-stats*, *ema*,
inprocessing-pure-lits-stats, *lbd-size-limit-stats*, *rephase-stats*, *64 word*,
64 word, *64 word*,*64 word*, *64 word*,
64 word, *64 word*, *32 word*, *64 word*) *tuple16*›

abbreviation *Stats* :: ‹- ⇒ - ⇒- ⇒ - ⇒- ⇒ - ⇒- ⇒ - ⇒- ⇒ - ⇒- ⇒ - ⇒- ⇒ - ⇒- ⇒ - ⇒ *isasat-stats*› **where**
‹*Stats* ≡ *Tuple16*›

definition *get-search-stats* :: ‹*isasat-stats* ⇒ *search-stats*› **where**
‹*get-search-stats* ≡ *Tuple16.Tuple16-get-a*›

definition *get-binary-stats* :: ‹*isasat-stats* ⇒ *inprocessing-binary-stats*› **where**
‹*get-binary-stats* ≡ *Tuple16.Tuple16-get-b*›

definition *get-subsumption-stats* :: ‹*isasat-stats* ⇒ *inprocessing-subsumption-stats*› **where**
‹*get-subsumption-stats* ≡ *Tuple16.Tuple16-get-c*›

definition *get-avg-lbd-stats* :: ‹*isasat-stats* ⇒ *ema*› **where**
‹*get-avg-lbd-stats* ≡ *Tuple16.Tuple16-get-d*›

definition *get-pure-lits-stats* :: ‹*isasat-stats* ⇒ *inprocessing-pure-lits-stats*› **where**
‹*get-pure-lits-stats* ≡ *Tuple16.Tuple16-get-e*›

definition *get-lsize-limit-stats* :: ‹*isasat-stats* ⇒ *lbd-size-limit-stats*› **where**
‹*get-lsize-limit-stats* ≡ *Tuple16.Tuple16-get-f*›

definition *get-rephase-stats* :: ‹*isasat-stats* ⇒ *rephase-stats*› **where**
‹*get-rephase-stats* ≡ *Tuple16.Tuple16-get-g*›

definition *set-propagation-stats* :: ‹*search-stats* ⇒ *isasat-stats* ⇒ *isasat-stats*› **where**
‹*set-propagation-stats* ≡ *Tuple16.set-a*›

definition *set-binary-stats* :: ‹*inprocessing-binary-stats* ⇒ *isasat-stats* ⇒ *isasat-stats*› **where**
‹*set-binary-stats* ≡ *Tuple16.set-b*›

definition *set-subsumption-stats* :: ‹*inprocessing-subsumption-stats* ⇒ *isasat-stats* ⇒ *isasat-stats*› **where**
‹*set-subsumption-stats* ≡ *Tuple16.set-c*›

definition *set-avg-lbd-stats* :: ‹*ema* ⇒ *isasat-stats* ⇒ *isasat-stats*› **where**
‹*set-avg-lbd-stats* ≡ *Tuple16.set-d*›

definition *set-pure-lits-stats* :: ‹*inprocessing-pure-lits-stats* ⇒ *isasat-stats* ⇒ *isasat-stats*› **where**
‹*set-pure-lits-stats* ≡ *Tuple16.set-e*›

definition *set-lsize-limit-stats* :: ‹*lbd-size-limit-stats* ⇒ *isasat-stats* ⇒ *isasat-stats*› **where**
‹*set-lsize-limit-stats* ≡ *Tuple16.set-f*›

definition *set-rephase-stats* :: ‹*rephase-stats* ⇒ *isasat-stats* ⇒ *isasat-stats*› **where**
‹*set-rephase-stats* ≡ *Tuple16.set-g*›

definition *incr-propagation* :: ‹*isasat-stats* ⇒ *isasat-stats*› **where**
‹*incr-propagation S* = (*set-propagation-stats* (*Search-Stats-incr-propagation* (*get-search-stats S*)) *S*)›

definition *incr-propagation-by* :: ‹*64 word* ⇒ *isasat-stats* ⇒ *isasat-stats*› **where**
‹*incr-propagation-by p S* = (*set-propagation-stats* (*Search-Stats-incr-propagation-by p* (*get-search-stats S*)) *S*)›

definition *incr-conflict* :: ‹*isasat-stats* ⇒ *isasat-stats*› **where**
‹*incr-conflict S* = (*set-propagation-stats* (*Search-Stats-incr-conflicts* (*get-search-stats S*)) *S*)›

definition *incr-decision* :: ‹*isasat-stats* ⇒ *isasat-stats*› **where**
‹*incr-decision S* = (*set-propagation-stats* (*Search-Stats-incr-decisions* (*get-search-stats S*)) *S*)›

definition *incr-restart* :: ‹*isasat-stats* ⇒ *isasat-stats*› **where**
‹*incr-restart S* = (*set-propagation-stats* (*Search-Stats-incr-restarts* (*get-search-stats S*)) *S*)›

definition *incr-reduction* :: ‹*isasat-stats* ⇒ *isasat-stats*› **where**
‹*incr-reduction S* = (*set-propagation-stats* (*Search-Stats-incr-reductions* (*get-search-stats S*)) *S*)›

definition *incr-uset* :: ‹*isasat-stats* ⇒ *isasat-stats*› **where**
‹*incr-uset S* = (*set-propagation-stats* (*Search-Stats-incr-fixed* (*get-search-stats S*)) *S*)›

definition *incr-uset-by* :: ‹*64 word* ⇒ *isasat-stats* ⇒ *isasat-stats*› **where**
‹*incr-uset-by p S* = (*set-propagation-stats* (*Search-Stats-incr-fixed-by p* (*get-search-stats S*)) *S*)›

definition *incr-GC* :: ‹*isasat-stats* ⇒ *isasat-stats*› **where**
‹*incr-GC S* = (*set-propagation-stats* (*Search-Stats-incr-gcs* (*get-search-stats S*)) *S*)›

definition *units-since-last-GC* :: ‹*isasat-stats* ⇒ *64 word*› **where**
‹*units-since-last-GC* = *Search-Stats-units-since-gcs o get-search-stats*›

definition *units-since-beginning* :: ‹*isasat-stats* ⇒ *64 word*› **where**
‹*units-since-beginning* = *Search-Stats-fixed o get-search-stats*›

definition *incr-units-since-last-GC* :: ‹*isasat-stats* ⇒ *isasat-stats*› **where**
‹*incr-units-since-last-GC S* = (*set-propagation-stats* (*Search-Stats-incr-units-since-gc* (*get-search-stats S*)) *S*)›

definition *incr-units-since-last-GC-by* :: ‹*64 word* ⇒ *isasat-stats* ⇒ *isasat-stats*› **where**
‹*incr-units-since-last-GC-by p S* = (*set-propagation-stats* (*Search-Stats-incr-units-since-gc-by p* (*get-search-stats S*)) *S*)›

definition *stats-conflicts* :: ‹*isasat-stats* ⇒ *64 word*› **where**
‹*stats-conflicts* = *Search-Stats-conflicts o get-search-stats*›

definition *incr-binary-unit-derived* :: ‹*isasat-stats* ⇒ *isasat-stats*› **where**
‹*incr-binary-unit-derived S* = *set-binary-stats* (*Binary-Stats-incr-units* (*get-binary-stats S*)) *S*›

definition *incr-binary-red-removed* :: ‹*isasat-stats* ⇒ *isasat-stats*› **where**
‹*incr-binary-red-removed S* = *set-binary-stats* (*Binary-Stats-incr-removed* (*get-binary-stats S*)) *S*›

definition *incr-forward-strengethening* :: ‹*isasat-stats* ⇒ *isasat-stats*› **where**
‹*incr-forward-strengethening S* = *set-subsumption-stats* (*Subsumption-Stats-incr-strengthening* (*get-subsumption-stats S*)) *S*›

definition *incr-forward-subsumed* :: ‹*isasat-stats* ⇒ *isasat-stats*› **where**
‹*incr-forward-subsumed S* = *set-subsumption-stats* (*Subsumption-Stats-incr-subsumed* (*get-subsumption-stats S*)) *S*›

definition *incr-forward-tried* :: ‹*isasat-stats* ⇒ *isasat-stats*› **where**
‹*incr-forward-tried S* = *set-subsumption-stats* (*Subsumption-Stats-incr-tried* (*get-subsumption-stats S*)) *S*›

definition *incr-forward-rounds* :: ‹*isasat-stats* ⇒ *isasat-stats*› **where**
‹*incr-forward-rounds S* = *set-subsumption-stats* (*Subsumption-Stats-incr-rounds* (*get-subsumption-stats S*)) *S*›

definition *stats-forward-rounds* :: ‹*isasat-stats* ⇒ -› **where**
‹*stats-forward-rounds* = *Subsumption-Stats-rounds o get-subsumption-stats*›

definition *stats-forward-subsumed* :: ‹*isasat-stats* ⇒ -› **where**
‹*stats-forward-subsumed* = *Subsumption-Stats-subsumed o get-subsumption-stats*›

definition *stats-forward-strengthened* :: ‹*isasat-stats* ⇒ -› **where**
‹*stats-forward-strengthened* = *Subsumption-Stats-strengthened o get-subsumption-stats*›

definition *stats-forward-tried* :: ‹*isasat-stats* ⇒ -› **where**
‹*stats-forward-tried* = *Subsumption-Stats-tried o get-subsumption-stats*›

definition *get-restart-count* **where**
‹*get-restart-count S* = *Search-Stats-restarts* (*get-search-stats S*)›

definition *get-reduction-count* **where**
‹*get-reduction-count S* = *Search-Stats-reductions* (*get-search-stats S*)›

definition *incr-rephase-total* :: ‹*isasat-stats* ⇒ *isasat-stats*› **where**
‹*incr-rephase-total S* = (*set-rephase-stats* (*Rephase-Stats-incr-total* (*get-rephase-stats S*)) *S*)›

definition *stats-rephase* :: ‹*isasat-stats* ⇒ *64 word*› **where**
‹*stats-rephase* = *Rephase-Stats-total o get-rephase-stats*›

definition *print-open-colour* :: ‹*64 word* ⇒ *unit*› **where**
‹*print-open-colour* - = ()›

definition *print-close-colour* :: ‹*64 word ⇒ unit*› **where**
‹*print-close-colour - = ()*›

definition *stats-propagations* :: ‹*isasat-stats ⇒ 64 word*› **where**
‹*stats-propagations = Search-Stats-propagations o get-search-stats*›

definition *stats-restarts* :: ‹*isasat-stats ⇒ 64 word*› **where**
‹*stats-restarts = Search-Stats-restarts o get-search-stats*›

definition *stats-reductions* :: ‹*isasat-stats ⇒ 64 word*› **where**
‹*stats-reductions = Search-Stats-reductions o get-search-stats*›

definition *stats-fixed* :: ‹*isasat-stats ⇒ 64 word*› **where**
‹*stats-fixed = Search-Stats-fixed o get-search-stats*›

definition *stats-gcs* :: ‹*isasat-stats ⇒ 64 word*› **where**
‹*stats-gcs = Search-Stats-gcs o get-search-stats*›

definition *stats-avg-lbd* :: ‹*isasat-stats ⇒ ema*› **where**
‹*stats-avg-lbd = get-avg-lbd-stats*›

definition *stats-decisions* :: ‹*isasat-stats ⇒ 64 word*› **where**
‹*stats-decisions = Search-Stats-decisions o get-search-stats*›

definition *stats-irred* :: ‹*isasat-stats ⇒ 64 word*› **where**
‹*stats-irred = Search-Stats-irred o get-search-stats*›

definition *Binary-Stats-rounds* :: ‹*inprocessing-binary-stats ⇒ 64 word*› **where**
‹*Binary-Stats-rounds* = (λ(*rounds*, *units*, *removed*). *rounds*)›

definition *stats-binary-rounds* :: ‹*isasat-stats ⇒ 64 word*› **where**
‹*stats-binary-rounds = Binary-Stats-rounds o get-binary-stats*›

definition *Binary-Stats-units* :: ‹*inprocessing-binary-stats ⇒ 64 word*› **where**
‹*Binary-Stats-units* = (λ(*units*, *units*, *removed*). *units*)›

definition *stats-binary-units* :: ‹*isasat-stats ⇒ 64 word*› **where**
‹*stats-binary-units = Binary-Stats-units o get-binary-stats*›

definition *Binary-Stats-removed* :: ‹*inprocessing-binary-stats ⇒ 64 word*› **where**
‹*Binary-Stats-removed* = (λ(*removed*, *units*, *removed*). *removed*)›

definition *stats-binary-removed* :: ‹*isasat-stats ⇒ 64 word*› **where**
‹*stats-binary-removed = Binary-Stats-removed o get-binary-stats*›

definition *Pure-Lits-Stats-removed* :: ‹*inprocessing-pure-lits-stats ⇒ 64 word*› **where**
‹*Pure-Lits-Stats-removed* = (λ(*removed*, *removed*). *removed*)›

definition *stats-pure-lits-removed* :: ‹*isasat-stats ⇒ 64 word*› **where**
‹*stats-pure-lits-removed = Pure-Lits-Stats-removed o get-pure-lits-stats*›

definition *Pure-Lits-Stats-rounds* :: ‹*inprocessing-pure-lits-stats ⇒ 64 word*› **where**
‹*Pure-Lits-Stats-rounds* = (λ(*rounds*, *rounds*). *rounds*)›

definition *stats-pure-lits-rounds* :: ‹*isasat-stats* ⇒ *64 word*› **where**
‹*stats-pure-lits-rounds* = *Pure-Lits-Stats-rounds o get-pure-lits-stats*›

definition *add-lbd* :: ‹*32 word* ⇒ *isasat-stats* ⇒ *isasat-stats*› **where**
‹*add-lbd lbd S* = *set-avg-lbd-stats* (*ema-update* (*unat lbd*) (*get-avg-lbd-stats S*)) *S*›

definition *incr-purelit-elim* :: ‹*isasat-stats* ⇒ *isasat-stats*› **where**
‹*incr-purelit-elim S* = *set-pure-lits-stats* (*Pure-lits-Stats-incr-removed* (*get-pure-lits-stats S*)) *S*›

definition *incr-purelit-rounds* :: ‹*isasat-stats* ⇒ *isasat-stats*› **where**
‹*incr-purelit-rounds S* = *set-pure-lits-stats* (*Pure-lits-Stats-incr-rounds* (*get-pure-lits-stats S*)) *S*›

definition *reset-units-since-last-GC* :: ‹*isasat-stats* ⇒ *isasat-stats*› **where**
‹*reset-units-since-last-GC S* = (*set-propagation-stats* (*Search-Stats-reset-units-since-gc* (*get-search-stats S*)) *S*)›

definition *incr-irred-clss* :: ‹*isasat-stats* ⇒ *isasat-stats*› **where**
‹*incr-irred-clss S* = *set-propagation-stats* (*Search-Stats-incr-irred* (*get-search-stats S*)) *S*›

definition *set-no-conflict-until* :: ‹*64 word* ⇒ *isasat-stats* ⇒ *isasat-stats*› **where**
‹*set-no-conflict-until p S* = *set-propagation-stats* (*Search-Stats-set-no-conflict-until p* (*get-search-stats S*)) *S*›

definition *no-conflict-until* :: ‹*isasat-stats* ⇒ *64 word*› **where**
‹*no-conflict-until S* = *Search-Stats-no-conflict-until* (*get-search-stats S*)›

definition *decr-irred-clss* :: ‹*isasat-stats* ⇒ *isasat-stats*› **where**
‹*decr-irred-clss S* = *set-propagation-stats* (*Search-Stats-decr-irred* (*get-search-stats S*)) *S*›

definition *irredundant-clss* :: ‹*isasat-stats* ⇒ *64 word*› **where**
‹*irredundant-clss* = *Search-Stats-irred o get-search-stats*›

abbreviation (*input*) *get-conflict-count* :: ‹*isasat-stats* ⇒ *64 word*› **where**
‹*get-conflict-count* ≡ *stats-conflicts*›

definition *stats-lbd-limit* :: ‹*isasat-stats* ⇒ *nat*› **where**
‹*stats-lbd-limit* = *LSize-Stats-lbd o get-lsize-limit-stats*›

definition *stats-size-limit* :: ‹*isasat-stats* ⇒ *nat*› **where**
‹*stats-size-limit* = *LSize-Stats-size o get-lsize-limit-stats*›

definition *set-stats-size-limit* :: ‹*nat* ⇒ *nat* ⇒ *isasat-stats* ⇒ *isasat-stats*› **where**
‹*set-stats-size-limit lbd size′* = *set-lsize-limit-stats* (*lbd*, *size′*)›

5.3 Information related to restarts

definition *FOCUSED-MODE* :: ‹*64 word*› **where**
‹*FOCUSED-MODE* = *0*›

definition *STABLE-MODE* :: ‹*64 word*› **where**
‹*STABLE-MODE* = *1*›

definition *DEFAULT-INIT-PHASE* :: ‹*64 word*› **where**

‹*DEFAULT-INIT-PHASE* = *10000*›

type-synonym *restart-info* = ‹*64 word* × *64 word* × *64 word* × *64 word* × *64 word*›

definition *incr-conflict-count-since-last-restart* :: ‹*restart-info* ⇒ *restart-info*› **where**
‹*incr-conflict-count-since-last-restart* = (λ(*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase*, *length-phase*).
(*ccount* + *1*, *ema-lvl*, *restart-phase*, *end-of-phase*, *length-phase*))›

definition *restart-info-update-lvl-avg* :: ‹*32 word* ⇒ *restart-info* ⇒ *restart-info*› **where**
‹*restart-info-update-lvl-avg* = (λ*lvl* (*ccount*, *ema-lvl*). (*ccount*, *ema-lvl*))›

definition *restart-info-init* :: ‹*restart-info*› **where**
‹*restart-info-init* = (*0*, *0*, *FOCUSED-MODE*, *DEFAULT-INIT-PHASE*, *1000*)›

definition *restart-info-restart-done* :: ‹*restart-info* ⇒ *restart-info*› **where**
‹*restart-info-restart-done* = (λ(*ccount*, *lvl-avg*). (*0*, *lvl-avg*))›

definition *empty-stats* :: ‹*isasat-stats*› **where**
‹*empty-stats* = *Tuple16*((*0*,*0*,*0*,*0*,*0*,*0*,*0*,*0*,*0*,*0*)::*search-stats*)
((*0*,*0*,*0*)::*inprocessing-binary-stats*) ((*0*,*0*,*0*,*0*,*0*)::*inprocessing-subsumption-stats*)
(*ema-fast-init*::*ema*) ((*0*,*0*)::*inprocessing-pure-lits-stats*) (*0*,*0*) (*0*,*0*,*0*,*0*,*0*,*0*) *0 0 0 0 0 0 0 0 0*›

definition *empty-stats-clss* :: ‹*64 word* ⇒ *isasat-stats*› **where**
‹*empty-stats-clss n* = *Tuple16*((*0*,*0*,*0*,*0*,*0*,*0*,*0*,*0*,*n*,*0*)::*search-stats*)
((*0*,*0*,*0*)::*inprocessing-binary-stats*) ((*0*,*0*,*0*,*0*,*0*)::*inprocessing-subsumption-stats*)
(*ema-fast-init*::*ema*) ((*0*,*0*)::*inprocessing-pure-lits-stats*) (*0*,*0*) (*0*,*0*,*0*,*0*,*0*,*0*) *0 0 0 0 0 0 0 0 0*›

5.4 Heuristics

type-synonym *schedule-info* = ‹*64 word* × *64 word* × *64 word*›

definition *schedule-next-pure-lits-info* :: ‹*schedule-info* ⇒ *schedule-info*› **where**
‹*schedule-next-pure-lits-info* = (λ(*inprocess*, *reduce*, *forwardsubsumption*). (*inprocess* ∗ *3* >> *1*, *reduce*, *forwardsubsumption*))›

definition *next-pure-lits-schedule-info* :: ‹*schedule-info* ⇒ *64 word*› **where**
‹*next-pure-lits-schedule-info* = (λ(*inprocess*, *reduce*, *forwardsubsumption*). *inprocess*)›

definition *schedule-next-reduce-info* :: ‹*64 word* ⇒ *schedule-info* ⇒ *schedule-info*› **where**
‹*schedule-next-reduce-info delta* = (λ(*inprocess*, *reduce*, *forwardsubsumption*). (*inprocess*, *reduce* + *delta*, *forwardsubsumption*))›

definition *next-reduce-schedule-info* :: ‹*schedule-info* ⇒ *64 word*› **where**
‹*next-reduce-schedule-info* = (λ(*inprocess*, *reduce*, *forwardsubsumption*). *reduce*)›

definition *next-subsume-schedule-info* :: ‹*schedule-info* ⇒ *64 word*› **where**
‹*next-subsume-schedule-info* = (λ(*inprocess*, *reduce*, *forwardsubsumption*). *forwardsubsumption*)›

definition *schedule-next-subsume-info* :: ‹*64 word* ⇒ *schedule-info* ⇒ *schedule-info*› **where**
‹*schedule-next-subsume-info delta* = (λ(*inprocess*, *reduce*, *forwardsubsumption*). (*inprocess*, *reduce*, *forwardsubsumption* + *delta*))›

datatype *'a code-hider* = *Constructor* (*get-content*: *'a*)

definition *code-hider-rel* **where** *code-hider-rel-def-internal*:
‹*code-hider-rel R* = {(*a*,*b*). (*a*, *get-content b*) ∈ *R*}›

lemma *code-hider-rel-def*[*refine-rel-defs*]:
⟨*R*⟩*code-hider-rel* ≡ {(*a*,*b*). (*a*, *get-content b*) ∈ *R*}
by (*simp add*: *code-hider-rel-def-internal relAPP-def*)

type-synonym *restart-heuristics* = ‹*ema* × *ema* × *restart-info* × *64 word* × *phase-save-heur* × *reluctant* × *bool* × *phase-saver* × *schedule-info* × *ema* × *ema*›

type-synonym *isasat-restart-heuristics* = ‹*restart-heuristics code-hider*›

abbreviation *Restart-Heuristics* :: ‹*restart-heuristics* ⇒ *isasat-restart-heuristics*› **where**
‹*Restart-Heuristics a* ≡ *Constructor a*›

abbreviation *get-restart-heuristics* :: ‹*isasat-restart-heuristics* ⇒ *restart-heuristics*› **where**
‹*get-restart-heuristics a* ≡ *get-content a*›

fun *fast-ema-of-stats* :: ‹*restart-heuristics* ⇒ *ema*› **where**
‹*fast-ema-of-stats* (*fast-ema*, *slow-ema*, *restart-info*, *wasted*, *φ*) = *fast-ema*›

fun *slow-ema-of-stats* :: ‹*restart-heuristics* ⇒ *ema*› **where**
‹*slow-ema-of-stats* (*fast-ema*, *slow-ema*, *restart-info*, *wasted*, *φ*) = *slow-ema*›

fun *restart-info-of-stats* :: ‹*restart-heuristics* ⇒ *restart-info*› **where**
‹*restart-info-of-stats* (*fast-ema*, *slow-ema*, *restart-info*, *wasted*, *φ*) = *restart-info*›

fun *schedule-info-of-stats* :: ‹*restart-heuristics* ⇒ *schedule-info*› **where**
‹*schedule-info-of-stats* (*fast-ema*, *slow-ema*, *restart-info*, *wasted*, *φ*, -, -, -, *schedule*, -, -) = *schedule*›

fun *current-restart-phase-stats* :: ‹*restart-heuristics* ⇒ *64 word*› **where**
‹*current-restart-phase-stats* (*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase*), *wasted*, *φ*) =
restart-phase›

fun *incr-restart-phase-stats* :: ‹*restart-heuristics* ⇒ *restart-heuristics*› **where**
‹*incr-restart-phase-stats* (*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase*), *wasted*, *φ*) =
(*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase XOR 1*, *end-of-phase*), *wasted*, *φ*)›

fun *incr-wasted-stats* :: ‹*64 word* ⇒ *restart-heuristics* ⇒ *restart-heuristics*› **where**
‹*incr-wasted-stats waste* (*fast-ema*, *slow-ema*, *res-info*, *wasted*, *φ*) =
(*fast-ema*, *slow-ema*, *res-info*, *wasted* + *waste*, *φ*)›

fun *set-zero-wasted-stats* :: ‹*restart-heuristics* ⇒ *restart-heuristics*› **where**
‹*set-zero-wasted-stats* (*fast-ema*, *slow-ema*, *res-info*, *wasted*, *φ*) =
(*fast-ema*, *slow-ema*, *res-info*, *0*, *φ*)›

fun *wasted-of-stats* :: ‹*restart-heuristics* ⇒ *64 word*› **where**
‹*wasted-of-stats* (*fast-ema*, *slow-ema*, *res-info*, *wasted*, *φ*) = *wasted*›

fun *get-conflict-count-since-last-restart-stats* :: ‹*restart-heuristics* ⇒ *64 word*› **where**
‹*get-conflict-count-since-last-restart-stats* (*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase*), *wasted*, *φ*) =

ccount›

definition *swap-emas-stats* :: ‹*restart-heuristics* ⇒ *restart-heuristics*› **where**
‹*swap-emas-stats* = (λ(*fast-ema*, *slow-ema*, *restart-info*, *wasted*, φ, *a*, *b*, *lit-st*, *schedule*, *other-fema*, *other-sema*).
(*other-fema*, *other-sema*, *restart-info*, *wasted*, φ, *a*, *b*, *lit-st*, *schedule*, *fast-ema*, *slow-ema*))›

definition *get-conflict-count-since-last-restart* :: ‹*isasat-restart-heuristics* ⇒ *64 word*› **where**
‹*get-conflict-count-since-last-restart* = *get-conflict-count-since-last-restart-stats o get-content*›

definition *heuristic-rel-stats* :: ‹*nat multiset* ⇒ *restart-heuristics* ⇒ *bool*› **where**
‹*heuristic-rel-stats* A = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, -, -, *lit-st*, *schedule*). *phase-save-heur-rel* A φ ∧ *phase-saving* A *lit-st*)›

definition *save-phase-heur-stats* :: ‹*nat* ⇒ *bool* ⇒ *restart-heuristics* ⇒ *restart-heuristics*› **where**
‹*save-phase-heur-stats L b* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, (φ, *target*, *best*), *reluctant*).
(*fast-ema*, *slow-ema*, *res-info*, *wasted*, (φ[*L* := *b*], *target*, *best*), *reluctant*))›

definition *save-phase-heur-pre-stats* :: ‹*nat* ⇒ *bool* ⇒ *restart-heuristics* ⇒ *bool*› **where**
‹*save-phase-heur-pre-stats L b* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, (φ, -), -). *L* < *length* φ)›

definition *mop-save-phase-heur-stats* :: ‹*nat* ⇒ *bool* ⇒ *restart-heuristics* ⇒ *restart-heuristics nres*› **where**
‹*mop-save-phase-heur-stats L b heur* = *do* {
ASSERT(*save-phase-heur-pre-stats L b heur*);
RETURN (*save-phase-heur-stats L b heur*)
}›

definition *mark-added-heur-stats* :: ‹*nat* ⇒ *bool* ⇒ *restart-heuristics* ⇒ *restart-heuristics*› **where**
‹*mark-added-heur-stats L b* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant*, *fully-proped*, *lits-st*, *schedule*).
(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant*, *fully-proped*, *lits-st*[*L* := *True*], *schedule*))›

definition *mark-added-heur-pre-stats* :: ‹*nat* ⇒ *bool* ⇒ *restart-heuristics* ⇒ *bool*› **where**
‹*mark-added-heur-pre-stats L b* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, -, *fully-proped*, *lits-st*, *schedule*). *L* < *length lits-st*)›

definition *mop-mark-added-heur-stats* :: ‹*nat* ⇒ *bool* ⇒ *restart-heuristics* ⇒ *restart-heuristics nres*› **where**
‹*mop-mark-added-heur-stats L b heur* = *do* {
ASSERT(*mark-added-heur-pre-stats L b heur*);
RETURN (*mark-added-heur-stats L b heur*)
}›

definition *reset-added-heur-stats* :: ‹*restart-heuristics* ⇒ *restart-heuristics*› **where**
‹*reset-added-heur-stats* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant*, *fully-proped*, *lits-st*, *schedule*).
(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant*, *fully-proped*, *replicate* (*length lits-st*) *False*, *schedule*))›

definition *reset-added-heur-stats2* :: ‹*restart-heuristics* ⇒ *restart-heuristics nres*› **where**
‹*reset-added-heur-stats2* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant*, *fully-proped*, *lits-st*$_0$, *schedule*).
do {
(-, *lits-st*) ← *WHILE*$_T$ λ(*i*, *lits-st*). (∀ *k* < *i*. ¬*lits-st* ! *k*) ∧ *i* ≤ *length lits-st* ∧ *length lits-st* = *length lits-st*$_0$

(λ(*i*, *lits-st*). *i* < *length lits-st*)
(λ(*i*, *lits-st*). *do* {*ASSERT* (*i* < *length lits-st*); *RETURN* (*i*+*1*, *lits-st*[*i* := *False*])})
(*0*, *lits-st*$_0$);
RETURN (*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant*, *fully-proped*, *lits-st*, *schedule*)
})›

lemma *reset-added-heur-stats2-reset-added-heur-stats*:
‹*reset-added-heur-stats2 heur* ≤⇓*Id* (*RETURN* (*reset-added-heur-stats heur*))›
proof −
obtain *fast-ema slow-ema res-info wasted* φ *reluctant fully-proped lits-st*$_0$ *schedule* **where**
heur: ‹*heur* = (*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant*, *fully-proped*, *lits-st*$_0$, *schedule*)›
by (*cases heur*)
have [*refine0*]: ‹*wf* (*measure* (λ(*i*, *lits-st*). *length lits-st*$_0$ − *i*))›
by *auto*
show *?thesis*
unfolding *reset-added-heur-stats2-def reset-added-heur-stats-def heur*
prod.simps
apply (*refine-vcg*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto split*: *if-splits*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*simp add*: *list-eq-iff-nth-eq*)
done
qed

definition *is-marked-added-heur-stats* :: ‹*restart-heuristics* ⇒ *nat* ⇒ *bool*› **where**
‹*is-marked-added-heur-stats* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant*, *fully-proped*, *lits-st*, *schedule*) *L*.
lits-st ! *L*)›

definition *is-marked-added-heur-pre-stats* :: ‹*restart-heuristics* ⇒*nat* ⇒ *bool*› **where**
‹*is-marked-added-heur-pre-stats* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, -, *fully-proped*, *lits-st*, *schedule*) *L*. *L* < *length lits-st*)›

definition *mop-is-marked-added-heur-stats* :: ‹*restart-heuristics* ⇒ *nat* ⇒ *bool nres*› **where**
‹*mop-is-marked-added-heur-stats L heur* = *do* {
ASSERT(*is-marked-added-heur-pre-stats L heur*);
RETURN (*is-marked-added-heur-stats L heur*)
}›

definition *get-saved-phase-heur-pre-stats* :: ‹*nat* ⇒ *restart-heuristics* ⇒ *bool*› **where**
‹*get-saved-phase-heur-pre-stats L* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, (φ, -), -). *L* < *length* φ)›

definition *get-saved-phase-heur-stats* :: ‹*nat* ⇒ *restart-heuristics* ⇒ *bool*› **where**
‹*get-saved-phase-heur-stats L* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, (φ, -), -). φ!*L*)›

definition *current-rephasing-phase-stats* :: ‹*restart-heuristics* ⇒ *64 word*› **where**
‹*current-rephasing-phase-stats* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, -). *phase-current-rephasing-phase* φ)›

definition *mop-get-saved-phase-heur-stats* :: ‹*nat* ⇒ *restart-heuristics* ⇒ *bool nres*› **where**

‹*mop-get-saved-phase-heur-stats L heur = do* {
  ASSERT(*get-saved-phase-heur-pre-stats L heur*);
  RETURN (*get-saved-phase-heur-stats L heur*)
}›

definition *heuristic-reluctant-tick-stats* :: ‹*restart-heuristics* ⇒ *restart-heuristics*› **where**
‹*heuristic-reluctant-tick-stats* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant*, *fullyproped*).
(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant-tick reluctant*, *fullyproped*))›

definition *heuristic-reluctant-enable-stats* :: ‹*restart-heuristics* ⇒ *restart-heuristics*› **where**
‹*heuristic-reluctant-enable-stats* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant*, *fullyproped*).
(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant-init*, *fullyproped*))›

definition *heuristic-reluctant-disable-stats* :: ‹*restart-heuristics* ⇒ *restart-heuristics*› **where**
‹*heuristic-reluctant-disable-stats* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant*, *fullyproped*).
(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant-disable reluctant*, *fullyproped*))›

definition *heuristic-reluctant-triggered-stats* :: ‹*restart-heuristics* ⇒ *restart-heuristics* × *bool*› **where**
‹*heuristic-reluctant-triggered-stats* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant*, *fullyproped*).
let (*reluctant*, *b*) = *reluctant-triggered reluctant in*
((*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant*, *fullyproped*), *b*))›

definition *heuristic-reluctant-triggered2-stats* :: ‹*restart-heuristics* ⇒ *bool*› **where**
‹*heuristic-reluctant-triggered2-stats* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant*, *fullyproped*).
reluctant-triggered2 reluctant)›

definition *heuristic-reluctant-untrigger-stats* :: ‹*restart-heuristics* ⇒ *restart-heuristics*› **where**
‹*heuristic-reluctant-untrigger-stats* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant*, *fullyproped*).
(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant-untrigger reluctant*, *fullyproped*))›

definition *end-of-rephasing-phase-heur-stats* :: ‹*restart-heuristics* ⇒ *64 word*› **where**
‹*end-of-rephasing-phase-heur-stats* =
(λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, *phasing*, *reluctant*). *end-of-rephasing-phase phasing*)›

definition *is-fully-propagated-heur-stats* :: ‹*restart-heuristics* ⇒ *bool*› **where**
‹*is-fully-propagated-heur-stats* =
(λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, *phasing*, *reluctant*, *fullyproped*, -). *fullyproped*)›

definition *set-fully-propagated-heur-stats* :: ‹*restart-heuristics* ⇒ *restart-heuristics*› **where**
‹*set-fully-propagated-heur-stats* =
(λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, *phasing*, *reluctant*, *fullyproped*, *lit-st*). (*fast-ema*, *slow-ema*, *res-info*, *wasted*, *phasing*, *reluctant*, *True*, *lit-st*))›

definition *unset-fully-propagated-heur-stats* :: ‹*restart-heuristics* ⇒ *restart-heuristics*› **where**
‹*unset-fully-propagated-heur-stats* =
(λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, *phasing*, *reluctant*, *fullyproped*, *lit-st*). (*fast-ema*, *slow-ema*, *res-info*, *wasted*, *phasing*, *reluctant*, *False*, *lit-st*))›

definition *restart-info-restart-done-heur-stats* :: ‹*restart-heuristics* ⇒ *restart-heuristics*› **where**
‹*restart-info-restart-done-heur-stats* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, *phasing*, *reluctant*, *fullyproped*, *lit-st*). (*fast-ema*, *slow-ema*, *restart-info-restart-done res-info*, *wasted*, *phasing*, *reluctant*, *False*, *lit-st*))›

lemma *heuristic-rel-statsI*[*intro*!]:
‹*heuristic-rel-stats* A *heur* ⟹ *heuristic-rel-stats* A (*incr-wasted-stats wast heur*)›
‹*heuristic-rel-stats* A *heur* ⟹ *heuristic-rel-stats* A (*set-zero-wasted-stats heur*)›

‹heuristic-rel-stats $\mathcal{A}$ heur $\Longrightarrow$ heuristic-rel-stats $\mathcal{A}$ (incr-restart-phase-stats heur)›
‹heuristic-rel-stats $\mathcal{A}$ heur $\Longrightarrow$ heuristic-rel-stats $\mathcal{A}$ (save-phase-heur-stats L b heur)›
‹heuristic-rel-stats $\mathcal{A}$ heur $\Longrightarrow$ heuristic-rel-stats $\mathcal{A}$ (mark-added-heur-stats L b heur)›
‹heuristic-rel-stats $\mathcal{A}$ heur $\Longrightarrow$ heuristic-rel-stats $\mathcal{A}$ (heuristic-reluctant-tick-stats heur)›
‹heuristic-rel-stats $\mathcal{A}$ heur $\Longrightarrow$ heuristic-rel-stats $\mathcal{A}$ (heuristic-reluctant-enable-stats heur)›
‹heuristic-rel-stats $\mathcal{A}$ heur $\Longrightarrow$ heuristic-rel-stats $\mathcal{A}$ (heuristic-reluctant-untrigger-stats heur)›
‹heuristic-rel-stats $\mathcal{A}$ heur $\Longrightarrow$ heuristic-rel-stats $\mathcal{A}$ (set-fully-propagated-heur-stats heur)›
‹heuristic-rel-stats $\mathcal{A}$ heur $\Longrightarrow$ heuristic-rel-stats $\mathcal{A}$ (unset-fully-propagated-heur-stats heur)›
‹heuristic-rel-stats $\mathcal{A}$ heur $\Longrightarrow$ heuristic-rel-stats $\mathcal{A}$ (swap-emas-stats heur)›
by (*clarsimp-all simp*: *heuristic-rel-stats-def save-phase-heur-stats-def phase-save-heur-rel-def phase-saving-def heuristic-reluctant-tick-stats-def heuristic-reluctant-enable-stats-def heuristic-reluctant-untrigger-stats-def set-fully-propagated-heur-stats-def unset-fully-propagated-heur-stats-def mark-added-heur-stats-def swap-emas-stats-def*)

lemma *heuristic-rel-stats-heuristic-reluctant-triggered-statsD*:
‹heuristic-rel-stats $\mathcal{A}$ heur $\Longrightarrow$
heuristic-rel-stats $\mathcal{A}$ (fst (heuristic-reluctant-triggered-stats heur))›
by (*clarsimp simp*: *heuristic-reluctant-triggered-stats-def heuristic-rel-stats-def phase-save-heur-rel-def phase-saving-def reluctant-triggered-def*)

lemma *save-phase-heur-pre-statsI*:
‹heuristic-rel-stats $\mathcal{A}$ heur $\Longrightarrow$ a $\in\#$ $\mathcal{A}$ $\Longrightarrow$ save-phase-heur-pre-stats a b heur›
by (*auto simp*: *heuristic-rel-stats-def phase-saving-def save-phase-heur-pre-stats-def phase-save-heur-rel-def atms-of-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$*)

definition *fast-ema-of* :: ‹*isasat-restart-heuristics* ⇒ *ema*› **where**
‹*fast-ema-of* = *fast-ema-of-stats o get-restart-heuristics*›

definition *slow-ema-of* :: ‹*isasat-restart-heuristics* ⇒ *ema*› **where**
‹*slow-ema-of* = *slow-ema-of-stats o get-restart-heuristics*›

definition *restart-info-of* :: ‹*isasat-restart-heuristics* ⇒ *restart-info*› **where**
‹*restart-info-of* = *restart-info-of-stats o get-restart-heuristics*›

definition *current-restart-phase* :: ‹*isasat-restart-heuristics* ⇒ *64 word*› **where**
‹*current-restart-phase* = *current-restart-phase-stats o get-restart-heuristics*›

definition *incr-restart-phase* :: ‹*isasat-restart-heuristics* ⇒ *isasat-restart-heuristics*› **where**
‹*incr-restart-phase* = *Restart-Heuristics o incr-restart-phase-stats o get-restart-heuristics*›

definition *incr-wasted* :: ‹*64 word* ⇒ *isasat-restart-heuristics* ⇒ *isasat-restart-heuristics*› **where**
‹*incr-wasted waste* = *Restart-Heuristics o incr-wasted-stats waste o get-restart-heuristics*›

definition *set-zero-wasted* :: ‹*isasat-restart-heuristics* ⇒ *isasat-restart-heuristics*› **where**
‹*set-zero-wasted* = *Restart-Heuristics o set-zero-wasted-stats o get-restart-heuristics*›

definition *wasted-of* :: ‹*isasat-restart-heuristics* ⇒ *64 word*› **where**
‹*wasted-of* = *wasted-of-stats o get-restart-heuristics*›

definition *swap-emas* :: ‹*isasat-restart-heuristics* ⇒ *isasat-restart-heuristics*› **where**
‹*swap-emas* = *Restart-Heuristics o swap-emas-stats o get-restart-heuristics*›

definition *heuristic-rel* :: ‹*nat multiset* ⇒ *isasat-restart-heuristics* ⇒ *bool*› **where**
‹*heuristic-rel* $\mathcal{A}$ = *heuristic-rel-stats* $\mathcal{A}$ *o get-restart-heuristics*›

definition *save-phase-heur* :: ‹*nat* ⇒ *bool* ⇒ *isasat-restart-heuristics* ⇒ *isasat-restart-heuristics*› **where**
‹*save-phase-heur L b* = *Restart-Heuristics o save-phase-heur-stats L b o get-restart-heuristics*›

definition *save-phase-heur-pre* :: ‹*nat* ⇒ *bool* ⇒ *isasat-restart-heuristics* ⇒ *bool*› **where**
‹*save-phase-heur-pre L b* = *save-phase-heur-pre-stats L b o get-restart-heuristics*›

definition *mop-save-phase-heur* :: ‹*nat* ⇒ *bool* ⇒ *isasat-restart-heuristics* ⇒ *isasat-restart-heuristics nres*› **where**
‹*mop-save-phase-heur L b heur* = *do* {
ASSERT(*save-phase-heur-pre L b heur*);
RETURN (*save-phase-heur L b heur*)
}›

definition *mark-added-heur* :: ‹*nat* ⇒ *bool* ⇒ *isasat-restart-heuristics* ⇒ *isasat-restart-heuristics*› **where**
‹*mark-added-heur L b* = *Restart-Heuristics o mark-added-heur-stats L b o get-restart-heuristics*›

definition *mark-added-heur-pre* :: ‹*nat* ⇒ *bool* ⇒ *isasat-restart-heuristics* ⇒ *bool*› **where**
‹*mark-added-heur-pre L b* = *mark-added-heur-pre-stats L b o get-restart-heuristics*›

definition *mop-mark-added-heur* :: ‹*nat* ⇒ *bool* ⇒ *isasat-restart-heuristics* ⇒ *isasat-restart-heuristics nres*› **where**
‹*mop-mark-added-heur L b heur* = *do* {
ASSERT(*mark-added-heur-pre L b heur*);
RETURN (*mark-added-heur L b heur*)
}›

definition *reset-added-heur* :: ‹*isasat-restart-heuristics* ⇒ *isasat-restart-heuristics*› **where**
‹*reset-added-heur* = *Restart-Heuristics o reset-added-heur-stats o get-restart-heuristics*›

definition *mop-reset-added-heur* :: ‹*isasat-restart-heuristics* ⇒ *isasat-restart-heuristics nres*› **where**
‹*mop-reset-added-heur heur* = *do* {
RETURN (*reset-added-heur heur*)
}›

definition *is-marked-added-heur* :: ‹*isasat-restart-heuristics* ⇒ *nat* ⇒ *bool*› **where**
‹*is-marked-added-heur* = *is-marked-added-heur-stats o get-restart-heuristics*›

definition *is-marked-added-heur-pre* :: ‹*isasat-restart-heuristics* ⇒ *nat* ⇒ *bool*› **where**
‹*is-marked-added-heur-pre* = *is-marked-added-heur-pre-stats o get-restart-heuristics*›

definition *mop-is-marked-added-heur* :: ‹*isasat-restart-heuristics* ⇒ *nat* ⇒ *bool nres*› **where**
‹*mop-is-marked-added-heur L heur* = *do* {
ASSERT(*is-marked-added-heur-pre L heur*);
RETURN (*is-marked-added-heur L heur*)
}›

definition *get-saved-phase-heur-pre* :: ‹*nat* ⇒ *isasat-restart-heuristics* ⇒ *bool*› **where**
‹*get-saved-phase-heur-pre L* = *get-saved-phase-heur-pre-stats L o get-restart-heuristics*›

definition *get-saved-phase-heur* :: ‹*nat* ⇒ *isasat-restart-heuristics* ⇒ *bool*› **where**
‹*get-saved-phase-heur L* = *get-saved-phase-heur-stats L o get-restart-heuristics*›

definition *current-rephasing-phase* :: ‹*isasat-restart-heuristics* ⇒ *64 word*› **where**
‹*current-rephasing-phase* = *current-rephasing-phase-stats o get-restart-heuristics*›

definition *mop-get-saved-phase-heur* :: ‹*nat* ⇒ *isasat-restart-heuristics* ⇒ *bool nres*› **where**
‹*mop-get-saved-phase-heur L heur* = *do* {
ASSERT(*get-saved-phase-heur-pre L heur*);

RETURN (*get-saved-phase-heur L heur*)
}›

definition *heuristic-reluctant-tick* :: ‹*isasat-restart-heuristics* ⇒ *isasat-restart-heuristics*› **where**
‹*heuristic-reluctant-tick* = *Restart-Heuristics o heuristic-reluctant-tick-stats o get-restart-heuristics*›

definition *heuristic-reluctant-enable* :: ‹*isasat-restart-heuristics* ⇒ *isasat-restart-heuristics*› **where**
‹*heuristic-reluctant-enable* = *Restart-Heuristics o heuristic-reluctant-enable-stats o get-restart-heuristics*›

definition *heuristic-reluctant-disable* :: ‹*isasat-restart-heuristics* ⇒ *isasat-restart-heuristics*› **where**
‹*heuristic-reluctant-disable* = *Restart-Heuristics o heuristic-reluctant-disable-stats o get-restart-heuristics*›

definition *heuristic-reluctant-triggered* :: ‹*isasat-restart-heuristics* ⇒ *isasat-restart-heuristics* × *bool*›
where
‹*heuristic-reluctant-triggered* = *apfst Restart-Heuristics o heuristic-reluctant-triggered-stats o get-restart-heuristics*›

definition *heuristic-reluctant-triggered2* :: ‹*isasat-restart-heuristics* ⇒ *bool*› **where**
‹*heuristic-reluctant-triggered2* = *heuristic-reluctant-triggered2-stats o get-restart-heuristics*›

definition *heuristic-reluctant-untrigger* :: ‹*isasat-restart-heuristics* ⇒ *isasat-restart-heuristics*› **where**
‹*heuristic-reluctant-untrigger* = *Restart-Heuristics o heuristic-reluctant-untrigger-stats o get-restart-heuristics*›

definition *end-of-rephasing-phase-heur* :: ‹*isasat-restart-heuristics* ⇒ *64 word*› **where**
‹*end-of-rephasing-phase-heur* = *end-of-rephasing-phase-heur-stats o get-restart-heuristics*›

definition *is-fully-propagated-heur* :: ‹*isasat-restart-heuristics* ⇒ *bool*› **where**
‹*is-fully-propagated-heur* = *is-fully-propagated-heur-stats o get-restart-heuristics*›

definition *set-fully-propagated-heur* :: ‹*isasat-restart-heuristics* ⇒ *isasat-restart-heuristics*› **where**
‹*set-fully-propagated-heur* = *Restart-Heuristics o set-fully-propagated-heur-stats o get-restart-heuristics*›

definition *unset-fully-propagated-heur* :: ‹*isasat-restart-heuristics* ⇒ *isasat-restart-heuristics*› **where**
‹*unset-fully-propagated-heur* =
Restart-Heuristics o unset-fully-propagated-heur-stats o get-restart-heuristics›

definition *restart-info-restart-done-heur* :: ‹*isasat-restart-heuristics* ⇒ *isasat-restart-heuristics*› **where**
‹*restart-info-restart-done-heur* =
Restart-Heuristics o restart-info-restart-done-heur-stats o get-restart-heuristics›

definition *schedule-info-of* :: ‹*isasat-restart-heuristics* ⇒ *schedule-info*› **where**
‹*schedule-info-of* = *schedule-info-of-stats o get-restart-heuristics*›

definition *schedule-next-pure-lits-stats* :: ‹*restart-heuristics* ⇒ *restart-heuristics*› **where**
‹*schedule-next-pure-lits-stats* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, *phasing*, *reluctant*, *fullyproped*, *lit-st*, *schedule*, *other-fema*, *other-sema*).
(*fast-ema*, *slow-ema*, *restart-info-restart-done res-info*, *wasted*, *phasing*, *reluctant*, *fullyproped*, *lit-st*, *schedule-next-pure-lits-info schedule*, *other-fema*, *other-sema*))›

definition *schedule-next-pure-lits* :: ‹*isasat-restart-heuristics* ⇒ *isasat-restart-heuristics*› **where**
‹*schedule-next-pure-lits* = *Restart-Heuristics o schedule-next-pure-lits-stats o get-restart-heuristics*›

definition *next-pure-lits-schedule-info-stats* :: ‹*restart-heuristics* ⇒ *64 word*› **where**
‹*next-pure-lits-schedule-info-stats* = *next-pure-lits-schedule-info o schedule-info-of-stats*›

definition *next-pure-lits-schedule* :: ‹*isasat-restart-heuristics* ⇒ *64 word*› **where**
‹*next-pure-lits-schedule* = *next-pure-lits-schedule-info-stats o get-restart-heuristics*›

definition *schedule-next-reduce-stats* :: ‹*64 word* ⇒ *restart-heuristics* ⇒ *restart-heuristics*› **where**
‹*schedule-next-reduce-stats delta* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, *phasing*, *reluctant*, *fullyproped*, *lit-st*, *schedule*, *other-fema*, *other-sema*). (*fast-ema*, *slow-ema*, *restart-info-restart-done res-info*, *wasted*, *phasing*, *reluctant*, *fullyproped*, *lit-st*, *schedule-next-reduce-info delta schedule*, *other-fema*, *other-sema*))›

definition *schedule-next-reduce* :: ‹*64 word* ⇒ *isasat-restart-heuristics* ⇒ *isasat-restart-heuristics*› **where**
‹*schedule-next-reduce delta* = *Restart-Heuristics o schedule-next-reduce-stats delta o get-restart-heuristics*›

definition *next-reduce-schedule-info-stats* :: ‹*restart-heuristics* ⇒ *64 word*› **where**
‹*next-reduce-schedule-info-stats* = *next-reduce-schedule-info o schedule-info-of-stats*›

definition *next-reduce-schedule* :: ‹*isasat-restart-heuristics* ⇒ *64 word*› **where**
‹*next-reduce-schedule* = *next-reduce-schedule-info-stats o get-restart-heuristics*›

definition *schedule-next-subsume-stats* :: ‹*64 word* ⇒ *restart-heuristics* ⇒ *restart-heuristics*› **where**
‹*schedule-next-subsume-stats delta* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, *phasing*, *reluctant*, *fullyproped*, *lit-st*, *schedule*, *other-fema*, *other-sema*). (*fast-ema*, *slow-ema*, *restart-info-restart-done res-info*, *wasted*, *phasing*, *reluctant*, *fullyproped*, *lit-st*, *schedule-next-subsume-info delta schedule*, *other-fema*, *other-sema*))›

definition *schedule-next-subsume* :: ‹*64 word* ⇒ *isasat-restart-heuristics* ⇒ *isasat-restart-heuristics*›
where
‹*schedule-next-subsume delta* = *Restart-Heuristics o schedule-next-subsume-stats delta o get-restart-heuristics*›

definition *next-subsume-schedule-info-stats* :: ‹*restart-heuristics* ⇒ *64 word*› **where**
‹*next-subsume-schedule-info-stats* = *next-subsume-schedule-info o schedule-info-of-stats*›

definition *next-subsume-schedule* :: ‹*isasat-restart-heuristics* ⇒ *64 word*› **where**
‹*next-subsume-schedule* = *next-subsume-schedule-info-stats o get-restart-heuristics*›

lemma *heuristic-relI*[*intro*!]:
‹*heuristic-rel* A *heur* ⟹ *heuristic-rel* A (*incr-wasted wast heur*)›
‹*heuristic-rel* A *heur* ⟹ *heuristic-rel* A (*set-zero-wasted heur*)›
‹*heuristic-rel* A *heur* ⟹ *heuristic-rel* A (*incr-restart-phase heur*)›
‹*heuristic-rel* A *heur* ⟹ *heuristic-rel* A (*save-phase-heur L b heur*)›
‹*heuristic-rel* A *heur* ⟹ *heuristic-rel* A (*heuristic-reluctant-tick heur*)›
‹*heuristic-rel* A *heur* ⟹ *heuristic-rel* A (*heuristic-reluctant-enable heur*)›
‹*heuristic-rel* A *heur* ⟹ *heuristic-rel* A (*heuristic-reluctant-untrigger heur*)›
‹*heuristic-rel* A *heur* ⟹ *heuristic-rel* A (*set-fully-propagated-heur heur*)›
‹*heuristic-rel* A *heur* ⟹ *heuristic-rel* A (*unset-fully-propagated-heur heur*)›
‹*heuristic-rel* A *heur* ⟹ *heuristic-rel* A (*mark-added-heur L b heur*)›
‹*heuristic-rel* A *heur* ⟹ *heuristic-rel* A (*swap-emas heur*)›
by (*auto simp*: *heuristic-rel-def save-phase-heur-def phase-save-heur-rel-def phase-saving-def heuristic-reluctant-tick-def heuristic-reluctant-enable-def heuristic-reluctant-untrigger-def set-fully-propagated-heur-def unset-fully-propagated-heur-def set-zero-wasted-def incr-wasted-def incr-restart-phase-def mark-added-heur-def swap-emas-def*)

lemma *heuristic-rel-heuristic-reluctant-triggeredD*:
‹*heuristic-rel* A *heur* ⟹
heuristic-rel A (*fst* (*heuristic-reluctant-triggered heur*))›
by (*clarsimp simp*: *heuristic-reluctant-triggered-def heuristic-rel-def phase-save-heur-rel-def phase-saving-def reluctant-triggered-def heuristic-rel-stats-heuristic-reluctant-triggered-statsD*)

lemma *save-phase-heur-preI*:
‹*heuristic-rel* A *heur* ⟹ *a* ∈# A ⟹ *save-phase-heur-pre a b heur*›

by (*auto simp*: *heuristic-rel-def phase-saving-def save-phase-heur-pre-def save-phase-heur-pre-statsI phase-save-heur-rel-def atms-of-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$*)

Using $a + (1::'a)$ ensures that we do not get stuck with 0.

fun *incr-restart-phase-end-stats* :: ‹*64 word* ⇒ *restart-heuristics* ⇒ *restart-heuristics*› **where**
‹*incr-restart-phase-end-stats end-of-phase* (*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase*, -, *length-phase*), *wasted*) =
(*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase* + *length-phase*, (*length-phase* ∗ *3*) >> *1*), *wasted*)›

definition *incr-restart-phase-end* :: ‹*64 word* ⇒ *isasat-restart-heuristics* ⇒ *isasat-restart-heuristics*›
where
‹*incr-restart-phase-end end-of-phase* = *Restart-Heuristics o* (*incr-restart-phase-end-stats end-of-phase*) *o get-content*›

lemma *heuristic-rel-incr-restartI*[*intro*!]:
‹*heuristic-rel* $\mathcal{A}$ *heur* ⟹ *heuristic-rel* $\mathcal{A}$ (*incr-restart-phase-end lcount heur*)›
by (*auto simp*: *heuristic-rel-def heuristic-rel-stats-def incr-restart-phase-end-def*)

lemma [*intro*!]:
‹*heuristic-rel* $\mathcal{A}$ *heur* ⟹ *heuristic-rel* $\mathcal{A}$ (*heuristic-reluctant-disable heur*)›
‹*heuristic-rel* $\mathcal{A}$ *heur* ⟹ *heuristic-rel* $\mathcal{A}$ (*schedule-next-pure-lits heur*)›
‹*heuristic-rel* $\mathcal{A}$ *heur* ⟹ *heuristic-rel* $\mathcal{A}$ (*heuristic-reluctant-disable heur*)›
‹*heuristic-rel* $\mathcal{A}$ *heur* ⟹ *heuristic-rel* $\mathcal{A}$ (*schedule-next-reduce delta heur*)›
by (*auto simp*: *heuristic-rel-def heuristic-reluctant-disable-def heuristic-rel-stats-def heuristic-reluctant-disable-stats-def next-pure-lits-schedule-def schedule-next-pure-lits-def schedule-next-pure-lits-stats-def next-reduce-schedule-def schedule-next-reduce-def schedule-next-reduce-stats-def*)

lemma [*simp*]:
‹*heuristic-rel* $\mathcal{A}$ *heur* ⟹ *heuristic-rel* $\mathcal{A}$ (*restart-info-restart-done-heur heur*)›
by (*auto simp*: *heuristic-rel-def heuristic-rel-stats-def restart-info-restart-done-heur-def restart-info-restart-done-heur-stats-def*)

5.4.1 Number of clauses

type-synonym *clss-size* = ‹*nat* × *nat* × *nat* × *nat* × *nat*›

definition *clss-size*
:: ‹*'v clauses-l* ⇒ *'v clauses* ⇒ *'v clauses* ⇒ *'v clauses* ⇒ *'v clauses* ⇒*'v clauses* ⇒ *'v clauses* ⇒ *'v clauses* ⇒ *'v clauses* ⇒ *clss-size*›
where
‹*clss-size N NE UE NEk UEk NS US N0 U0* =
(*size* (*learned-clss-lf N*), *size UE*, *size UEk*, *size US*, *size U0*)›

definition *clss-size-lcount* :: ‹*clss-size* ⇒ *nat*› **where**
‹*clss-size-lcount* = (λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*). *lcount*)›

definition *clss-size-lcountUE* :: ‹*clss-size* ⇒ *nat*› **where**
‹*clss-size-lcountUE* = (λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*). *lcountUE*)›

definition *clss-size-lcountUEk* :: ‹*clss-size* ⇒ *nat*› **where**
‹*clss-size-lcountUEk* = (λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*). *lcountUEk*)›

definition *clss-size-lcountUS* :: ‹*clss-size* ⇒ *nat*› **where**
‹*clss-size-lcountUS* = (λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, -). *lcountUS*)›

definition *clss-size-lcountU0* :: ‹*clss-size* ⇒ *nat*› **where**
‹*clss-size-lcountU0* = (λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *lcountU0*). *lcountU0*)›

definition *clss-size-incr-lcount* :: ‹*clss-size* ⇒ *clss-size*› **where**
‹*clss-size-incr-lcount* =
(λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*). (*lcount* + *1*, *lcountUE*, *lcountUEk*, *lcountUS*))›

definition *clss-size-decr-lcount* :: ‹*clss-size* ⇒ *clss-size*› **where**
‹*clss-size-decr-lcount* =
(λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*). (*lcount* − *1*, *lcountUE*, *lcountUEk*, *lcountUS*))›

definition *clss-size-incr-lcountUE* :: ‹*clss-size* ⇒ *clss-size*› **where**
‹*clss-size-incr-lcountUE* =
(λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*). (*lcount*, *lcountUE* + *1*, *lcountUEk*, *lcountUS*))›

definition *clss-size-incr-lcountUEk* :: ‹*clss-size* ⇒ *clss-size*› **where**
‹*clss-size-incr-lcountUEk* =
(λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*). (*lcount*, *lcountUE*, *lcountUEk* + *1*, *lcountUS*))›

definition *clss-size-incr-lcountUS* :: ‹*clss-size* ⇒ *clss-size*› **where**
‹*clss-size-incr-lcountUS* =
(λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *lcountU0*). (*lcount*, *lcountUE*, *lcountUEk*, *lcountUS* + *1*, *lcountU0*))›

definition *clss-size-incr-lcountU0* :: ‹*clss-size* ⇒ *clss-size*› **where**
‹*clss-size-incr-lcountU0* =
(λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *lcountU0*). (*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *lcountU0* + *1*))›

definition *clss-size-resetUS* :: ‹*clss-size* ⇒ *clss-size*› **where**
‹*clss-size-resetUS* =
(λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *lcountU0*). (*lcount*, *lcountUE*, *lcountUEk*, *0*, *lcountU0*))›

definition *clss-size-resetU0* :: ‹*clss-size* ⇒ *clss-size*› **where**
‹*clss-size-resetU0* =
(λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *lcountU0*). (*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *0*))›

definition *clss-size-resetUE* :: ‹*clss-size* ⇒ *clss-size*› **where**
‹*clss-size-resetUE* =
(λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *lcountU0*). (*lcount*, *0*, *lcountUEk*, *lcountUS*, *lcountU0*))›

definition *clss-size-resetUEk* :: ‹*clss-size* ⇒ *clss-size*› **where**
‹*clss-size-resetUEk* =
(λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *lcountU0*). (*lcount*, *lcountUE*, *0*, *lcountUS*, *lcountU0*))›

definition *clss-size-allcount* :: ‹*clss-size* ⇒ *nat*› **where**
‹*clss-size-allcount* =
(λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *lcountU0*). *lcount* + *lcountUE* + *lcountUEk* + *lcountUS* + *lcountU0*)›

abbreviation *clss-size-resetUS0* :: ‹*clss-size* ⇒ -› **where**
‹*clss-size-resetUS0 lcount* ≡ *clss-size-resetUE* (*clss-size-resetUS* (*clss-size-resetU0 lcount*))›

lemma *clss-size-add-simp*[*simp*]:
‹*clss-size N NE* (*add-mset D UE*) *NEk UEk NS US N0 U0* = *clss-size-incr-lcountUE* (*clss-size N NE UE NEk UEk NS US N0 U0*)›

‹clss-size N (add-mset C NE) UE NEk UEk NS US N0 U0 = clss-size N NE UE NEk UEk NS US N0 U0›
‹clss-size N NE UE NEk UEk (add-mset C NS) US N0 U0 = clss-size N NE UE NEk UEk NS US N0 U0›
by (*auto simp*: *clss-size-def ran-m-fmdrop-If clss-size-decr-lcount-def*
clss-size-incr-lcountUE-def size-remove1-mset-If clss-size-resetUS-def)

lemma *clss-size-upd-simp*[*simp*]:
‹*C* ∈# *dom-m N* ⟹ *clss-size* (*N*(*C* ↪ *C'*)) *NE UE NEk UEk NS US N0 U0* = *clss-size N NE UE NEk UEk NS US N0 U0*›
‹*C* ∉# *dom-m N* ⟹ ¬*snd D* ⟹ *clss-size* (*fmupd C D N*) *NE UE NEk UEk NS US N0 U0* = *clss-size-incr-lcount* (*clss-size N NE UE NEk UEk NS US N0 U0*)›
‹*C* ∉# *dom-m N* ⟹ *snd D* ⟹ *clss-size* (*fmupd C D N*) *NE UE NEk UEk NS US N0 U0* = (*clss-size N NE UE NEk UEk NS US N0 U0*)›
by (*auto simp*: *clss-size-def learned-clss-l-fmupd-if clss-size-incr-lcount-def*)

lemma *clss-size-del-simp*[*simp*]:
‹*C* ∈# *dom-m N* ⟹ ¬*irred N C* ⟹ *clss-size* (*fmdrop C N*) *NE UE NEk UEk NS US N0 U0* = *clss-size-decr-lcount* (*clss-size N NE UE NEk UEk NS US N0 U0*)›
‹*C* ∈# *dom-m N* ⟹ *irred N C* ⟹ *clss-size* (*fmdrop C N*) *NE UE NEk UEk NS US N0 U0* = (*clss-size N NE UE NEk UEk NS US N0 U0*)›
by (*auto simp*: *clss-size-def ran-m-fmdrop-If clss-size-decr-lcount-def*
size-remove1-mset-If clss-size-resetUS-def)

lemma *clss-size-lcount-clss-size*[*simp*]:
‹*clss-size-lcount* (*clss-size N NE UE NEk UEk NS US N0 U0*) = *size* (*learned-clss-l N*)›
‹*clss-size-allcount* (*clss-size N NE UE NEk UEk NS US N0 U0*) = *size* (*learned-clss-l N*) + *size UE* + *size UEk* + *size US* + *size U0*›
by (*auto simp*: *clss-size-lcount-def clss-size-def clss-size-allcount-def*)

lemma *clss-size-resetUS-simp*[*simp*]:
‹*clss-size-resetUS* (*clss-size-decr-lcount* (*clss-size baa da ea NEk UEk fa ga ha ia*)) =
clss-size-decr-lcount (*clss-size baa da ea NEk UEk fa* {#} *ha ia*)›
‹*clss-size-resetUS* (*clss-size-incr-lcount* (*clss-size baa da ea NEk UEk fa ga ha ia*)) =
clss-size-incr-lcount (*clss-size baa da ea NEk UEk fa* {#} *ha ia*)›
‹*clss-size-resetUS* (*clss-size-incr-lcountUE* (*clss-size baa da ea NEk UEk fa ga ha ia*)) =
clss-size-incr-lcountUE (*clss-size baa da ea NEk UEk fa* {#} *ha ia*)›
‹*clss-size-resetUS* (*clss-size N NE UE NEk UEk NS US N0 U0*) = (*clss-size N NE UE NEk UEk NS* {#} *N0 U0*)›
‹*clss-size-lcountU0* (*clss-size-resetUS x*) = *clss-size-lcountU0 x*›
by (*auto simp*: *clss-size-resetUS-def clss-size-decr-lcount-def clss-size-def*
clss-size-incr-lcount-def clss-size-incr-lcountUE-def clss-size-lcountU0-def
split: *prod.splits*)

lemma [*simp*]: ‹*clss-size-resetUS* (*clss-size-incr-lcountUE st*) =
clss-size-incr-lcountUE (*clss-size-resetUS st*)›
by (*solves* ‹*cases st*; *auto simp*: *clss-size-incr-lcountUE-def clss-size-resetUS-def*›)+

lemma *clss-size-lcount-simps2*[*simp*]:
‹*clss-size-lcount* (*clss-size-resetUS S*) = *clss-size-lcount S*›
‹*clss-size-lcountUE* (*clss-size-resetUS S*) = *clss-size-lcountUE S*›
‹*clss-size-lcountUS* (*clss-size-resetUS S*) = *0*›

‹*clss-size-lcount* (*clss-size-incr-lcountUE S*) = *clss-size-lcount S*›

‹*clss-size-lcountUE* (*clss-size-incr-lcountUE S*) = *Suc* (*clss-size-lcountUE S*)›
‹*clss-size-lcountUS* (*clss-size-incr-lcountUE S*) = *clss-size-lcountUS S*›

‹*clss-size-lcount* (*clss-size-decr-lcount S*) = *clss-size-lcount S* − *1*›
‹*clss-size-lcountUE* (*clss-size-decr-lcount S*) = *clss-size-lcountUE S*›
‹*clss-size-lcountUS* (*clss-size-decr-lcount S*) = *clss-size-lcountUS S*›

‹*clss-size-incr-lcountUE* (*clss-size-decr-lcount S*) =
clss-size-decr-lcount (*clss-size-incr-lcountUE S*)›
‹*clss-size-resetUS* (*clss-size-decr-lcount S*) =
clss-size-decr-lcount (*clss-size-resetUS S*)›
‹*clss-size-resetUS* (*clss-size-incr-lcountUE S*) = *clss-size-incr-lcountUE* (*clss-size-resetUS S*)›
by (*solves* ‹*cases S*; *auto simp*: *clss-size-lcount-def clss-size-resetUS-def*
clss-size-lcountUE-def clss-size-lcountUS-def
clss-size-incr-lcountUE-def clss-size-decr-lcount-def›)+

lemma [*simp*]:
‹*clss-size-lcountU0* (*clss-size-decr-lcount S*) = *clss-size-lcountU0 S*›
‹*clss-size-lcountU0* (*clss-size-incr-lcountUE S*) = *clss-size-lcountU0 S*›
‹*clss-size-lcountU0* (*clss-size-incr-lcountUS S*) = *clss-size-lcountU0 S*›
‹*clss-size-lcountU0* (*clss-size-incr-lcountU0 S*) = *clss-size-lcountU0 S* + *1*›
by (*auto simp*: *clss-size-lcountU0-def clss-size-decr-lcount-def clss-size-incr-lcountUE-def*
clss-size-incr-lcountUS-def clss-size-incr-lcountU0-def
split: *prod.splits*)

lemma [*simp*]:
‹*clss-size-lcount* (*clss-size-incr-lcountUEk c*) = *clss-size-lcount c*›
‹*clss-size-lcountUE* (*clss-size-incr-lcountUEk c*) = *clss-size-lcountUE c*›
‹*clss-size-lcountUEk* (*clss-size-incr-lcountUEk c*) = *clss-size-lcountUEk c*+*1*›
‹*clss-size-lcountU0* (*clss-size-incr-lcountUEk c*) = *clss-size-lcountU0 c*›
‹*clss-size-lcountUS* (*clss-size-incr-lcountUEk c*) = *clss-size-lcountUS c*›
by (*auto simp*: *clss-size-lcountUE-def clss-size-lcount-def clss-size-incr-lcountUEk-def*
clss-size-lcountUEk-def clss-size-lcountU0-def clss-size-lcountUS-def
split: *prod.splits*)

lemma *clss-size-simps3*[*simp*]:
‹*clss-size-lcountUE* (*clss-size baa da ea NEk UEk fa x N0 U0*) = *size ea*›
‹*clss-size-lcountUEk* (*clss-size baa da ea NEk UEk fa x N0 U0*) = *size UEk*›
‹*clss-size-lcountUS* (*clss-size baa da ea NEk UEk fa x N0 U0*) = *size x*›
‹*clss-size-lcountU0* (*clss-size baa da ea NEk UEk fa x N0 U0*) = *size U0*›
by (*auto simp*: *clss-size-lcountUE-def clss-size-lcountUS-def clss-size-lcountU0-def clss-size-def*
clss-size-lcountUEk-def)

lemma *clss-size-lcount-incr-lcount-simps*[*simp*]:
‹*clss-size-lcount* (*clss-size-incr-lcount S*) = *Suc* (*clss-size-lcount S*)›
‹*clss-size-lcountUE* (*clss-size-incr-lcount S*) = (*clss-size-lcountUE S*)›
‹*clss-size-lcountUEk* (*clss-size-incr-lcount S*) = (*clss-size-lcountUEk S*)›
‹*clss-size-lcountUS* (*clss-size-incr-lcount S*) = (*clss-size-lcountUS S*)›
‹*clss-size-lcountU0* (*clss-size-incr-lcount* (*S*)) = *clss-size-lcountU0* ((*S*))›
by (*cases S*; *auto simp*: *clss-size-incr-lcount-def*
clss-size-lcount-def clss-size-def clss-size-lcountUEk-def
clss-size-lcountUE-def clss-size-lcountUS-def clss-size-lcountU0-def; *fail*)+

lemma [*simp*]:

‹*clss-size-lcount* (*clss-size-resetU0 c*) = *clss-size-lcount c*›
‹*clss-size-lcount* (*clss-size-resetUE c*) = *clss-size-lcount c*›
‹*clss-size-lcount* (*clss-size-resetUEk c*) = *clss-size-lcount c*›
‹*clss-size-lcountUE* (*clss-size-resetU0 c*) = *clss-size-lcountUE c*›
‹*clss-size-lcountUE* (*clss-size-resetUEk c*) = *clss-size-lcountUE c*›
‹*clss-size-lcountU0* (*clss-size-resetUE c*) = *clss-size-lcountU0 c*›
‹*clss-size-lcountU0* (*clss-size-resetUEk c*) = *clss-size-lcountU0 c*›
‹*clss-size-lcountU0* (*clss-size-resetU0 c*) = *0*›
‹*clss-size-lcountU0* (*clss-size-decr-lcount c*) = *clss-size-lcountU0 c*›
‹*clss-size-lcountUEk* (*clss-size-resetUE c*) = *clss-size-lcountUEk c*›
‹*clss-size-lcountUEk* (*clss-size-resetUS c*) = *clss-size-lcountUEk c*›
‹*clss-size-lcountUEk* (*clss-size-resetU0 c*) = *clss-size-lcountUEk c*›
‹*clss-size-lcountUEk* (*clss-size-resetUEk c*) = *0*›
‹*clss-size-lcountUEk* (*clss-size-decr-lcount c*) = *clss-size-lcountUEk c*›
‹*clss-size-lcountUE* (*clss-size-resetUE c*) = *0*›
‹*clss-size-lcountUS* (*clss-size-resetUE c*) = *clss-size-lcountUS c*›
‹*clss-size-lcountUS* (*clss-size-resetUEk c*) = *clss-size-lcountUS c*›
by (*auto simp*: *clss-size-resetU0-def clss-size-lcount-def clss-size-lcountU0-def*
clss-size-lcountUS-def clss-size-decr-lcount-def
clss-size-resetUE-def clss-size-resetUEk-def clss-size-lcountUE-def clss-size-lcountUEk-def
clss-size-resetUS-def split: *prod.splits*)

definition *print-literal-of-trail* **where**
‹*print-literal-of-trail* - = *RETURN* ()›

definition *print-trail* **where**
‹*print-trail* = (λ(*M*, -). *do* {
i ← *WHILE*$_T$(λ*i*. *i* < *length M*)
(λ*i*. *do* {
ASSERT(*i* < *length M*);
print-literal-of-trail (*M*!*i*);
RETURN (*i*+*1*)})
0;
print-literal-of-trail ((*0*::*nat*));
RETURN ()
})›

definition *print-trail2* **where**
‹*print-trail2* = (λ(*M*, -). *RETURN* ())›

lemma *print-trail-print-trail2*:
‹(*M*,*M'*)∈*Id* ⟹ *print-trail M* ≤ ⇓ *Id* (*print-trail2 M'*)›
unfolding *print-trail-def print-trail2-def*
apply (*refine-vcg WHILET-rule*[**where**
R = ‹*measure* (λ*i*. *Suc* (*length* (*fst M*)) − *i*)› **and**
I = ‹λ*i*. *i* ≤ *length* (*fst M*)›])
subgoal by *auto*
subgoal by *auto*
subgoal unfolding *print-literal-of-trail-def* **by** *auto*
subgoal unfolding *print-literal-of-trail-def* **by** *auto*
done

lemma *print-trail-print-trail2-rel*:
‹(*print-trail*, *print-trail2*) ∈ *Id* →$_f$ ⟨*unit-rel*⟩*nres-rel*›
using *print-trail-print-trail2* **by** (*fastforce intro*: *frefI nres-relI*)

definition *clss-size-corr* :: ‹- ⇒ - ⇒ - ⇒ - ⇒- ⇒ - ⇒- ⇒ - ⇒- ⇒ *clss-size* ⇒ *bool*› **where**
‹*clss-size-corr N NE UE NEk UEk NS US N0 U0 c* ⟷ *c* = *clss-size N NE UE NEk UEk NS US N0 U0*›

There is no equivalence because of rounding errors. However, we do not care about that in the proofs and we are always safe in IsaSAT.

However, the intro rule are still too dangerous and make it hard to recognize the original goal. Therefore, they are not marked as safe.

lemma
clss-size-corr-intro[*intro*!]:
‹*C* ∈# *dom-m N* ⟹ ¬*irred N C* ⟹*clss-size-corr N NE UE NEk UEk NS US N0 U0 c* ⟹ *clss-size-corr* (*fmdrop C N*) *NE UE NEk UEk NS US N0 U0* (*clss-size-decr-lcount c*)›
‹*C* ∉# *dom-m N* ⟹ ¬*b* ⟹*clss-size-corr N NE UE NEk UEk NS US N0 U0 c* ⟹ *clss-size-corr* (*fmupd C* (*D*, *b*) *N*) *NE UE NEk UEk NS US N0 U0* (*clss-size-incr-lcount c*)›
‹*clss-size-corr N NE UE NEk UEk NS US N0 U0 c* ⟹ *clss-size-corr N NE UE NEk* (*add-mset E UEk*) *NS US N0 U0* (*clss-size-incr-lcountUEk c*)›
‹*clss-size-corr N NE UE NEk UEk NS US N0 U0 c* ⟹ *clss-size-corr N NE* (*add-mset E UE*) *NEk UEk NS US N0 U0* (*clss-size-incr-lcountUE c*)›
‹*clss-size-corr N NE UE NEk UEk NS US N0 U0 c* ⟹ *clss-size-corr N NE UE NEk UEk NS* (*add-mset E US*) *N0 U0* (*clss-size-incr-lcountUS c*)›
‹*clss-size-corr N NE UE NEk UEk NS US N0 U0 c* ⟹ *clss-size-corr N NE UE NEk UEk NS US N0* (*add-mset E U0*) (*clss-size-incr-lcountU0 c*)›
‹*clss-size-corr N NE UE NEk UEk NS US N0 U0* (*clss-size N NE UE NEk UEk NS US N0 U0*)›
and
clss-size-corr-simp[*simp*]:
‹*clss-size-corr N NE UE NEk UEk NS US N0 U0 c* ⟹ *clss-size-corr N NE UE NEk UEk NS* {#} *N0 U0* (*clss-size-resetUS c*)›
‹*clss-size-corr N NE UE NEk UEk NS US N0 U0 c* ⟹ *clss-size-corr N NE UE NEk* {#} *NS US N0 U0* (*clss-size-resetUEk c*)›
‹*clss-size-corr N NE UE NEk UEk NS US N0 U0 c* ⟹ *clss-size-corr N NE UE NEk UEk NS US N0* {#} (*clss-size-resetU0 c*)›
‹*C* ∉# *dom-m N* ⟹ *b* ⟹ *clss-size-corr* (*fmupd C* (*D*, *b*) *N*) *NE UE NEk UEk NS US N0 U0 c* ⟷
clss-size-corr N NE UE NEk UEk NS US N0 U0 c›
‹*C* ∈# *dom-m N* ⟹ *irred N C* ⟹ *clss-size-corr* (*fmdrop C N*) *NE UE NEk UEk NS US N0 U0 c* = *clss-size-corr N NE UE NEk UEk NS US N0 U0 c*›
‹*C* ∈# *dom-m N* ⟹ *clss-size-corr* (*N*(*C* ↪ *swap* (*N* ∝ *C*) *i j*)) *NE UE NEk UEk NS US N0 U0 c* = *clss-size-corr N NE UE NEk UEk NS US N0 U0 c*›
‹*clss-size-corr N NE UE* (*add-mset E NEk*) *UEk NS US N0 U0 c* = *clss-size-corr N NE UE NEk UEk NS US N0 U0 c*›
‹*clss-size-corr N* (*add-mset E NE*) *UE NEk UEk NS US N0 U0 c* = *clss-size-corr N NE UE NEk UEk NS US N0 U0 c*›
‹*clss-size-corr N NE UE NEk UEk* (*add-mset E NS*) *US N0 U0 c* = *clss-size-corr N NE UE NEk UEk NS US N0 U0 c*›
‹*clss-size-corr N NE UE NEk UEk NS US* (*add-mset E N0*) *U0 c* = *clss-size-corr N NE UE NEk UEk NS US N0 U0 c*›
‹*clss-size-corr N NE UE NEk UEk NS US N0 U0 lcount* ⟹ *clss-size-lcount lcount* = *size* (*learned-clss-lf N*)›
by (*auto simp*: *clss-size-def ran-m-fmdrop-If clss-size-decr-lcount-def learned-clss-l-fmupd-if clss-size-incr-lcount-def clss-size-incr-lcountUS-def clss-size-incr-lcountU0-def clss-size-incr-lcountUEk-def clss-size-incr-lcountUE-def clss-size-lcount-def clss-size-resetUEk-def clss-size-resetU0-def size-remove1-mset-If clss-size-resetUS-def clss-size-corr-def*; *fail*)+

This version of the counter is incomplete. It is however useful because we do not need to care about some of the counts during restarts. In particular, it avoids taking care of overflows.

definition *clss-size-corr-restart* :: ‹'v clauses-l ⇒ 'v clauses ⇒ 'v clauses ⇒ 'v clauses ⇒ 'v clauses ⇒ 'v clauses ⇒ 'v clauses ⇒ 'v clauses ⇒ clss-size ⇒ bool› **where**
‹*clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c* ⟷ (∃ *UE US U0*. *c* = *clss-size N NE UE NEk UEk NS US N0 U0*)›

lemma *clss-size-corr-restart-clss-size-corr*:
‹*clss-size-corr N NE UE NEk UEk NS US N0 U0 c* ⟹ *clss-size-corr-restart N NE UE' NEk UEk NS US' N0 U0' c*›
‹*clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c* ⟹ *clss-size-corr N NE* {#} *NEk UEk NS* {#} *N0* {#} (*clss-size-resetUS0 c*)›
by (*auto simp*: *clss-size-corr-def clss-size-corr-restart-def clss-size-resetUS-def clss-size-resetU0-def clss-size-def clss-size-resetUE-def*)

lemma
clss-size-corr-restart-intro[*intro*]:
‹*C* ∈# *dom-m N* ⟹ ¬*irred N C* ⟹*clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c* ⟹ *clss-size-corr-restart* (*fmdrop C N*) *NE* {#} *NEk UEk NS* {#} *N0* {#} (*clss-size-decr-lcount c*)›
‹*C* ∉# *dom-m N* ⟹ ¬*b* ⟹*clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c* ⟹ *clss-size-corr-restart* (*fmupd C* (*D*, *b*) *N*) *NE* {#} *NEk UEk NS* {#} *N0* {#} (*clss-size-incr-lcount c*)›
‹*clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c* ⟹ *clss-size-corr-restart N NE* {#} *NEk* (*add-mset E UEk*) *NS* {#} *N0* {#} (*clss-size-incr-lcountUEk c*)›
‹*clss-size-corr-restart N NE UE NEk UEk NS US N0 U0* (*clss-size N NE* {#} *NEk UEk NS* {#} *N0* {#})›
‹*clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c* ⟹ *clss-size-corr-restart N NE* {#} *NEk UEk NS* {#} *N0* {#} *c*›

‹*clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c* ⟹ *clss-size-corr-restart N NE* {#} *NEk UEk NS US N0 U0* (*c*)›
‹*clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c* ⟹ *clss-size-corr-restart N NE UE NEk UEk NS US N0* {#} (*c*)›
‹*clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c* ⟹ *clss-size-corr-restart N NE UE NEk UEk NS* {#} *N0 U0* (*c*)›
‹*clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c* ⟹ *clss-size-corr-restart N NE UE NEk UEk NS US N0* {#} (*c*)›
and
clss-size-corr-restart-simp[*simp*]:
‹*NO-MATCH* {#} *UE* ⟹ *clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c* ⟷ *clss-size-corr-restart N NE* {#} *NEk UEk NS US N0 U0* (*c*)›
‹*NO-MATCH* {#} *U0* ⟹ *clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c* ⟷ *clss-size-corr-restart N NE UE NEk UEk NS US N0* {#} (*c*)›
‹*NO-MATCH* {#} *US* ⟹ *clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c* ⟷ *clss-size-corr-restart N NE UE NEk UEk NS* {#} *N0 U0* (*c*)›
‹*NO-MATCH* {#} *UE* ⟹ *clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c* ⟷ *clss-size-corr-restart N NE UE NEk UEk NS US N0* {#} (*c*)›

‹*clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c* ⟹ *clss-size-corr-restart N NE UE NEk* {#} *NS US N0 U0* (*clss-size-resetUEk c*)›
‹*clss-size-corr-restart N NE UE NEk UEk NS* (*add-mset E US*) *N0 U0* (*c*) ⟷ *clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c*›
‹*clss-size-corr-restart N NE UE NEk UEk NS US N0* (*add-mset E U0*) (*c*) ⟷ *clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c*›
‹*C* ∉# *dom-m N* ⟹ *b* ⟹ *clss-size-corr-restart* (*fmupd C* (*D*, *b*) *N*) *NE UE NEk UEk NS US N0 U0 c* ⟷
clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c›
‹*C* ∈# *dom-m N* ⟹ *irred N C* ⟹ *clss-size-corr-restart* (*fmdrop C N*) *NE UE NEk UEk NS US*

N0 U0 c = clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c›
‹*C ∈# dom-m N ⟹ clss-size-corr-restart* (*N*(*C ↪ swap* (*N ∝ C*) *i j*)) *NE UE NEk UEk NS US N0 U0 c = clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c*›
‹*clss-size-corr-restart N* (*add-mset E NE*) *UE NEk UEk NS US N0 U0 c = clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c*›
‹*clss-size-corr-restart N NE UE* (*add-mset E NEk*) *UEk NS US N0 U0 c = clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c*›
‹*clss-size-corr-restart N NE UE NEk UEk* (*add-mset E NS*) *US N0 U0 c = clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c*›
‹*clss-size-corr-restart N NE UE NEk UEk NS US* (*add-mset E N0*) *U0 c = clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c*›
by (*auto simp*: *clss-size-def ran-m-fmdrop-If clss-size-decr-lcount-def learned-clss-l-fmupd-if clss-size-incr-lcount-def clss-size-incr-lcountUS-def clss-size-incr-lcountU0-def clss-size-incr-lcountUEk-def clss-size-incr-lcountUE-def clss-size-lcount-def clss-size-resetUEk-def clss-size-resetU0-def clss-size-resetUE-def size-remove1-mset-If clss-size-resetUS-def clss-size-corr-restart-def*; *fail*)+

The following lemmas produce loops, but usually only in the next file (!). Hence, we do not activate them by default as simp rules.

lemma *clss-size-corr-restart-rew*:
‹*clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 lcount ⟹ clss-size-lcount lcount = size* (*learned-clss-lf N*)›
by (*auto simp*: *clss-size-def ran-m-fmdrop-If clss-size-decr-lcount-def learned-clss-l-fmupd-if clss-size-incr-lcount-def clss-size-incr-lcountUS-def clss-size-incr-lcountU0-def clss-size-incr-lcountUEk-def clss-size-incr-lcountUE-def clss-size-lcount-def clss-size-resetUEk-def clss-size-resetU0-def clss-size-resetUE-def size-remove1-mset-If clss-size-resetUS-def clss-size-corr-restart-def*; *fail*)+

lemma *clss-size-corr-restart-simp3*:
‹*clss-size-corr-restart N NE UE NEk* (*add-mset E UEk*) *NS US N0 U0* (*clss-size-incr-lcountUEk c*) ⟷
clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c›
by (*auto simp*: *clss-size-corr-restart-def clss-size-incr-lcountUEk-def clss-size-def split*: *prod.splits*)

5.4.2 Lifting to heuristic level

definition *get-next-phase-heur-pre-stats* :: ‹*bool ⇒ nat ⇒ restart-heuristics ⇒ bool*› **where**
‹*get-next-phase-heur-pre-stats* = (*λb L* (-, -, -, -, *rephase*, -).
get-next-phase-pre b L rephase)›

definition *get-next-phase-heur-stats* :: ‹*bool ⇒ nat ⇒ restart-heuristics ⇒ bool nres*› **where**
‹*get-next-phase-heur-stats* = (*λb L* (-, -, -, -, *rephase*, -).
get-next-phase-stats b L rephase)›

definition *get-next-phase-heur* :: ‹*bool ⇒ nat ⇒ isasat-restart-heuristics ⇒ bool nres*› **where**
‹*get-next-phase-heur* = (*λb L heur*.
let heur = *get-restart-heuristics heur in*
get-next-phase-heur-stats b L heur)›

definition *end-of-restart-phase-stats* :: ‹*restart-heuristics ⇒ 64 word*› **where**
‹*end-of-restart-phase-stats* = (*λ*(-, -, (*restart-phase*,- ,- , *end-of-phase*, -), -).
end-of-phase)›

definition *end-of-restart-phase* :: ‹*isasat-restart-heuristics ⇒ 64 word*› **where**
‹*end-of-restart-phase* = *end-of-restart-phase-stats o get-content*›

end
theory *IsaSAT-Options-LLVM*
imports *IsaSAT-Options IsaSAT-Literals-LLVM*
begin

type-synonym *opts-assn* = ‹*1 word* × *1 word* × *1 word* × *64 word* × *64 word* × *64 word* × *3 word* × *64 word*
× *64 word* × *64 word* × *1 word*›

definition *opts-rel-assn* :: ‹*opts-ref* ⇒ - ⇒ *assn*› **where**
‹*opts-rel-assn* = *bool1-assn* ×$_a$ *bool1-assn* ×$_a$ *bool1-assn* ×$_a$ *word-assn* ×$_a$ *word-assn*
×$_a$ *snat-assn*′ *TYPE*(*64*) ×$_a$ *word-assn*′ *TYPE*(*3*) ×$_a$ *word64-assn* ×$_a$ *word64-assn* ×$_a$ *word64-assn*
×$_a$ *bool1-assn*›

sepref-def *opts-rel-restart-code*
is ‹*RETURN o opts-rel-restart*›
:: ‹*opts-rel-assn*^{k} →$_a$ *bool1-assn*›
unfolding *opts-rel-alt-defs opts-rel-assn-def*
by *sepref*

sepref-def *opts-rel-reduce-code*
is ‹*RETURN o opts-rel-reduce*›
:: ‹*opts-rel-assn*^{k} →$_a$ *bool1-assn*›
unfolding *opts-rel-alt-defs opts-rel-assn-def*
by *sepref*

sepref-def *opts-rel-unbounded-mode-code*
is ‹*RETURN o opts-rel-unbounded-mode*›
:: ‹*opts-rel-assn*^{k} →$_a$ *bool1-assn*›
unfolding *opts-rel-alt-defs opts-rel-assn-def*
by *sepref*

sepref-def *opts-rel-miminum-between-restart-code*
is ‹*RETURN o opts-rel-miminum-between-restart*›
:: ‹*opts-rel-assn*^{k} →$_a$ *word-assn*›
unfolding *opts-rel-alt-defs opts-rel-assn-def*
by *sepref*

sepref-def *opts-rel-restart-coeff1-code*
is ‹*RETURN o opts-rel-restart-coeff1*›
:: ‹*opts-rel-assn*^{k} →$_a$ *word-assn*›
unfolding *opts-rel-alt-defs opts-rel-assn-def*
by *sepref*

sepref-def *opts-rel-restart-coeff2-code*
is ‹*RETURN o opts-rel-restart-coeff2*›
:: ‹*opts-rel-assn*^{k} →$_a$ *snat-assn*′ *TYPE*(*64*)›
unfolding *opts-rel-alt-defs opts-rel-assn-def*
by *sepref*

sepref-def *opts-rel-target-code*
is ‹*RETURN o opts-rel-target*›
:: ‹*opts-rel-assn*^{k} →$_a$ *word-assn*′ *TYPE*(*3*)›
unfolding *opts-rel-alt-defs opts-rel-assn-def*

by *sepref*

sepref-def *opts-rel-fema-code*
is ‹*RETURN o opts-rel-fema*›
:: ‹*opts-rel-assn*^{k} →$_a$ *word64-assn*›
unfolding *opts-rel-alt-defs opts-rel-assn-def*
by *sepref*

sepref-def *opts-rel-sema-code*
is ‹*RETURN o opts-rel-sema*›
:: ‹*opts-rel-assn*^{k} →$_a$ *word64-assn*›
unfolding *opts-rel-alt-defs opts-rel-assn-def*
by *sepref*

sepref-def *opts-rel-GC-units-lim-code*
is ‹*RETURN o opts-rel-GC-units-lim*›
:: ‹*opts-rel-assn*^{k} →$_a$ *word64-assn*›
unfolding *opts-rel-alt-defs opts-rel-assn-def*
by *sepref*

sepref-def *opts-rel-subsumption-code*
is ‹*RETURN o opts-rel-subsumption*›
:: ‹*opts-rel-assn*^{k} →$_a$ *bool1-assn*›
unfolding *opts-rel-alt-defs opts-rel-assn-def*
by *sepref*

definition *opts-assn* :: ‹*opts* ⇒ *opts-assn* ⇒ *assn*› **where**
‹*opts-assn* = *hr-comp opts-rel-assn opts-rel*›

lemmas *opts-refine*[*sepref-fr-rules*] =
opts-rel-restart-code.refine[*FCOMP opts-rel-restart*, *unfolded opts-assn-def*[*symmetric*]]
opts-rel-reduce-code.refine[*FCOMP opts-rel-reduce*, *unfolded opts-assn-def*[*symmetric*]]
opts-rel-unbounded-mode-code.refine[*FCOMP opts-rel-unbounded-mode*, *unfolded opts-assn-def*[*symmetric*]]
opts-rel-miminum-between-restart-code.refine[*FCOMP opts-rel-miminum-between-restart*, *unfolded opts-assn-def*[*symme*
opts-rel-restart-coeff1-code.refine[*FCOMP opts-rel-restart-coeff1*, *unfolded opts-assn-def*[*symmetric*]]
opts-rel-restart-coeff2-code.refine[*FCOMP opts-rel-restart-coeff2*, *unfolded opts-assn-def*[*symmetric*]]
opts-rel-target-code.refine[*FCOMP opts-rel-target*, *unfolded opts-assn-def*[*symmetric*]]
opts-rel-fema-code.refine[*FCOMP opts-rel-fema*, *unfolded opts-assn-def*[*symmetric*]]
opts-rel-sema-code.refine[*FCOMP opts-rel-sema*, *unfolded opts-assn-def*[*symmetric*]]
opts-rel-GC-units-lim-code.refine[*FCOMP opts-GC-units-lim*, *unfolded opts-assn-def*[*symmetric*]]
opts-rel-subsumption-code.refine[*FCOMP opts-subsumption*, *unfolded opts-assn-def*[*symmetric*]]

sepref-register *opts-restart opts-reduce opts-minimum-between-restart opts-restart-coeff1*
opts-restart-coeff2 opts-target opts-fema opts-sema opts-subsumption

lemma *opts-assn-assn-pure*[*safe-constraint-rules*]: ‹*CONSTRAINT is-pure opts-assn*›
unfolding *opts-assn-def opts-rel-assn-def*
by *solve-constraint*

lemmas [*sepref-frame-free-rules*] = *mk-free-is-pure*[*OF opts-assn-assn-pure*[*unfolded CONSTRAINT-def*]]

definition *default-opts* :: *opts* **where**
‹*default-opts* = *IsaOptions True True True 50 11 4 1 128849010 429450 15 True*›

definition *default-opts2* :: *opts-ref* **where**
‹*default-opts2* = (*True*, *True*, *True*, *50*, *11*, *4*, *2*, *128849010*, *429450*, *15*, *True*)›

definition *IsaOptions-rel*
:: ‹*bool* ⇒ *bool* ⇒ *bool* ⇒ *64 word* ⇒ *64 word* ⇒ *nat* ⇒ *opts-target* ⇒ *64 word* ⇒ *64 word* ⇒ *64 word* ⇒ *bool* ⇒ *opts-ref*› **where**
‹*IsaOptions-rel a b c d e f g h i j k* = (*a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*, *i*, *j*, *k*)›

lemma *IsaOptions-rel*:
‹(*uncurry10* (*RETURN* o_{11} *IsaOptions-rel*), *uncurry10* (*RETURN* o_{11} *IsaOptions*)) ∈
bool-rel ×$_f$ *bool-rel* ×$_f$ *bool-rel* ×$_f$ *word-rel* ×$_f$ *word-rel* ×$_f$ *nat-rel* ×$_f$ *word-rel* ×$_f$ *word-rel* ×$_f$ *word-rel* ×$_f$ *word-rel* ×$_f$ *bool-rel* →
⟨*opts-rel*⟩*nres-rel*›
by (*auto intro*!: *frefI nres-relI simp*: *opts-rel-def IsaOptions-rel-def*)

sepref-def *IsaOptions-rel-impl*
is ‹*uncurry10* (*RETURN* o_{11} *IsaOptions-rel*)›
:: ‹*bool1-assn*^{k} *∗*$_a$ *bool1-assn*^{k} *∗*$_a$ *bool1-assn*^{k} *∗*$_a$ *word-assn*^{k} *∗*$_a$ *word-assn*^{k} *∗*$_a$ (*snat-assn*′ (*TYPE*(*64*)))^k *∗*$_a$ (*word-assn*′ (*TYPE*(*3*)))^k *∗*$_a$ *word-assn*^{k} *∗*$_a$ *word-assn*^{k} *∗*$_a$ *word-assn*^{k} *∗*$_a$ *bool1-assn*^{k} →$_a$ *opts-rel-assn*›
unfolding *IsaOptions-rel-def opts-rel-assn-def*
by *sepref*

sepref-register *IsaOptions*

lemmas [*sepref-fr-rules*] =
IsaOptions-rel-impl.refine[*FCOMP IsaOptions-rel*, *unfolded opts-assn-def*[*symmetric*]]

lemma [*sepref-import-param*]:
‹(*0*, *TARGET-NEVER*) ∈ *word-rel*›
‹(*1*, *TARGET-STABLE-ONLY*) ∈ *word-rel*›
‹(*2*, *TARGET-ALWAYS*) ∈ *word-rel*›
by (*auto simp*: *TARGET-NEVER-def TARGET-ALWAYS-def TARGET-STABLE-ONLY-def*)

experiment begin

export-llvm
opts-rel-restart-code
opts-rel-reduce-code
end

end
theory *IsaSAT-EMA-LLVM*
imports *IsaSAT-EMA IsaSAT-Literals-LLVM*
begin

abbreviation *ema-rel* :: ‹(*ema*×*ema*) *set*› **where**
‹*ema-rel* ≡ *word64-rel* ×$_r$ *word64-rel* ×$_r$ *word64-rel* ×$_r$ *word64-rel* ×$_r$ *word64-rel*›

abbreviation *ema-assn* :: ‹*ema* ⇒ *ema* ⇒ *assn*› **where**
‹*ema-assn* ≡ *word64-assn* ×$_a$ *word64-assn* ×$_a$ *word64-assn* ×$_a$ *word64-assn* ×$_a$ *word64-assn*›

lemma [*sepref-import-param*]:
‹(*ema-get-value*, *ema-get-value*) ∈ *ema-rel* → *word64-rel*›
‹(*ema-bitshifting*,*ema-bitshifting*) ∈ *word64-rel*›
‹(*ema-reinit*,*ema-reinit*) ∈ *ema-rel* → *ema-rel*›

‹(*ema-init*,*ema-init*) ∈ *word-rel* → *ema-rel*›
by *auto*

sepref-register *EMA-FIXPOINT-SIZE ema-bitshifting*
sepref-def *EMA-FIXPOINT-SIZE-impl*
is ‹*uncurry0* (*RETURN EMA-FIXPOINT-SIZE*)›
:: ‹*unit-assn*^{k} →$_a$ *uint64-nat-assn*›
unfolding *EMA-FIXPOINT-SIZE-def*
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *EMA*[*simp*]:
‹*EMA-FIXPOINT-SIZE* < *64*›
‹*EMA-MULT-SHIFT* < *64*›
‹*EMA-FIXPOINT-SIZE* − *EMA-MULT-SHIFT* < *64*›
‹*EMA-MULT-SHIFT* ≤ *EMA-FIXPOINT-SIZE*›
‹*EMA-FIXPOINT-SIZE* − *32* < *64*›
‹*EMA-FIXPOINT-SIZE* ≥ *32*›
by (*auto simp*: *EMA-FIXPOINT-SIZE-def EMA-MULT-SHIFT-def*)

sepref-def *ema-bitshifting-impl*
is ‹*uncurry0* (*RETURN ema-bitshifting*)›
:: ‹*unit-assn*^{k} →$_a$ *word64-assn*›
unfolding *ema-bitshifting-def*
by *sepref*

lemma *ema-reinit-inline*[*llvm-inline*]:
ema-reinit = (λ(*value*, α, β, *wait*, *period*).
(*value*, α, *ema-bitshifting*, *1*::- *word*, *0*:: - *word*))
by (*auto simp*: *ema-bitshifting-def intro*!: *ext*)

sepref-def *EMA-MULT-SHIFT-impl*
is ‹*uncurry0* (*RETURN EMA-MULT-SHIFT*)›
:: ‹*unit-assn*^{k} →$_a$ *uint64-nat-assn*›
unfolding *EMA-MULT-SHIFT-def*
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemmas [*llvm-inline*] = *ema-init-def*

sepref-def *ema-update-impl* **is** ‹*uncurry* (*RETURN oo ema-update*)›
:: ‹*uint32-nat-assn*^{k} ×$_a$ *ema-assn*^{k} →$_a$ *ema-assn*›
unfolding *ema-update-def Let-def*[*of - − 1*]
apply (*rewrite at* ‹*let - = of-nat* ⌑ × - *in* -› *annot-unat-unat-upcast*[**where** *'l* = *64*])
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-def *ema-init-impl*
is ‹*RETURN o ema-init*›
:: ‹*word64-assn*^{k} →$_a$ *ema-assn*›
unfolding *ema-init-def*
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
by *sepref*

end

theory *IsaSAT-Rephase-LLVM*
imports *IsaSAT-Rephase IsaSAT-Literals-LLVM*
begin

hide-const (**open**) *NEMonad.ASSERT NEMonad.RETURN*

type-synonym *phase-saver-assn* = ‹*1 word larray64*›
abbreviation *phase-saver-assn* :: ‹*phase-saver* ⇒ *phase-saver-assn* ⇒ *assn*› **where**
‹*phase-saver-assn* ≡ *larray64-assn bool1-assn*›

type-synonym *phase-saver'-assn* = ‹*1 word ptr*›

abbreviation *phase-saver'-assn* :: ‹*phase-saver* ⇒ *phase-saver'-assn* ⇒ *assn*› **where**
‹*phase-saver'-assn* ≡ *array-assn bool1-assn*›

definition *phase-heur-assn* :: ‹*phase-save-heur* ⇒ -› **where**
‹*phase-heur-assn* ≡ *phase-saver-assn* ×$_a$ *word64-assn* ×$_a$ *phase-saver'-assn* ×$_a$ *word64-assn* ×$_a$
phase-saver'-assn ×$_a$ *word64-assn* ×$_a$ *word64-assn* ×$_a$ *word64-assn*›

schematic-goal *mk-free-lookup-clause-rel-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE phase-heur-assn ?fr*›
unfolding *phase-heur-assn-def*
by *synthesize-free+*

sepref-def *rephase-random-impl*
is ‹*uncurry rephase-random*›
:: ‹*word-assn*^{k} *$*_a$ *phase-saver-assn*^{d} →$_a$ *phase-saver-assn*›
supply [[*goals-limit=1*]]
unfolding *rephase-random-def*
while-eq-nfoldli[*symmetric*]
apply (*subst while-upt-while-direct*, *simp*)
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *rephase-init-impl*
is ‹*uncurry rephase-init*›
:: ‹*bool1-assn*^{k} *$*_a$ *phase-saver-assn*^{d} →$_a$ *phase-saver-assn*›
unfolding *rephase-init-def*
while-eq-nfoldli[*symmetric*]
apply (*subst while-upt-while-direct*, *simp*)
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *copy-phase-impl*
is ‹*uncurry copy-phase*›
:: ‹*phase-saver-assn*^{k} *$*_a$ *phase-saver'-assn*^{d} →$_a$ *phase-saver'-assn*›
unfolding *copy-phase-alt-def*
while-eq-nfoldli[*symmetric*]
apply (*subst while-upt-while-direct*, *simp*)
unfolding *simp-thms*(*21*) — remove *a* ∧ *True* from condition
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

definition *copy-phase2* **where**
‹*copy-phase2* = *copy-phase*›

sepref-def *copy-phase-impl2*
is ‹*uncurry copy-phase2*›
:: ‹*phase-saver'-assn*^{k} $*_a$ *phase-saver-assn*^{d} $\rightarrow_a$ *phase-saver-assn*›
unfolding *copy-phase-def copy-phase2-def*
while-eq-nfoldli[*symmetric*]
apply (*subst while-upt-while-direct*, *simp*)
unfolding *simp-thms*(*21*) — remove $a \wedge$ *True* from condition
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *rephase-flipped-impl*
is ‹*rephase-flipped*›
:: ‹*phase-saver-assn*^{d} $\rightarrow_a$ *phase-saver-assn*›
supply [[*goals-limit=1*]]
unfolding *rephase-flipped-def*
while-eq-nfoldli[*symmetric*]
apply (*subst while-upt-while-direct*, *simp*)
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-register *rephase-init rephase-random copy-phase reset-best-phase reset-target-phase rephase-flipped*

sepref-def *reset-best-phase-impl*
is ‹*reset-best-phase*›
:: ‹*phase-heur-assn*^{d} $\rightarrow_a$ *phase-heur-assn*›
supply [[*goals-limit=1*]]
unfolding *reset-best-phase-def phase-heur-assn-def*
by *sepref*

sepref-def *reset-target-phase-impl*
is ‹*reset-target-phase*›
:: ‹*phase-heur-assn*^{d} $\rightarrow_a$ *phase-heur-assn*›
supply [[*goals-limit=1*]]
unfolding *reset-target-phase-def phase-heur-assn-def*
by *sepref*

sepref-def *phase-save-phase-impl*
is ‹*uncurry phase-save-phase*›
:: ‹*word64-assn*^{k} $*_a$ *phase-heur-assn*^{d} $\rightarrow_a$ *phase-heur-assn*›
supply [[*goals-limit=1*]]
unfolding *phase-save-phase-def phase-heur-assn-def*
by *sepref*

sepref-def *get-next-phase-imp*
is ‹*uncurry2 get-next-phase-stats*›
:: ‹*bool1-assn*^{k} $*_a$ *atom-assn*^{k} $*_a$ *phase-heur-assn*^{k} $\rightarrow_a$ *bool1-assn*›
unfolding *get-next-phase-stats-def phase-heur-assn-def*
apply *annot-all-atm-idxs*
by *sepref*

sepref-register *current-phase-letter*
sepref-def *current-phase-letter-impl*
is ‹*RETURN o current-phase-letter*›

:: ‹*word64-assn*^{k} $\rightarrow_a$ *word64-assn*›
unfolding *current-phase-letter-def*
by *sepref*

end
theory *IsaSAT-Reluctant-LLVM*
imports *IsaSAT-Reluctant WB-More-Word IsaSAT-Literals Watched-Literals.WB-More-IICF-LLVM IsaSAT-Literals-LLVM*
begin

type-synonym *reluctant-rel* = ‹*bool* × *bool* × *64 word* × *64 word* × *64 word* × *64 word* × *64 word*›
type-synonym *reluctant-rel-assn* = ‹*1 word* × *1 word* × *64 word* × *64 word* × *64 word* × *64 word* × *64 word*›
definition *reluctant-rel-assn* :: ‹*reluctant-rel* ⇒ *reluctant-rel-assn*⇒ *assn*›**where**
‹*reluctant-rel-assn* = *bool1-assn* ×$_a$ *bool1-assn* ×$_a$ *word-assn* ×$_a$ *word-assn* ×$_a$ *word-assn* ×$_a$ *word-assn* ×$_a$ *word-assn*›

definition *reluctant-rel* :: ‹(*reluctant-rel* × *reluctant*) *set*› **where**
‹*reluctant-rel* = {((*limited*, *trigger*, *u*, *v*, *period*, *wait*, *limit*), *r*).
limited = *reluctant-limited r* ∧
trigger = *reluctant-trigger r* ∧
u = *reluctant-u r* ∧
v = *reluctant-v r* ∧
period = *reluctant-period r* ∧
wait = *reluctant-wait r* ∧
limit = *reluctant-limit r*}›

definition *reluctant-assn* **where**
‹*reluctant-assn* = *hr-comp reluctant-rel-assn reluctant-rel*›
schematic-goal *mk-free-reluctant-rel-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE reluctant-rel-assn ?fr*›
unfolding *reluctant-rel-assn-def*
by *synthesize-free*

schematic-goal *mk-free-reluctant-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE reluctant-assn ?fr*›
unfolding *reluctant-assn-def*
by *synthesize-free*

lemma [*safe-constraint-rules*]:
‹*CONSTRAINT Sepref-Basic.is-pure reluctant-rel-assn*›
‹*CONSTRAINT Sepref-Basic.is-pure reluctant-assn*›
unfolding *reluctant-rel-assn-def reluctant-assn-def*
by (*auto intro*!: *hr-comp-is-pure*)

definition *reluctant-c* :: ‹ - ⇒ - ⇒ - ⇒ - ⇒ - ⇒ - ⇒ - ⇒ *reluctant-rel*›**where**
‹*reluctant-c limited trigger u v period wait limit* = (*limited*, *trigger*, *u*, *v*, *period*, *wait*, *limit*)›

lemma *reluctant-c-Reluctant*:
‹(*uncurry6* (*RETURN ooooooo reluctant-c*), *uncurry6* (*RETURN ooooooo Reluctant*)) ∈
Id ×$_f$ *Id* ×$_f$ *Id* ×$_f$ *Id* ×$_f$ *Id* ×$_f$ *Id* ×$_f$ *Id* → ⟨*reluctant-rel*⟩*nres-rel*›
by (*auto simp*: *reluctant-c-def reluctant-rel-def intro*!: *frefI nres-relI*)

sepref-register *reluctant-c*
sepref-def *reluctant-impl*
is ‹*uncurry6* (*RETURN ooooooo reluctant-c*)›
:: ‹*bool1-assn*^{k} *a* *bool1-assn*^{k} *a* *word-assn*^{k} *a* *word-assn*^{k} *a* *word-assn*^{k} *a* *word-assn*^{k} *a* *word-assn*^{k} $\rightarrow_a$ *reluctant-rel-assn*›

unfolding *reluctant-c-def reluctant-rel-assn-def*
by *sepref*

lemmas [*sepref-fr-rules*] =
reluctant-impl.refine[*FCOMP reluctant-c-Reluctant*, *unfolded reluctant-assn-def*[*symmetric*]]

definition *reluctant-c-limited* :: ‹*reluctant-rel* ⇒ *bool*›**where**
‹*reluctant-c-limited* = (λ(*limited*, *trigger*, *u*, *v*, *period*, *wait*, *limit*). *limited*)›

definition *reluctant-c-triggered* :: ‹*reluctant-rel* ⇒ *bool*›**where**
‹*reluctant-c-triggered* = (λ(*triggered*, *trigger*, *u*, *v*, *period*, *wait*, *limit*). *trigger*)›

definition *reluctant-c-u* :: ‹*reluctant-rel* ⇒ *64 word*›**where**
‹*reluctant-c-u* = (λ(*triggered*, *trigger*, *u*, *v*, *period*, *wait*, *limit*). *u*)›

definition *reluctant-c-v* :: ‹*reluctant-rel* ⇒ *64 word*›**where**
‹*reluctant-c-v* = (λ(*triggered*, *trigger*, *u*, *v*, *period*, *wait*, *limit*). *v*)›
definition *reluctant-c-period* :: ‹*reluctant-rel* ⇒ *64 word*›**where**
‹*reluctant-c-period* = (λ(*triggered*, *trigger*, *u*, *v*, *period*, *wait*, *limit*). *period*)›

definition *reluctant-c-wait* :: ‹*reluctant-rel* ⇒ *64 word*›**where**
‹*reluctant-c-wait* = (λ(*triggered*, *trigger*, *u*, *v*, *period*, *wait*, *limit*). *wait*)›

definition *reluctant-c-limit* :: ‹*reluctant-rel* ⇒ *64 word*›**where**
‹*reluctant-c-limit* = (λ(*triggered*, *trigger*, *u*, *v*, *period*, *wait*, *limit*). *limit*)›

sepref-def *reluctant-limited-impl*
is ‹*RETURN o reluctant-c-limited*›
:: ‹*reluctant-rel-assn*^{k} →$_a$ *bool1-assn*›
unfolding *reluctant-c-limited-def reluctant-rel-assn-def*
by *sepref*

sepref-def *reluctant-triggered-impl*
is ‹*RETURN o reluctant-c-triggered*›
:: ‹*reluctant-rel-assn*^{k} →$_a$ *bool1-assn*›
unfolding *reluctant-c-triggered-def reluctant-rel-assn-def*
by *sepref*

sepref-def *reluctant-u-impl*
is ‹*RETURN o reluctant-c-u*›
:: ‹*reluctant-rel-assn*^{k} →$_a$ *word64-assn*›
unfolding *reluctant-c-u-def reluctant-rel-assn-def*
by *sepref*

sepref-def *reluctant-v-impl*
is ‹*RETURN o reluctant-c-v*›
:: ‹*reluctant-rel-assn*^{k} →$_a$ *word64-assn*›
unfolding *reluctant-c-v-def reluctant-rel-assn-def*
by *sepref*

sepref-def *reluctant-wait-impl*
is ‹*RETURN o reluctant-c-wait*›
:: ‹*reluctant-rel-assn*^{k} →$_a$ *word64-assn*›
unfolding *reluctant-c-wait-def reluctant-rel-assn-def*
by *sepref*

sepref-def *reluctant-period-impl*
is ‹*RETURN o reluctant-c-period*›
:: ‹*reluctant-rel-assn*^{k} $\rightarrow_a$ *word64-assn*›
unfolding *reluctant-c-period-def reluctant-rel-assn-def*
by *sepref*

sepref-def *reluctant-limit-impl*
is ‹*RETURN o reluctant-c-limit*›
:: ‹*reluctant-rel-assn*^{k} $\rightarrow_a$ *word64-assn*›
unfolding *reluctant-c-limit-def reluctant-rel-assn-def*
by *sepref*

lemma *reluctant-c-limited*: ‹(*reluctant-c-limited*, *reluctant-limited*) ∈ *reluctant-rel* → *Id*› **and**
reluctant-c-triggered: ‹(*reluctant-c-triggered*, *reluctant-trigger*) ∈ *reluctant-rel* → *Id*› **and**
reluctant-c-u: ‹(*reluctant-c-u*, *reluctant-u*) ∈ *reluctant-rel* → *Id*› **and**
reluctant-c-v: ‹(*reluctant-c-v*, *reluctant-v*) ∈ *reluctant-rel* → *Id*› **and**
reluctant-c-wait: ‹(*reluctant-c-wait*, *reluctant-wait*) ∈ *reluctant-rel* → *Id*› **and**
reluctant-c-period: ‹(*reluctant-c-period*, *reluctant-period*) ∈ *reluctant-rel* → *Id*› **and**
reluctant-c-limit: ‹(*reluctant-c-limit*, *reluctant-limit*) ∈ *reluctant-rel* → *Id*›
by (*auto simp*: *reluctant-rel-def reluctant-c-limited-def reluctant-c-triggered-def*
reluctant-c-u-def reluctant-c-v-def reluctant-c-wait-def reluctant-c-period-def reluctant-c-limit-def
intro!: *frefI nres-relI*)

lemmas [*sepref-fr-rules*] =
reluctant-limited-impl.refine[*FCOMP reluctant-c-limited*, *unfolded reluctant-assn-def*[*symmetric*]]
reluctant-triggered-impl.refine[*FCOMP reluctant-c-triggered*, *unfolded reluctant-assn-def*[*symmetric*]]
reluctant-u-impl.refine[*FCOMP reluctant-c-u*, *unfolded reluctant-assn-def*[*symmetric*]]
reluctant-v-impl.refine[*FCOMP reluctant-c-v*, *unfolded reluctant-assn-def*[*symmetric*]]
reluctant-wait-impl.refine[*FCOMP reluctant-c-wait*, *unfolded reluctant-assn-def*[*symmetric*]]
reluctant-period-impl.refine[*FCOMP reluctant-c-period*, *unfolded reluctant-assn-def*[*symmetric*]]
reluctant-limit-impl.refine[*FCOMP reluctant-c-limit*, *unfolded reluctant-assn-def*[*symmetric*]]

sepref-register *reluctant-impl reluctant-tick reluctant-enable reluctant-set-trigger*
reluctant-triggered reluctant-untrigger reluctant-triggered2 reluctant-init
reluctant-disable

lemma *reluctant-tick-alt-def*:
‹*RETURN o reluctant-tick* =
(λ*r*. *let*
limited = *reluctant-limited r*;
trigger = *reluctant-trigger r*;
u = *reluctant-u r*;
v = *reluctant-v r*;
period = *reluctant-period r*;
wait = *reluctant-wait r*;
limit = *reluctant-limit r in*
(*if period* = *0* ∨ *trigger then RETURN* (*Reluctant limited trigger u v period* (*wait*) *limit*)
else if wait > *1 then RETURN* (*Reluctant limited trigger u v period* (*wait* − *1*) *limit*)
else let zero = *u*−*u*;
b = *u AND* (*zero* − *u*);
(*u*, *v*) = (*if b* = *v then* (*u*+*1*, *1*) *else* (*u*, *2* ∗ *v*));
(*u*, *v*) = (*if limited* ∧ *wait* > *limit then* (*1*,*1*) *else* (*u*, *v*));
wait = *v* ∗ *period in*
RETURN (*Reluctant limited True u v period wait limit*)))›
by (*auto intro*!: *ext simp*: *reluctant-tick-def Let-def*)

sepref-register ‹(*AND*) :: *'a* :: *len word* ⇒ - ⇒ -›
sepref-def *reluctant-tick-impl*
  is ‹*RETURN o reluctant-tick*›
  :: ‹*reluctant-assn*^{k} →$_a$ *reluctant-assn*›
  unfolding *reluctant-tick-alt-def*
  by *sepref*

export-llvm *reluctant-tick-impl*

sepref-def *reluctant-enable-impl*
  is ‹*uncurry* (*RETURN oo reluctant-enable*)›
  :: ‹*word-assn*^{k} *$_a$ *word-assn*^{k} →$_a$ *reluctant-assn*›
  unfolding *reluctant-enable-def*
  by *sepref*

sepref-def *reluctant-set-trigger-impl*
  is ‹*uncurry* (*RETURN oo reluctant-set-trigger*)›
  :: ‹*bool1-assn*^{k} *$_a$ *reluctant-assn*^{k} →$_a$ *reluctant-assn*›
  unfolding *reluctant-set-trigger-def*
  by *sepref*

sepref-def *reluctant-triggered-ether-impl*
  is ‹(*RETURN o reluctant-triggered*)›
  :: ‹*reluctant-assn*^{k} →$_a$ *reluctant-assn* ×$_a$ *bool1-assn*›
  unfolding *reluctant-triggered-def*
  by *sepref*

sepref-def *reluctant-triggered2-impl*
  is ‹(*RETURN o reluctant-triggered2*)›
  :: ‹*reluctant-assn*^{k} →$_a$ *bool1-assn*›
  unfolding *reluctant-triggered2-def*
  by *sepref*

sepref-def *reluctant-untrigger-impl*
  is ‹(*RETURN o reluctant-untrigger*)›
  :: ‹*reluctant-assn*^{k} →$_a$ *reluctant-assn*›
  unfolding *reluctant-untrigger-def*
  by *sepref*

sepref-def *reluctant-disable-impl*
  is ‹*RETURN o reluctant-disable*›
  :: ‹*reluctant-assn*^{k} →$_a$ *reluctant-assn*›
  unfolding *reluctant-disable-def*
  by *sepref*

sepref-def *reluctant-init-impl*
  is ‹*uncurry0* (*RETURN reluctant-init*)›
  :: ‹*unit-assn*^{k} →$_a$ *reluctant-assn*›
  unfolding *reluctant-init-def*
  apply (*annot-snat-const* ‹*TYPE*(*64*)›)
  by *sepref*

experiment
begin
  export-llvm *reluctant-init-impl reluctant-enable-impl reluctant-disable-impl reluctant-triggered2-impl*

reluctant-triggered-impl reluctant-set-trigger-impl reluctant-enable-impl reluctant-triggered-ether-impl
export-llvm *reluctant-tick-impl*

end

end
theory *Tuple16-LLVM*
imports *Tuple16 IsaSAT-Literals-LLVM*
begin

hide-const (**open**) *NEMonad.ASSERT NEMonad.RETURN*

instantiation *tuple16* ::
(*llvm-rep*,*llvm-rep*,*llvm-rep*,*llvm-rep*,
llvm-rep,*llvm-rep*,*llvm-rep*,*llvm-rep*,
llvm-rep,*llvm-rep*,*llvm-rep*,*llvm-rep*,
llvm-rep,*llvm-rep*,*llvm-rep*,*llvm-rep*) *llvm-rep*
begin
definition *to-val-tuple16* **where**
‹*to-val-tuple16* ≡ (λ*S*. *case S of*
Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena ⇒ *LL-STRUCT* [*to-val M*, *to-val N*, *to-val D*, *to-val i*, *to-val W*, *to-val ivmtf*,
to-val icount, *to-val ccach*, *to-val lbd*,
to-val outl, *to-val stats*, *to-val heur*, *to-val aivdom*, *to-val clss*, *to-val opts*, *to-val arena*])›

definition *from-val-tuple16* :: ‹*llvm-val* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*, *'p*) *tuple16*› **where**
‹*from-val-tuple16* ≡ (λ*p*. *case llvm-val.the-fields p of*
[*M*, *N*, *D*, *i*, *W*, *ivmtf*, *icount*, *ccach*, *lbd*, *outl*, *stats*, *heur*, *aivdom*, *clss*, *opts*, *arena*] ⇒
Tuple16 (*from-val M*) (*from-val N*) (*from-val D*) (*from-val i*) (*from-val W*) (*from-val ivmtf*) (*from-val icount*) (*from-val ccach*) (*from-val lbd*)
(*from-val outl*) (*from-val stats*) (*from-val heur*) (*from-val aivdom*) (*from-val clss*) (*from-val opts*) (*from-val arena*))›

definition [*simp*]: *struct-of-tuple16* (- :: (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*, *'p*) *tuple16 itself*) ≡
VS-STRUCT [*struct-of TYPE*(*'a*), *struct-of TYPE*(*'b*), *struct-of TYPE*(*'c*),
struct-of TYPE(*'d*), *struct-of TYPE*(*'e*), *struct-of TYPE*(*'f*), *struct-of TYPE*(*'g*), *struct-of TYPE*(*'h*),
struct-of TYPE(*'i*), *struct-of TYPE*(*'j*), *struct-of TYPE*(*'k*), *struct-of TYPE*(*'l*),
struct-of TYPE(*'m*), *struct-of TYPE*(*'n*), *struct-of TYPE*(*'o*), *struct-of TYPE*(*'p*)]

definition [*simp*]: *init-tuple16* :: (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*, *'p*) *tuple16* ≡ *Tuple16 init init init init init init init init init init init init init init init init*

instance
apply *standard*
unfolding *from-val-tuple16-def to-val-tuple16-def struct-of-tuple16-def init-tuple16-def comp-def tuple16.case-distrib*
subgoal
by (*auto simp*: *init-zero fun-eq-iff from-val-tuple16-def split*: *tuple16.splits*)
subgoal for *v*
by (*cases v*) (*auto split*: *list.splits tuple16.splits*)
subgoal for *v*
by (*cases v*)
(*simp add*: *LLVM-Shallow.null-def to-val-ptr-def split*: *tuple16.splits*)

subgoal
by (*simp add*: *LLVM-Shallow.null-def to-val-ptr-def to-val-word-def init-zero split*: *tuple16.splits*)
done
end

Setup for LLVM code export

Declare structure to code generator.

lemma *to-val-tuple16*[*ll-struct-of*]: *struct-of TYPE*(('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*, '*p*) *tuple16*) = *VS-STRUCT* [
struct-of TYPE('*a*::*llvm-rep*),
struct-of TYPE('*b*::*llvm-rep*),
struct-of TYPE('*c*::*llvm-rep*),
struct-of TYPE('*d*::*llvm-rep*),
struct-of TYPE('*e*::*llvm-rep*),
struct-of TYPE('*f*::*llvm-rep*),
struct-of TYPE('*g*::*llvm-rep*),
struct-of TYPE('*h*::*llvm-rep*),
struct-of TYPE('*i*::*llvm-rep*),
struct-of TYPE('*j*::*llvm-rep*),
struct-of TYPE('*k*::*llvm-rep*),
struct-of TYPE('*l*::*llvm-rep*),
struct-of TYPE('*m*::*llvm-rep*),
struct-of TYPE('*n*::*llvm-rep*),
struct-of TYPE('*o*::*llvm-rep*),
struct-of TYPE('*p*::*llvm-rep*)]
by (*auto*)

lemma *node-insert-value*:
ll-insert-value (*Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena*) *M'* *0* = *Mreturn* (*Tuple16 M' N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena*)
ll-insert-value (*Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena*) *N'* (*Suc 0*) = *Mreturn* (*Tuple16 M N' D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena*)
ll-insert-value (*Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena*) *D'* *2* = *Mreturn* (*Tuple16 M N D' i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena*)
ll-insert-value (*Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena*) *i'* *3* = *Mreturn* (*Tuple16 M N D i' W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena*)
ll-insert-value (*Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena*) *W'* *4* = *Mreturn* (*Tuple16 M N D i W' ivmtf icount ccach lbd outl stats heur aivdom clss opts arena*)
ll-insert-value (*Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena*) *ivmtf'* *5* = *Mreturn* (*Tuple16 M N D i W ivmtf' icount ccach lbd outl stats heur aivdom clss opts arena*)
ll-insert-value (*Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena*) *icount'* *6* = *Mreturn* (*Tuple16 M N D i W ivmtf icount' ccach lbd outl stats heur aivdom clss opts arena*)
ll-insert-value (*Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena*) *ccach'* *7* = *Mreturn* (*Tuple16 M N D i W ivmtf icount ccach' lbd outl stats heur aivdom clss opts arena*)
ll-insert-value (*Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena*) *lbd'* *8* = *Mreturn* (*Tuple16 M N D i W ivmtf icount ccach lbd' outl stats heur aivdom clss opts arena*)
ll-insert-value (*Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena*) *outl'* *9* = *Mreturn* (*Tuple16 M N D i W ivmtf icount ccach lbd outl' stats heur aivdom clss opts arena*)
ll-insert-value (*Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena*) *stats'* *10* = *Mreturn* (*Tuple16 M N D i W ivmtf icount ccach lbd outl stats' heur aivdom clss opts arena*)
ll-insert-value (*Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena*) *heur'* *11* = *Mreturn* (*Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur' aivdom clss opts arena*)
ll-insert-value (*Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena*)

aivdom' 12 = Mreturn (Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom' clss opts arena)
ll-insert-value (Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena) clss' 13 = Mreturn (Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss' opts arena)
ll-insert-value (Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena) opts' 14 = Mreturn (Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts' arena)
ll-insert-value (Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena) arena' 15 = Mreturn (Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena')
by (*simp-all add*: *ll-insert-value-def llvm-insert-value-def Let-def checked-from-val-def to-val-tuple16-def from-val-tuple16-def*)

lemma *node-extract-value*:
ll-extract-value (Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena) 0 = Mreturn M
ll-extract-value (Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena) (Suc 0) = Mreturn N
ll-extract-value (Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena) 2 = Mreturn D
ll-extract-value (Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena) 3 = Mreturn i
ll-extract-value (Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena) 4 = Mreturn W
ll-extract-value (Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena) 5 = Mreturn ivmtf
ll-extract-value (Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena) 6 = Mreturn icount
ll-extract-value (Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena) 7 = Mreturn ccach
ll-extract-value (Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena) 8 = Mreturn lbd
ll-extract-value (Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena) 9 = Mreturn outl
ll-extract-value (Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena) 10 = Mreturn stats
ll-extract-value (Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena) 11 = Mreturn heur
ll-extract-value (Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena) 12 = Mreturn aivdom
ll-extract-value (Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena) 13 = Mreturn clss
ll-extract-value (Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena) 14 = Mreturn opts
ll-extract-value (Tuple16 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena) 15 = Mreturn arena
apply (*simp-all add*: *ll-extract-value-def llvm-extract-value-def Let-def checked-from-val-def to-val-tuple16-def from-val-tuple16-def*)
done

Lemmas to translate node construction and destruction

lemma *inline-return-node*[*llvm-pre-simp*]: *Mreturn (Tuple16 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena) = doM {*
r ← ll-insert-value init M 0;
r ← ll-insert-value r N 1;
r ← ll-insert-value r D 2;
r ← ll-insert-value r i 3;
r ← ll-insert-value r W 4;

```
    r ← ll-insert-value r ivmtf 5;
    r ← ll-insert-value r icount 6;
    r ← ll-insert-value r ccach 7;
    r ← ll-insert-value r lbd 8;
    r ← ll-insert-value r outl 9;
    r ← ll-insert-value r heur 10;
    r ← ll-insert-value r stats 11;
    r ← ll-insert-value r aivdom 12;
    r ← ll-insert-value r clss 13;
    r ← ll-insert-value r opts 14;
    r ← ll-insert-value r arena 15;
    Mreturn r
  }
  apply (auto simp: node-insert-value)
  done

lemma inline-node-case[llvm-pre-simp]: (case r of (Tuple16 M N D i W ivmtf icount ccach lbd outl heur
stats aivdom clss opts arena) ⇒ f M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts
arena) = doM {
    M ← ll-extract-value r 0;
    N ← ll-extract-value r 1;
    D ← ll-extract-value r 2;
    i ← ll-extract-value r 3;
    W ← ll-extract-value r 4;
    ivmtf ← ll-extract-value r 5;
    icount ← ll-extract-value r 6;
    ccach ← ll-extract-value r 7;
    lbd ← ll-extract-value r 8;
    outl ← ll-extract-value r 9;
    heur ← ll-extract-value r 10;
    stats ← ll-extract-value r 11;
    aivdom ← ll-extract-value r 12;
    clss ← ll-extract-value r 13;
    opts ← ll-extract-value r 14;
    arena ← ll-extract-value r 15;
  f M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena
}
  apply (cases r)
  apply (auto simp: node-extract-value)
  done

lemma inline-return-node-case[llvm-pre-simp]: doM {Mreturn (case r of (Tuple16 M N D i W ivmtf
icount ccach lbd outl heur stats aivdom clss opts arena) ⇒ f M N D i W ivmtf icount ccach lbd outl heur
stats aivdom clss opts arena)} = doM {
    M ← ll-extract-value r 0;
    N ← ll-extract-value r 1;
    D ← ll-extract-value r 2;
    i ← ll-extract-value r 3;
    W ← ll-extract-value r 4;
    ivmtf ← ll-extract-value r 5;
    icount ← ll-extract-value r 6;
    ccach ← ll-extract-value r 7;
    lbd ← ll-extract-value r 8;
    outl ← ll-extract-value r 9;
    heur ← ll-extract-value r 10;
    stats ← ll-extract-value r 11;
```

```
    aivdom ← ll-extract-value r 12;
    clss ← ll-extract-value r 13;
    opts ← ll-extract-value r 14;
    arena ← ll-extract-value r 15;
  Mreturn (f M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena)
}
  apply (cases r)
  apply (auto simp: node-extract-value)
  done
lemma inline-direct-return-node-case[llvm-pre-simp]: doM {(case r of (Tuple16 M N D i W ivmtf icount
ccach lbd outl heur stats aivdom clss opts arena) ⇒ f M N D i W ivmtf icount ccach lbd outl heur stats
aivdom clss opts arena)} = doM {
    M ← ll-extract-value r 0;
    N ← ll-extract-value r 1;
    D ← ll-extract-value r 2;
    i ← ll-extract-value r 3;
    W ← ll-extract-value r 4;
    ivmtf ← ll-extract-value r 5;
    icount ← ll-extract-value r 6;
    ccach ← ll-extract-value r 7;
    lbd ← ll-extract-value r 8;
    outl ← ll-extract-value r 9;
    heur ← ll-extract-value r 10;
    stats ← ll-extract-value r 11;
    aivdom ← ll-extract-value r 12;
    clss ← ll-extract-value r 13;
    opts ← ll-extract-value r 14;
    arena ← ll-extract-value r 15;
   (f M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena)
}
  apply (cases r)
  apply (auto simp: node-extract-value)
  done

lemmas [llvm-inline] =
  tuple16.Tuple16-get-a-def
  tuple16.Tuple16-get-b-def
  tuple16.Tuple16-get-c-def
  tuple16.Tuple16-get-d-def
  tuple16.Tuple16-get-e-def
  tuple16.Tuple16-get-f-def
  tuple16.Tuple16-get-g-def
  tuple16.Tuple16-get-h-def
  tuple16.Tuple16-get-i-def
  tuple16.Tuple16-get-j-def
  tuple16.Tuple16-get-l-def
  tuple16.Tuple16-get-k-def
  tuple16.Tuple16-get-m-def
  tuple16.Tuple16-get-n-def
  tuple16.Tuple16-get-o-def
  tuple16.Tuple16-get-p-def

fun tuple16-assn :: ‹
  ('a ⇒ - ⇒ llvm-amemory ⇒ bool) ⇒
  ('b ⇒ - ⇒ llvm-amemory ⇒ bool) ⇒
  ('c ⇒ - ⇒ llvm-amemory ⇒ bool) ⇒
```

```
  ('d ⇒ - ⇒ llvm-amemory ⇒ bool) ⇒
  ('e ⇒ - ⇒ llvm-amemory ⇒ bool) ⇒
  ('f ⇒ - ⇒ llvm-amemory ⇒ bool) ⇒
  ('g ⇒ - ⇒ llvm-amemory ⇒ bool) ⇒
  ('h ⇒ - ⇒ llvm-amemory ⇒ bool) ⇒
  ('i ⇒ - ⇒ llvm-amemory ⇒ bool) ⇒
  ('j ⇒ - ⇒ llvm-amemory ⇒ bool) ⇒
  ('k ⇒ - ⇒ llvm-amemory ⇒ bool) ⇒
  ('l ⇒ - ⇒ llvm-amemory ⇒ bool) ⇒
  ('m ⇒ - ⇒ llvm-amemory ⇒ bool) ⇒
  ('n ⇒ - ⇒ llvm-amemory ⇒ bool) ⇒
  ('o ⇒ - ⇒ llvm-amemory ⇒ bool) ⇒
  ('p ⇒ - ⇒ llvm-amemory ⇒ bool) ⇒
  ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
     'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ - ⇒ assn› where
 ‹tuple16-assn a-assn b-assn' c-assn d-assn e-assn f-assn g-assn h-assn i-assn j-assn k-assn l-assn m-assn
n-assn o-assn p-assn S T =
   (case (S, T) of
  (Tuple16 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena,
   Tuple16 M' N' D' i' W' ivmtf' icount' ccach' lbd' outl' heur' stats' aivdom' clss' opts' arena')
     ⇒
 (a-assn M (M') ∧* b-assn' N (N') ∧* c-assn D (D') ∧* d-assn i (i') ∧*
 e-assn W (W') ∧* f-assn ivmtf (ivmtf') ∧* g-assn icount (icount') ∧* h-assn ccach (ccach') ∧*
 i-assn lbd (lbd') ∧* j-assn outl (outl') ∧* k-assn heur (heur') ∧* l-assn stats (stats') ∧*
 m-assn aivdom (aivdom') ∧* n-assn clss (clss') ∧* o-assn opts (opts') ∧* p-assn arena (arena')))
  ›


locale tuple16-ops =
  fixes
    a-assn :: ‹'a ⇒ 'xa :: llvm-rep ⇒ assn› and
    b-assn :: ‹'b ⇒ 'xb:: llvm-rep ⇒ assn› and
    c-assn :: ‹'c ⇒ 'xc:: llvm-rep ⇒ assn› and
    d-assn :: ‹'d ⇒ 'xd:: llvm-rep ⇒ assn› and
    e-assn :: ‹'e ⇒ 'xe:: llvm-rep ⇒ assn› and
    f-assn :: ‹'f ⇒ 'xf:: llvm-rep ⇒ assn› and
    g-assn :: ‹'g ⇒ 'xg:: llvm-rep ⇒ assn› and
    h-assn :: ‹'h ⇒ 'xh:: llvm-rep ⇒ assn› and
    i-assn :: ‹'i ⇒ 'xi:: llvm-rep ⇒ assn› and
    j-assn :: ‹'j ⇒ 'xj:: llvm-rep ⇒ assn› and
    k-assn :: ‹'k ⇒ 'xk:: llvm-rep ⇒ assn› and
    l-assn :: ‹'l ⇒ 'xl:: llvm-rep ⇒ assn› and
    m-assn :: ‹'m ⇒ 'xm:: llvm-rep ⇒ assn› and
    n-assn :: ‹'n ⇒ 'xn:: llvm-rep ⇒ assn› and
    o-assn :: ‹'o ⇒ 'xo:: llvm-rep ⇒ assn› and
    p-assn :: ‹'p ⇒ 'xp:: llvm-rep ⇒ assn› and
    a-default :: 'a and
    a :: ‹'xa llM› and
    b-default :: 'b and
    b :: ‹'xb llM› and
    c-default :: 'c and
    c :: ‹'xc llM› and
    d-default :: 'd and
    d :: ‹'xd llM› and
    e-default :: 'e and
    e :: ‹'xe llM› and
```

f-default :: *'f* **and**
f :: ‹*'xf llM*› **and**
g-default :: *'g* **and**
g :: ‹*'xg llM*› **and**
h-default :: *'h* **and**
h :: ‹*'xh llM*› **and**
i-default :: *'i* **and**
i :: ‹*'xi llM*› **and**
j-default :: *'j* **and**
j :: ‹*'xj llM*› **and**
k-default :: *'k* **and**
k :: ‹*'xk llM*› **and**
l-default :: *'l* **and**
l :: ‹*'xl llM*› **and**
m-default :: *'m* **and**
m :: ‹*'xm llM*› **and**
n-default :: *'n* **and**
n :: ‹*'xn llM*› **and**
ko-default :: *'o* **and**
ko :: ‹*'xo llM*› **and**
p-default :: *'p* **and**
p :: ‹*'xp llM*›
begin

definition *isasat-assn* :: ‹- ⇒ - ⇒ *assn*› **where**
‹*isasat-assn* = *tuple16-assn*
a-assn b-assn c-assn d-assn
e-assn f-assn g-assn h-assn
i-assn j-assn k-assn l-assn
m-assn n-assn o-assn p-assn›

definition *remove-a* :: ‹(*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*) *tuple16* ⇒ - × (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*) *tuple16*› **where**
‹*remove-a tuple16* = (*case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16*
⇒
(*x1*, *Tuple16 a-default x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16*))›

definition *remove-b* :: ‹(*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*) *tuple16* ⇒ *'b* × (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*) *tuple16*› **where**
‹*remove-b tuple16* = (*case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16*
⇒
(*x2*, *Tuple16 x1 b-default x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16*))›

definition *remove-c*:: ‹(*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*) *tuple16* ⇒ - × (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*) *tuple16*› **where**
‹*remove-c tuple16* = (*case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16*
⇒
(*x3*, *Tuple16 x1 x2 c-default x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16*))›

definition *remove-d* :: ‹(*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*) *tuple16* ⇒ - × (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*) *tuple16*› **where**
‹*remove-d tuple16* = (*case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16*

```
⇒
     (x4, Tuple16 x1 x2 x3 d-default x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16))›

definition remove-e :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
     'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ 'e × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
     'k, 'l, 'm, 'n, 'o, 'p) tuple16› where
  ‹remove-e tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16
⇒
     (x5, Tuple16 x1 x2 x3 x4 e-default x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16))›

definition remove-f :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
     'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ 'f × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
     'k, 'l, 'm, 'n, 'o, 'p) tuple16› where
  ‹remove-f tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16
⇒
     (x6, Tuple16 x1 x2 x3 x4 x5 f-default x7 x8 x9 x10 x11 x12 x13 x14 x15 x16))›

definition remove-g :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
     'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ 'g × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
     'k, 'l, 'm, 'n, 'o, 'p) tuple16› where
  ‹remove-g tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16
⇒
     (x7, Tuple16 x1 x2 x3 x4 x5 x6 g-default x8 x9 x10 x11 x12 x13 x14 x15 x16))›

definition remove-h :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
     'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ 'h × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
     'k, 'l, 'm, 'n, 'o, 'p) tuple16› where
  ‹remove-h tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16
⇒
     (x8, Tuple16 x1 x2 x3 x4 x5 x6 x7 h-default x9 x10 x11 x12 x13 x14 x15 x16))›

definition remove-i :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
     'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ 'i × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
     'k, 'l, 'm, 'n, 'o, 'p) tuple16› where
  ‹remove-i tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16
⇒
     (x9, Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 i-default x10 x11 x12 x13 x14 x15 x16))›

definition remove-j :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
     'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ 'j × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
     'k, 'l, 'm, 'n, 'o, 'p) tuple16› where
  ‹remove-j tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16
⇒
     (x10, Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 j-default x11 x12 x13 x14 x15 x16))›

definition remove-k :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
     'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ 'k × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
     'k, 'l, 'm, 'n, 'o, 'p) tuple16› where
  ‹remove-k tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16
⇒
     (x11, Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 k-default x12 x13 x14 x15 x16))›

definition remove-l :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
     'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ 'l × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
     'k, 'l, 'm, 'n, 'o, 'p) tuple16› where
  ‹remove-l tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16
```

```
⇒
    (x12, Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 l-default x13 x14 x15 x16))›

definition remove-m :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
    'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ 'm × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
    'k, 'l, 'm, 'n, 'o, 'p) tuple16› where
  ‹remove-m tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16
⇒
    (x13, Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 m-default x14 x15 x16))›

definition remove-n :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
    'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ 'n × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
    'k, 'l, 'm, 'n, 'o, 'p) tuple16› where
  ‹remove-n tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16
⇒
    (x14, Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 n-default x15 x16))›

definition remove-o :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
    'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ 'o × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
    'k, 'l, 'm, 'n, 'o, 'p) tuple16› where
  ‹remove-o tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16
⇒
    (x15, Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 ko-default x16))›

definition remove-p :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
    'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ 'p × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
    'k, 'l, 'm, 'n, 'o, 'p) tuple16› where
  ‹remove-p tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16
⇒
    (x16, Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 p-default))›

definition update-a :: ‹'a ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
    'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
    'k, 'l, 'm, 'n, 'o, 'p) tuple16› where
  ‹update-a x1 tuple16 = (case tuple16 of Tuple16 M x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15
x16 ⇒
    let - = M in
    Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16)›

definition update-b :: ‹'b ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
    'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
    'k, 'l, 'm, 'n, 'o, 'p) tuple16› where
  ‹update-b x2 tuple16 = (case tuple16 of Tuple16 x1 M x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15
x16 ⇒
    let - = M in
    Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16)›

definition update-c:: ‹'c ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
    'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
    'k, 'l, 'm, 'n, 'o, 'p) tuple16› where
  ‹update-c x3 tuple16 = (case tuple16 of Tuple16 x1 x2 M x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15
x16 ⇒
    let - = M in
    Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16)›

definition update-d :: ‹'d ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
```

'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16› **where**
‹update-d x4 tuple16 = (case tuple16 of Tuple16 x1 x2 x3 M x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 ⇒
let - = M in
Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16)›

definition update-e :: ‹'e ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16› **where**
‹update-e x5 tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 M x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 ⇒
let - = M in
Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16)›

definition update-f :: ‹'f ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16› **where**
‹update-f x6 tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 x5 M x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 ⇒
let - = M in
Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16)›

definition update-g :: ‹'g ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16› **where**
‹update-g x7 tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 M x8 x9 x10 x11 x12 x13 x14 x15 x16 ⇒
let - = M in
Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16)›

definition update-h :: ‹'h ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16› **where**
‹update-h x8 tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 M x9 x10 x11 x12 x13 x14 x15 x16 ⇒
let - = M in
Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16)›

definition update-i :: ‹'i ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16› **where**
‹update-i x9 tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 M x10 x11 x12 x13 x14 x15 x16 ⇒
let - = M in
Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16)›

definition update-j :: ‹'j ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16› **where**
‹update-j x10 tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 M x11 x12 x13 x14 x15 x16 ⇒
let - = M in
Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16)›

definition update-k :: ‹'k ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,

'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16› **where**
‹update-k x11 tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 M x12 x13 x14 x15 x16 ⇒
let - = M in
Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16)›

definition update-l :: ‹'l ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16› **where**
‹update-l x12 tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 M x13 x14 x15 x16 ⇒
let - = M in
Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16)›

definition update-m :: ‹'m ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16› **where**
‹update-m x13 tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 M x14 x15 x16 ⇒
let - = M in
Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16)›

definition update-n :: ‹'n ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16› **where**
‹update-n x14 tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 M x15 x16 ⇒
let - = M in
Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16)›

definition update-o :: ‹'o ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16› **where**
‹update-o x15 tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 M x16 ⇒
let - = M in
Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16)›

definition update-p :: ‹'p ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16› **where**
‹update-p x16 tuple16 = (case tuple16 of Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 M ⇒
let - = M in
Tuple16 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16)›

end

lemma tuple16-assn-conv[simp]:
tuple16-assn P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13 P14 P15 P16 (Tuple16 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16)
(Tuple16 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15' a16') =
(P1 a1 a1' ∧*

P2 a2 a2' ∧*
P3 a3 a3' ∧*
P4 a4 a4' ∧*
P5 a5 a5' ∧*
P6 a6 a6' ∧*
P7 a7 a7' ∧*
P8 a8 a8' ∧* *P9 a9 a9'* ∧* *P10 a10 a10'* ∧* *P11 a11 a11'* ∧* *P12 a12 a12'* ∧* *P13 a13 a13'* ∧* *P14 a14 a14'* ∧* *P15 a15 a15'* ∧* *P16 a16 a16'*)
unfolding *tuple16-assn.simps* **by** *auto*
lemma *tuple16-assn-ctxt*:
‹*tuple16-assn P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13 P14 P15 P16* (*Tuple16 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16*)
(*Tuple16 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15' a16'*) = *z* ⟹
hn-ctxt (*tuple16-assn P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13 P14 P15 P16*) (*Tuple16 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16*)
(*Tuple16 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15' a16'*)= *z*›
by (*simp add*: *hn-ctxt-def*)

lemma *hn-case-tuple16'*[*sepref-comb-rules*]:
assumes *FR*: ‹Γ ⊢ *hn-ctxt* (*tuple16-assn P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13 P14 P15 P16*) *p' p* ** Γ*1*›
assumes *Pair*: ⋀*a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15' a16'*.
⟦*p'*=*Tuple16 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15' a16'*⟧
⟹ *hn-refine* (*hn-ctxt P1 a1' a1* ∧* *hn-ctxt P2 a2' a2* ∧* *hn-ctxt P3 a3' a3* ∧* *hn-ctxt P4 a4' a4* ∧*
hn-ctxt P5 a5' a5 ∧* *hn-ctxt P6 a6' a6* ∧* *hn-ctxt P7 a7' a7* ∧* *hn-ctxt P8 a8' a8* ∧*
hn-ctxt P9 a9' a9 ∧* *hn-ctxt P10 a10' a10* ∧* *hn-ctxt P11 a11' a11* ∧* *hn-ctxt P12 a12' a12* ∧*
hn-ctxt P13 a13' a13 ∧* *hn-ctxt P14 a14' a14* ∧* *hn-ctxt P15 a15' a15* ∧* *hn-ctxt P16 a16' a16* ∧* Γ*1*)
(*f a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16*)
(Γ*2 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15' a16'*) *R*
(*CP a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16*)
(*f' a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15' a16'*)
assumes *FR2*: ‹⋀*a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15' a16'*.
Γ*2 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15' a16'* ⊢
hn-ctxt P1' a1' a1 ** *hn-ctxt P2' a2' a2* ** *hn-ctxt P3' a3' a3* ** *hn-ctxt P4' a4' a4* **
hn-ctxt P5' a5' a5 ** *hn-ctxt P6' a6' a6* ** *hn-ctxt P7' a7' a7* ** *hn-ctxt P8' a8' a8* **
hn-ctxt P9' a9' a9 ** *hn-ctxt P10' a10' a10* ** *hn-ctxt P11' a11' a11* ** *hn-ctxt P12' a12' a12* **
hn-ctxt P13' a13' a13 ** *hn-ctxt P14' a14' a14* ** *hn-ctxt P15' a15' a15* ** *hn-ctxt P16' a16' a16* ** Γ*1'*›
shows ‹*hn-refine* Γ (*case-tuple16 f p*) (*hn-ctxt* (*tuple16-assn P1' P2' P3' P4' P5' P6' P7' P8' P9' P10' P11' P12' P13' P14' P15' P16'*) *p' p* ** Γ*1'*)
R (*case-tuple16 CP p*) (*case-tuple16*$(λ*a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16*. *f' a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16*)$*p'*)› (**is** ‹*?G* Γ›)
unfolding *autoref-tag-defs PROTECT2-def*
apply1 (*rule hn-refine-cons-pre*[*OF FR*])
apply1 (*cases p*; *cases p'*; *simp add*: *tuple16-assn-conv*[*THEN tuple16-assn-ctxt*])
unfolding *CP-SPLIT-def prod.simps*
apply (*rule hn-refine-cons*[*OF - Pair - entails-refl*])
applyS (*simp add*: *hn-ctxt-def*)
applyS *simp* **using** *FR2*

by (*simp add*: *hn-ctxt-def*)

lemma *case-tuple16-arity*[*sepref-monadify-arity*]:
‹*case-tuple16* ≡ λ$_2$*fp p*. *SP case-tuple16*$(λ$_2$*a b*. *fp*$*a*$*b*)$*p*›
by (*simp-all only*: *SP-def APP-def PROTECT2-def RCALL-def*)

lemma *case-tuple16-comb*[*sepref-monadify-comb*]:
‹⋀*fp p*. *case-tuple16*$*fp*$*p* ≡ *Refine-Basic.bind*$(*EVAL*$*p*)$(λ$_2$*p*. (*SP case-tuple16*$*fp*$*p*))›
by (*simp-all*)

lemma *case-tuple16-plain-comb*[*sepref-monadify-comb*]:
EVAL$(*case-tuple16*$(λ$_2$*a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16*. *fp a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16*)$*p*) ≡
Refine-Basic.bind$(*EVAL*$*p*)$(λ$_2$*p*. *case-tuple16*$(λ$_2$*a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16*. *EVAL*$(*fp a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16*))$*p*)
apply (*rule eq-reflection*, *simp split*: *list.split prod.split option.split tuple16.split*)+
done

lemma *ho-tuple16-move*[*sepref-preproc*]: ‹*case-tuple16* (λ*a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 x*. *f x a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16*) =
(λ*p x*. *case-tuple16* (*f x*) *p*)›
by (*auto split*: *tuple16.splits*)

locale *tuple16* =
tuple16-ops a-assn b-assn c-assn d-assn e-assn
f-assn g-assn h-assn i-assn j-assn
k-assn l-assn m-assn n-assn o-assn p-assn
a-default a
b-default b
c-default c
d-default d
e-default e
f-default f
g-default g
h-default h
i-default i
j-default j
k-default k
l-default l
m-default m
n-default n
ko-default ko
p-default p
for
a-assn :: ‹'*a* ⇒ '*xa*:: *llvm-rep* ⇒ *assn*› **and**
b-assn :: ‹'*b* ⇒ '*xb*:: *llvm-rep* ⇒ *assn*› **and**
c-assn :: ‹'*c* ⇒ '*xc*:: *llvm-rep* ⇒ *assn*› **and**
d-assn :: ‹'*d* ⇒ '*xd*:: *llvm-rep* ⇒ *assn*› **and**
e-assn :: ‹'*e* ⇒ '*xe*:: *llvm-rep* ⇒ *assn*› **and**
f-assn :: ‹'*f* ⇒ '*xf*:: *llvm-rep* ⇒ *assn*› **and**
g-assn :: ‹'*g* ⇒ '*xg*:: *llvm-rep* ⇒ *assn*› **and**
h-assn :: ‹'*h* ⇒ '*xh*:: *llvm-rep* ⇒ *assn*› **and**
i-assn :: ‹'*i* ⇒ '*xi*:: *llvm-rep* ⇒ *assn*› **and**
j-assn :: ‹'*j* ⇒ '*xj*:: *llvm-rep* ⇒ *assn*› **and**
k-assn :: ‹'*k* ⇒ '*xk*:: *llvm-rep* ⇒ *assn*› **and**

l-assn :: ‹*'l* ⇒ *'xl*:: *llvm-rep* ⇒ *assn*› **and**
m-assn :: ‹*'m* ⇒ *'xm*:: *llvm-rep* ⇒ *assn*› **and**
n-assn :: ‹*'n* ⇒ *'xn*:: *llvm-rep* ⇒ *assn*› **and**
o-assn :: ‹*'o* ⇒ *'xo*:: *llvm-rep* ⇒ *assn*› **and**
p-assn :: ‹*'p* ⇒ *'xp*:: *llvm-rep* ⇒ *assn*› **and**
a-default :: *'a* **and**
a :: ‹*'xa llM*› **and**
b-default :: *'b* **and**
b :: ‹*'xb llM*› **and**
c-default :: *'c* **and**
c :: ‹*'xc llM*› **and**
d-default :: *'d* **and**
d :: ‹*'xd llM*› **and**
e-default :: *'e* **and**
e :: ‹*'xe llM*› **and**
f-default :: *'f* **and**
f :: ‹*'xf llM*› **and**
g-default :: *'g* **and**
g :: ‹*'xg llM*› **and**
h-default :: *'h* **and**
h :: ‹*'xh llM*› **and**
i-default :: *'i* **and**
i :: ‹*'xi llM*› **and**
j-default :: *'j* **and**
j :: ‹*'xj llM*› **and**
k-default :: *'k* **and**
k :: ‹*'xk llM*› **and**
l-default :: *'l* **and**
l :: ‹*'xl llM*› **and**
m-default :: *'m* **and**
m :: ‹*'xm llM*› **and**
n-default :: *'n* **and**
n :: ‹*'xn llM*› **and**
ko-default :: *'o* **and**
ko :: ‹*'xo llM*› **and**
p-default :: *'p* **and**
p :: ‹*'xp llM*› **and**
a-free :: ‹*'xa* ⇒ *unit llM*› **and**
b-free :: ‹*'xb* ⇒ *unit llM*› **and**
c-free :: ‹*'xc* ⇒ *unit llM*› **and**
d-free :: ‹*'xd* ⇒ *unit llM*› **and**
e-free :: ‹*'xe* ⇒ *unit llM*› **and**
f-free :: ‹*'xf* ⇒ *unit llM*› **and**
g-free :: ‹*'xg* ⇒ *unit llM*› **and**
h-free :: ‹*'xh* ⇒ *unit llM*› **and**
i-free :: ‹*'xi* ⇒ *unit llM*› **and**
j-free :: ‹*'xj* ⇒ *unit llM*› **and**
k-free :: ‹*'xk* ⇒ *unit llM*› **and**
l-free :: ‹*'xl* ⇒ *unit llM*› **and**
m-free :: ‹*'xm* ⇒ *unit llM*› **and**
n-free :: ‹*'xn* ⇒ *unit llM*› **and**
o-free :: ‹*'xo* ⇒ *unit llM*› **and**
p-free :: ‹*'xp* ⇒ *unit llM*› +
assumes
a: ‹(*uncurry0 a*, *uncurry0* (*RETURN a-default*)) ∈ *unit-assn*^{k} →$_a$ *a-assn*› **and**
b: ‹(*uncurry0 b*, *uncurry0* (*RETURN b-default*)) ∈ *unit-assn*^{k} →$_a$ *b-assn*› **and**

c: ‹(*uncurry0 c*, *uncurry0* (*RETURN c-default*)) ∈ *unit-assn*^{k} →$_a$ *c-assn*› **and**
d: ‹(*uncurry0 d*, *uncurry0* (*RETURN d-default*)) ∈ *unit-assn*^{k} →$_a$ *d-assn*› **and**
e: ‹(*uncurry0 e*, *uncurry0* (*RETURN e-default*)) ∈ *unit-assn*^{k} →$_a$ *e-assn*› **and**
f: ‹(*uncurry0 f*, *uncurry0* (*RETURN f-default*)) ∈ *unit-assn*^{k} →$_a$ *f-assn*› **and**
g: ‹(*uncurry0 g*, *uncurry0* (*RETURN g-default*)) ∈ *unit-assn*^{k} →$_a$ *g-assn*› **and**
h: ‹(*uncurry0 h*, *uncurry0* (*RETURN h-default*)) ∈ *unit-assn*^{k} →$_a$ *h-assn*› **and**
i: ‹(*uncurry0 i*, *uncurry0* (*RETURN i-default*)) ∈ *unit-assn*^{k} →$_a$ *i-assn*› **and**
j: ‹(*uncurry0 j*, *uncurry0* (*RETURN j-default*)) ∈ *unit-assn*^{k} →$_a$ *j-assn*› **and**
k: ‹(*uncurry0 k*, *uncurry0* (*RETURN k-default*)) ∈ *unit-assn*^{k} →$_a$ *k-assn*› **and**
l: ‹(*uncurry0 l*, *uncurry0* (*RETURN l-default*)) ∈ *unit-assn*^{k} →$_a$ *l-assn*› **and**
m: ‹(*uncurry0 m*, *uncurry0* (*RETURN m-default*)) ∈ *unit-assn*^{k} →$_a$ *m-assn*› **and**
n: ‹(*uncurry0 n*, *uncurry0* (*RETURN n-default*)) ∈ *unit-assn*^{k} →$_a$ *n-assn*› **and**
o: ‹(*uncurry0 ko*, *uncurry0* (*RETURN ko-default*)) ∈ *unit-assn*^{k} →$_a$ *o-assn*›**and**
p: ‹(*uncurry0 p*, *uncurry0* (*RETURN p-default*)) ∈ *unit-assn*^{k} →$_a$ *p-assn*› **and**
a-free: ‹*MK-FREE a-assn a-free*› **and**
b-free: ‹*MK-FREE b-assn b-free*›**and**
c-free: ‹*MK-FREE c-assn c-free*›**and**
d-free: ‹*MK-FREE d-assn d-free*›**and**
e-free: ‹*MK-FREE e-assn e-free*›**and**
f-free: ‹*MK-FREE f-assn f-free*›**and**
g-free: ‹*MK-FREE g-assn g-free*›**and**
h-free: ‹*MK-FREE h-assn h-free*›**and**
i-free: ‹*MK-FREE i-assn i-free*›**and**
j-free: ‹*MK-FREE j-assn j-free*›**and**
k-free: ‹*MK-FREE k-assn k-free*›**and**
l-free: ‹*MK-FREE l-assn l-free*›**and**
m-free: ‹*MK-FREE m-assn m-free*›**and**
n-free: ‹*MK-FREE n-assn n-free*›**and**
o-free: ‹*MK-FREE o-assn o-free*›**and**
p-free: ‹*MK-FREE p-assn p-free*›
notes [[*sepref-register-adhoc a-default b-default c-default d-default e-default f-default g-default h-default i-default j-default k-default l-default m-default n-default ko-default p-default*]]
begin

lemmas [*sepref-comb-rules*] = *hn-case-tuple16'*[*of* - *a-assn b-assn c-assn d-assn e-assn f-assn g-assn h-assn i-assn j-assn k-assn l-assn m-assn n-assn o-assn p-assn*, *unfolded isasat-assn-def*[*symmetric*]]

lemma *ex-tuple16-iff*: (∃ *b* :: (-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-)*tuple16*. *P b*) ⟷
(∃ *a b c d e f g h i j k l m n ko p*. *P* (*Tuple16 a b c d e f g h i j k l m n ko p*))
apply *auto*
apply (*case-tac b*)
by *force*

lemmas [*sepref-frame-free-rules*] = *a-free b-free c-free d-free e-free f-free g-free h-free i-free j-free k-free l-free m-free n-free o-free p-free*
sepref-register
‹*Tuple16*›
lemma [*sepref-fr-rules*]: ‹(*uncurry15* (*Mreturn* o_{16} *Tuple16*), *uncurry15* (*RETURN* o_{16} *Tuple16*))
∈ *a-assn*^{d} **$_a$ *b-assn*^{d} **$_a$ *c-assn*^{d} **$_a$ *d-assn*^{d} **$_a$
e-assn^d **$_a$ *f-assn*^{d} **$_a$ *g-assn*^{d} **$_a$ *h-assn*^{d} **$_a$
i-assn^d **$_a$ *j-assn*^{d} **$_a$ *k-assn*^{d} **$_a$ *l-assn*^{d} **$_a$
m-assn^d **$_a$ *n-assn*^{d} **$_a$ *o-assn*^{d} **$_a$ *p-assn*^{d}
→$_a$ *isasat-assn*›
unfolding *isasat-assn-def*
apply *sepref-to-hoare*

apply *vcg*
unfolding *ex-tuple16-iff*
apply *vcg'*
done

lemma [*sepref-frame-match-rules*]:
‹ *hn-ctxt*
(*tuple16-assn* (*invalid-assn a-assn*) (*invalid-assn b-assn*) (*invalid-assn c-assn*) (*invalid-assn d-assn*) (*invalid-assn e-assn*)
(*invalid-assn f-assn*) (*invalid-assn g-assn*) (*invalid-assn h-assn*) (*invalid-assn i-assn*) (*invalid-assn j-assn*) (*invalid-assn k-assn*)
(*invalid-assn l-assn*) (*invalid-assn m-assn*) (*invalid-assn n-assn*) (*invalid-assn o-assn*) (*invalid-assn p-assn*)) *ax bx* ⊢ *hn-val UNIV ax bx*›
unfolding *hn-ctxt-def invalid-assn-def isasat-assn-def entails-def*
apply (*auto split*: *prod.split tuple16.splits elim*: *is-pureE*
simp: *sep-algebra-simps pure-part-pure-conj-eq*)
apply (*auto simp*: *pure-part-def*)
apply (*auto simp*: *pure-def pure-true-conv*)
done

lemma *RETURN-case-tuple16-inverse*: ‹*RETURN*
(*let* - = *M*
in ff) =
(*do* {- ← *mop-free M*;
RETURN (*ff*)})›
by (*auto intro*!: *ext simp*: *mop-free-def split*: *tuple16.splits*)

sepref-def *update-a-code*
is ‹*uncurry* (*RETURN oo update-a*)›
:: ‹*a-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=5*]]
unfolding *update-a-def tuple16.case-distrib comp-def RETURN-case-tuple16-inverse*
by *sepref*

sepref-def *update-b-code*
is ‹*uncurry* (*RETURN oo update-b*)›
:: ‹*b-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-b-def tuple16.case-distrib comp-def RETURN-case-tuple16-inverse*
by *sepref*

sepref-def *update-c-code*
is ‹*uncurry* (*RETURN oo update-c*)›
:: ‹*c-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-c-def tuple16.case-distrib comp-def RETURN-case-tuple16-inverse*
by *sepref*

sepref-def *update-d-code*
is ‹*uncurry* (*RETURN oo update-d*)›
:: ‹*d-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-d-def tuple16.case-distrib comp-def RETURN-case-tuple16-inverse*
by *sepref*

sepref-def *update-e-code*

is ‹*uncurry* (*RETURN oo update-e*)›
:: ‹*e-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-e-def tuple16.case-distrib comp-def RETURN-case-tuple16-inverse*
by *sepref*

sepref-def *update-f-code*
is ‹*uncurry* (*RETURN oo update-f*)›
:: ‹*f-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-f-def tuple16.case-distrib comp-def RETURN-case-tuple16-inverse*
by *sepref*

sepref-def *update-g-code*
is ‹*uncurry* (*RETURN oo update-g*)›
:: ‹*g-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-g-def tuple16.case-distrib comp-def RETURN-case-tuple16-inverse*
by *sepref*

sepref-def *update-h-code*
is ‹*uncurry* (*RETURN oo update-h*)›
:: ‹*h-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-h-def tuple16.case-distrib comp-def RETURN-case-tuple16-inverse*
by *sepref*

sepref-def *update-i-code*
is ‹*uncurry* (*RETURN oo update-i*)›
:: ‹*i-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-i-def tuple16.case-distrib comp-def RETURN-case-tuple16-inverse*
by *sepref*

sepref-def *update-j-code*
is ‹*uncurry* (*RETURN oo update-j*)›
:: ‹*j-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-j-def tuple16.case-distrib comp-def RETURN-case-tuple16-inverse*
by *sepref*

sepref-def *update-k-code*
is ‹*uncurry* (*RETURN oo update-k*)›
:: ‹*k-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-k-def tuple16.case-distrib comp-def RETURN-case-tuple16-inverse*
by *sepref*

sepref-def *update-l-code*
is ‹*uncurry* (*RETURN oo update-l*)›
:: ‹*l-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-l-def tuple16.case-distrib comp-def RETURN-case-tuple16-inverse*
by *sepref*

sepref-def *update-m-code*

is ‹*uncurry* (*RETURN oo update-m*)›
:: ‹*m-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-m-def tuple16.case-distrib comp-def RETURN-case-tuple16-inverse*
by *sepref*

sepref-def *update-n-code*
is ‹*uncurry* (*RETURN oo update-n*)›
:: ‹*n-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-n-def tuple16.case-distrib comp-def RETURN-case-tuple16-inverse*
by *sepref*

sepref-def *update-o-code*
is ‹*uncurry* (*RETURN oo update-o*)›
:: ‹*o-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-o-def tuple16.case-distrib comp-def RETURN-case-tuple16-inverse*
by *sepref*

sepref-def *update-p-code*
is ‹*uncurry* (*RETURN oo update-p*)›
:: ‹*p-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-p-def tuple16.case-distrib comp-def RETURN-case-tuple16-inverse*
by *sepref*

method *stuff-pre* =
sepref-to-hoare;
case-tac x;
vcg;
unfold wpa-return;
subst (*asm*)(*2*) *sep-algebra-class.sep-conj-empty'*[*symmetric*];
rule apply-htriple-rule

method *stuff-post1* =
rule POSTCONDI;
rule STATE-monoI

method *stuff-post2* =
unfold ex-tuple16-iff entails-def;
auto simp: *Exists-eq-simp ex-tuple16-iff entails-def entails-eq-iff pure-true-conv sep-conj-left-commute*;
smt (*z3*) *entails-def entails-eq-iff pure-true-conv sep-conj-aci*(*4*) *sep-conj-aci*(*5*) *sep-conj-left-commute*

lemma *RETURN-case-tuple16-invers*: ‹(*RETURN* ∘∘ *case-tuple16*)
(λ*x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16*.
ff x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16) =
case-tuple16
(λ*x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16*.
RETURN (*ff x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16*))›
by (*auto intro*!: *ext split*: *tuple16.splits*)

lemmas [*sepref-fr-rules*] = *a b c d e f g h i j k l m n o p*

sepref-definition *remove-a-code*
is ‹*RETURN o remove-a*›

:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *a-assn* $\times_a$ *isasat-assn*›
unfolding *remove-a-def RETURN-case-tuple16-invers*
by *sepref*

sepref-definition *remove-b-code*
is ‹*RETURN o remove-b*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *b-assn* $\times_a$ *isasat-assn*›
unfolding *remove-b-def RETURN-case-tuple16-invers*
by *sepref*

sepref-definition *remove-c-code*
is ‹*RETURN o remove-c*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *c-assn* $\times_a$ *isasat-assn*›
unfolding *remove-c-def RETURN-case-tuple16-invers*
by *sepref*

sepref-definition *remove-d-code*
is ‹*RETURN o remove-d*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *d-assn* $\times_a$ *isasat-assn*›
unfolding *remove-d-def RETURN-case-tuple16-invers*
by *sepref*

sepref-definition *remove-e-code*
is ‹*RETURN o remove-e*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *e-assn* $\times_a$ *isasat-assn*›
unfolding *remove-e-def RETURN-case-tuple16-invers*
by *sepref*

sepref-definition *remove-f-code*
is ‹*RETURN o remove-f*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *f-assn* $\times_a$ *isasat-assn*›
unfolding *remove-f-def RETURN-case-tuple16-invers*
by *sepref*

sepref-definition *remove-g-code*
is ‹*RETURN o remove-g*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *g-assn* $\times_a$ *isasat-assn*›
unfolding *remove-g-def RETURN-case-tuple16-invers*
by *sepref*

sepref-definition *remove-h-code*
is ‹*RETURN o remove-h*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *h-assn* $\times_a$ *isasat-assn*›
unfolding *remove-h-def RETURN-case-tuple16-invers*
by *sepref*

sepref-definition *remove-i-code*
is ‹*RETURN o remove-i*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *i-assn* $\times_a$ *isasat-assn*›
unfolding *remove-i-def RETURN-case-tuple16-invers*
by *sepref*

sepref-definition *remove-j-code*
is ‹*RETURN o remove-j*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *j-assn* $\times_a$ *isasat-assn*›
unfolding *remove-j-def RETURN-case-tuple16-invers*

by *sepref*

sepref-definition *remove-k-code*
is ‹*RETURN o remove-k*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *k-assn* $\times_a$ *isasat-assn*›
unfolding *remove-k-def RETURN-case-tuple16-invers*
by *sepref*

sepref-definition *remove-l-code*
is ‹*RETURN o remove-l*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *l-assn* $\times_a$ *isasat-assn*›
unfolding *remove-l-def RETURN-case-tuple16-invers*
by *sepref*

sepref-definition *remove-m-code*
is ‹*RETURN o remove-m*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *m-assn* $\times_a$ *isasat-assn*›
unfolding *remove-m-def RETURN-case-tuple16-invers*
by *sepref*

sepref-definition *remove-n-code*
is ‹*RETURN o remove-n*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *n-assn* $\times_a$ *isasat-assn*›
unfolding *remove-n-def RETURN-case-tuple16-invers*
by *sepref*

sepref-definition *remove-o-code*
is ‹*RETURN o remove-o*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *o-assn* $\times_a$ *isasat-assn*›
unfolding *remove-o-def RETURN-case-tuple16-invers*
by *sepref*

sepref-definition *remove-p-code*
is ‹*RETURN o remove-p*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *p-assn* $\times_a$ *isasat-assn*›
unfolding *remove-p-def RETURN-case-tuple16-invers*
by *sepref*

lemma *remove-a-code-alt-def*: ‹*remove-a-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 0*;
x ← *a*;
x ← *ll-insert-value xi x 0*;
return$_M$ (*M*, *x*)
}› **and**
remove-b-code-alt-def: ‹*remove-b-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 1*;
x ← *b*;
x ← *ll-insert-value xi x 1*;
return$_M$ (*M*, *x*)
}›**and**
remove-c-code-alt-def: ‹*remove-c-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 2*;
x ← *c*;
x ← *ll-insert-value xi x 2*;
return$_M$ (*M*, *x*)

}›**and**
remove-d-code-alt-def: ‹*remove-d-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 3*;
x ← *d*;
x ← *ll-insert-value xi x 3*;
return$_M$ (*M*, *x*)
}›**and**
remove-e-code-alt-def: ‹*remove-e-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 4*;
x ← *e*;
x ← *ll-insert-value xi x 4*;
return$_M$ (*M*, *x*)
}›**and**
remove-f-code-alt-def: ‹*remove-f-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 5*;
x ← *f*;
x ← *ll-insert-value xi x 5*;
return$_M$ (*M*, *x*)
}›**and**
remove-g-code-alt-def: ‹*remove-g-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 6*;
x ← *g*;
x ← *ll-insert-value xi x 6*;
return$_M$ (*M*, *x*)
}›**and**
remove-h-code-alt-def: ‹*remove-h-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 7*;
x ← *h*;
x ← *ll-insert-value xi x 7*;
return$_M$ (*M*, *x*)
}›**and**
remove-i-code-alt-def: ‹*remove-i-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 8*;
x ← *i*;
x ← *ll-insert-value xi x 8*;
return$_M$ (*M*, *x*)
}›**and**
remove-j-code-alt-def: ‹*remove-j-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 9*;
x ← *j*;
x ← *ll-insert-value xi x 9*;
return$_M$ (*M*, *x*)
}›**and**
remove-k-code-alt-def: ‹*remove-k-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 10*;
x ← *k*;
x ← *ll-insert-value xi x 10*;
return$_M$ (*M*, *x*)
}›**and**
remove-l-code-alt-def: ‹*remove-l-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 11*;
x ← *l*;
x ← *ll-insert-value xi x 11*;
return$_M$ (*M*, *x*)
}›**and**
remove-m-code-alt-def: ‹*remove-m-code xi* = *do*$_M$ {

```
            M ← ll-extract-value xi 12;
            x ← m;
            x ← ll-insert-value xi x 12;
            return_M (M, x)
  }›and
  remove-n-code-alt-def: ‹remove-n-code xi = do_M {
            M ← ll-extract-value xi 13;
            x ← n;
            x ← ll-insert-value xi x 13;
            return_M (M, x)
  }›and
  remove-o-code-alt-def: ‹remove-o-code xi = do_M {
            M ← ll-extract-value xi 14;
            x ← ko;
            x ← ll-insert-value xi x 14;
            return_M (M, x)
  }›and
  remove-p-code-alt-def: ‹remove-p-code xi = do_M {
            M ← ll-extract-value xi 15;
            x ← p;
            x ← ll-insert-value xi x 15;
            return_M (M, x)
  }›
  supply [simp] = llvm-extract-value-def
    ll-extract-value-def to-val-tuple16-def
    checked-from-val-def ll-insert-value-def
    llvm-insert-value-def
  unfolding remove-a-code-def remove-b-code-def inline-direct-return-node-case
    remove-c-code-def remove-d-code-def remove-e-code-def remove-f-code-def remove-g-code-def
    remove-h-code-def remove-i-code-def remove-j-code-def remove-k-code-def remove-l-code-def
    remove-m-code-def remove-n-code-def remove-o-code-def remove-p-code-def
  by (solves ‹cases xi, rewrite in ‹Mreturn (-, ⌑)› llvm-rep-class.from-to-id'[symmetric],
    simp flip: from-val-tuple16-def›)+

lemma update-a-code-alt-def: ‹⋀x. update-a-code x xi = do_M {
            M ← ll-extract-value xi 0; a-free M;
            x ← ll-insert-value xi x 0;
            return_M (x)
  }› and
  update-b-code-alt-def: ‹⋀x. update-b-code x xi = do_M {
            M ← ll-extract-value xi 1; b-free M;
            x ← ll-insert-value xi x 1;
            return_M (x)
  }›and
  update-c-code-alt-def: ‹⋀x. update-c-code x xi = do_M {
            M ← ll-extract-value xi 2; c-free M;
            x ← ll-insert-value xi x 2;
            return_M (x)
  }›and
  update-d-code-alt-def: ‹⋀x. update-d-code x xi = do_M {
            M ← ll-extract-value xi 3; d-free M;
            x ← ll-insert-value xi x 3;
            return_M (x)
  }›and
  update-e-code-alt-def: ‹⋀x. update-e-code x xi = do_M {
            M ← ll-extract-value xi 4; e-free M;
```

```
        x ← ll-insert-value xi x 4;
        return_M (x)
}›and
update-f-code-alt-def: ‹⋀x. update-f-code x xi = do_M {
        M ← ll-extract-value xi 5; f-free M;
        x ← ll-insert-value xi x 5;
        return_M (x)
}›and
update-g-code-alt-def: ‹⋀x. update-g-code x xi = do_M {
        M ← ll-extract-value xi 6; g-free M;
        x ← ll-insert-value xi x 6;
        return_M (x)
}›and
update-h-code-alt-def: ‹⋀x. update-h-code x xi = do_M {
        M ← ll-extract-value xi 7; h-free M;
        x ← ll-insert-value xi x 7;
        return_M (x)
}›and
update-i-code-alt-def: ‹⋀x. update-i-code x xi = do_M {
        M ← ll-extract-value xi 8; i-free M;
        x ← ll-insert-value xi x 8;
        return_M (x)
}›and
update-j-code-alt-def: ‹⋀x. update-j-code x xi = do_M {
        M ← ll-extract-value xi 9; j-free M;
        x ← ll-insert-value xi x 9;
        return_M (x)
}›and
update-k-code-alt-def: ‹⋀x. update-k-code x xi = do_M {
        M ← ll-extract-value xi 10; k-free M;
        x ← ll-insert-value xi x 10;
        return_M (x)
}›and
update-l-code-alt-def: ‹⋀x. update-l-code x xi = do_M {
        M ← ll-extract-value xi 11; l-free M;
        x ← ll-insert-value xi x 11;
        return_M (x)
}›and
update-m-code-alt-def: ‹⋀x. update-m-code x xi = do_M {
        M ← ll-extract-value xi 12; m-free M;
        x ← ll-insert-value xi x 12;
        return_M (x)
}›and
update-n-code-alt-def: ‹⋀x. update-n-code x xi = do_M {
        M ← ll-extract-value xi 13; n-free M;
        x ← ll-insert-value xi x 13;
        return_M (x)
}›and
update-o-code-alt-def: ‹⋀x. update-o-code x xi = do_M {
        M ← ll-extract-value xi 14; o-free M;
        x ← ll-insert-value xi x 14;
        return_M (x)
}›and
update-p-code-alt-def: ‹⋀x. update-p-code x xi = do_M {
        M ← ll-extract-value xi 15; p-free M;
        x ← ll-insert-value xi x 15;
```

```
        return_M (x)
  }›
  supply [simp] = llvm-extract-value-def
    ll-extract-value-def to-val-tuple16-def
    checked-from-val-def ll-insert-value-def
    llvm-insert-value-def
  unfolding update-a-code-def update-b-code-def inline-direct-return-node-case
    update-c-code-def update-d-code-def update-e-code-def update-f-code-def update-g-code-def
    update-h-code-def update-i-code-def update-j-code-def update-k-code-def update-l-code-def
    update-m-code-def update-n-code-def update-o-code-def update-p-code-def comp-def
  by (solves ‹cases xi, rewrite in ‹Mreturn (¤)› llvm-rep-class.from-to-id'[symmetric],
    simp flip: from-val-tuple16-def›)+

lemma tuple15-free[sepref-frame-free-rules]:
  shows
  ‹MK-FREE (tuple16-assn a-assn b-assn c-assn d-assn e-assn f-assn g-assn h-assn i-assn j-assn
    k-assn l-assn m-assn n-assn o-assn p-assn) (λS. case S of Tuple16 a b c d e f g h i j k l m n ko p ⇒
do_M {
  a-free a; b-free b; c-free c; d-free d; e-free e; f-free f; g-free g; h-free h; i-free i; j-free j;
  k-free k; l-free l; m-free m; n-free n; o-free ko; p-free p
  })›
 supply [vcg-rules] = a-free[THEN MK-FREED] b-free[THEN MK-FREED] c-free[THEN MK-FREED]
d-free[THEN MK-FREED]
   e-free[THEN MK-FREED] f-free[THEN MK-FREED] g-free[THEN MK-FREED] h-free[THEN MK-FREED]
   i-free[THEN MK-FREED] j-free[THEN MK-FREED] k-free[THEN MK-FREED] l-free[THEN MK-FREED]
     m-free[THEN MK-FREED] n-free[THEN MK-FREED] o-free[THEN MK-FREED] p-free[THEN
MK-FREED]
  apply (rule)+
  apply (clarsimp split: tuple16.splits)
  by vcg'

end


context tuple16
begin
lemma reconstruct-isasat[sepref-frame-match-rules]:
  ‹hn-ctxt
     (tuple16-assn (a-assn) (b-assn) (c-assn) (d-assn) (e-assn)
    (f-assn) (g-assn) (h-assn) (i-assn) (j-assn) (k-assn)
   (l-assn) (m-assn) (n-assn) (o-assn) (p-assn)) ax bx ⊢ hn-ctxt isasat-assn ax bx›
    unfolding isasat-assn-def
    apply (auto split: prod.split tuple16.splits elim: is-pureE
      simp: sep-algebra-simps pure-part-pure-conj-eq)
      done

context
  fixes x-assn :: ‹'r ⇒ 'q ⇒ assn› and
    read-all-code :: ‹'xa ⇒ 'xb ⇒ 'xc ⇒ 'xd ⇒ 'xe ⇒ 'xf ⇒ 'xg ⇒ 'xh ⇒ 'xi ⇒ 'xj ⇒ 'xk ⇒ 'xl ⇒ 'xm
⇒ 'xn ⇒ 'xo ⇒ 'xp ⇒ 'q llM› and
    read-all :: ‹'a ⇒ 'b ⇒ 'c ⇒ 'd ⇒ 'e ⇒ 'f ⇒ 'g ⇒ 'h ⇒ 'i ⇒ 'j ⇒ 'k ⇒ 'l ⇒ 'm ⇒ 'n ⇒ 'o ⇒ 'p
⇒ 'r nres›
begin

definition read-all-st-code :: ‹-› where
  ‹read-all-st-code xi = (case xi of
```

Tuple16 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 ⇒
read-all-code a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16)›

definition *read-all-st* :: ‹('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*,
'*k*, '*l*, '*m*, '*n*, '*o*, '*p*) *tuple16* ⇒ -› **where**
‹*read-all-st t* = (*case t of Tuple16 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16* ⇒
*read-all a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16)›

context
fixes *P*
assumes *trail-read*[*sepref-fr-rules*]: ‹(*uncurry15 read-all-code*, *uncurry15 read-all*) ∈
[*uncurry15 P*]$_a$ *a-assn*^{k} *$*_a$ *b-assn*^{k} *$*_a$ *c-assn*^{k} *$*_a$ *d-assn*^{k} *$*_a$ *e-assn*^{k} *$*_a$ *f-assn*^{k} *$*_a$
g-assn^k *$*_a$ *h-assn*^{k} *$*_a$ *i-assn*^{k} *$*_a$ *j-assn*^{k} *$*_a$ *k-assn*^{k} *$*_a$ *l-assn*^{k} *$*_a$
m-assn^k *$*_a$ *n-assn*^{k} *$*_a$ *o-assn*^{k} *$*_a$ *p-assn*^{k} → *x-assn*›
notes [[*sepref-register-adhoc read-all*]]
begin
sepref-definition *read-all-code-tmp*
is *read-all-st*
:: ‹[*case-tuple16 P*]$_a$ *isasat-assn*^{k} → *x-assn*›
unfolding *read-all-st-def*
by *sepref*

lemmas *read-all-code-refine* =
read-all-code-tmp.refine[*unfolded read-all-code-tmp-def*
read-all-st-code-def[*symmetric*]]
end

end

lemmas [*unfolded Let-def*, *tuple16-getters-setters*] =
update-a-def
update-b-def
update-c-def
update-d-def
update-e-def
update-f-def
update-g-def
update-h-def
update-i-def
update-j-def
update-k-def
update-l-def
update-m-def
update-n-def
update-o-def
update-p-def

remove-a-def
remove-b-def
remove-c-def
remove-d-def
remove-e-def
remove-f-def
remove-g-def
remove-h-def
remove-i-def

remove-j-def
remove-k-def
remove-l-def
remove-m-def
remove-n-def
remove-o-def
remove-p-def

end

lemmas [*tuple16-getters-setters*] =
tuple16-ops.remove-a-def
tuple16-ops.remove-b-def
tuple16-ops.remove-c-def
tuple16-ops.remove-d-def
tuple16-ops.remove-e-def
tuple16-ops.remove-f-def
tuple16-ops.remove-g-def
tuple16-ops.remove-h-def
tuple16-ops.remove-i-def
tuple16-ops.remove-j-def
tuple16-ops.remove-k-def
tuple16-ops.remove-l-def
tuple16-ops.remove-m-def
tuple16-ops.remove-n-def
tuple16-ops.remove-o-def
tuple16-ops.remove-p-def

end
theory *IsaSAT-Stats-LLVM*
imports *IsaSAT-Stats IsaSAT-EMA-LLVM IsaSAT-Rephase-LLVM IsaSAT-Reluctant-LLVM Tuple16-LLVM*
begin

hide-const (**open**) *NEMonad.RETURN NEMonad.ASSERT*

lemma *Exists-eq-simp-sym*: ‹(∃ *x*. (*P x* ∧* ↑ (*b* = *x*)) *s*) ⟷ *P b s*›
by (*subst eq-commute*[*of b*])
(*rule Exists-eq-simp*)

definition *code-hider-assn* **where**
‹*code-hider-assn R S* = *hr-comp R* (⟨*S*⟩*code-hider-rel*)›

lemma *get-content-destroyed-kept*[*sepref-fr-rules*]:
‹*CONSTRAINT is-pure R* ⟹ (*Mreturn o id*, *RETURN o get-content*) ∈ (*code-hider-assn R S*)^k →$_a$ *hr-comp R S*›
unfolding *code-hider-assn-def code-hider-rel-def*
apply *sepref-to-hoare*
apply *vcg*
apply (*auto simp*: *br-def ENTAILS-def Exists-eq-simp Exists-eq-simp-sym hr-comp-def pure-true-conv*)
by (*smt* (*z3*) *Sep-Algebra-Add.pure-part-pure entails-def is-pure-conv pure-app-eq pure-partI sep.mult-assoc sep-conj-def split-conj-is-pure*)

lemma *Constructor-assn-destroyed*:
‹(*Mreturn o id*, *RETURN o Constructor*) ∈ (*hr-comp R S*)^d →$_a$ *code-hider-assn R S*›

unfolding *code-hider-assn-def code-hider-rel-def*
apply *sepref-to-hoare*
apply *vcg*
by (*auto simp*: *br-def ENTAILS-def Exists-eq-simp Exists-eq-simp-sym hr-comp-def pure-true-conv*)

lemma *get-content-destroyed*:
‹(*Mreturn o id*, *RETURN o get-content*) ∈ (*code-hider-assn R S*)^d →$_a$ *hr-comp R S*›
unfolding *code-hider-assn-def code-hider-rel-def*
apply *sepref-to-hoare*
apply *vcg*
by (*auto simp*: *br-def ENTAILS-def Exists-eq-simp Exists-eq-simp-sym hr-comp-def pure-true-conv*)

lemma *get-content-hnr*[*sepref-fr-rules*]:
‹(*id*, *get-content*) ∈ ⟨*S*⟩*code-hider-rel* →$_f$ *S*›
unfolding *code-hider-rel-def*
by (*auto simp*: *intro*!: *frefI*)

lemma *Constructor-hnr*[*sepref-fr-rules*]:
‹(*id*, *Constructor*) ∈ *S* →$_f$ ⟨*S*⟩*code-hider-rel*›
unfolding *code-hider-rel-def*
by (*auto simp*: *intro*!: *frefI*)

definition *search-stats-assn* :: ‹*search-stats* ⇒ *search-stats* ⇒ -› **where**
‹*search-stats-assn* ≡ *word64-assn* ×$_a$ *word64-assn* ×$_a$ *word64-assn* ×$_a$*word64-assn* ×$_a$ *word64-assn* ×$_a$ *word64-assn* ×$_a$*word64-assn* ×$_a$ *word64-assn* ×$_a$ *word64-assn*›

definition *subsumption-stats-assn* :: ‹*inprocessing-subsumption-stats* ⇒ *inprocessing-subsumption-stats* ⇒ -› **where**
‹*subsumption-stats-assn* = *word64-assn* ×$_a$ *word64-assn* ×$_a$ *word64-assn* ×$_a$ *word64-assn* ×$_a$ *word64-assn*›

definition *binary-stats-assn* :: ‹*inprocessing-binary-stats* ⇒ *inprocessing-binary-stats* ⇒ -› **where**
‹*binary-stats-assn* = *word64-assn* ×$_a$ *word64-assn* ×$_a$ *word64-assn*›

definition *pure-lits-stats-assn* :: ‹*inprocessing-pure-lits-stats* ⇒ *inprocessing-pure-lits-stats* ⇒ -› **where**
‹*pure-lits-stats-assn* = *word64-assn* ×$_a$ *word64-assn*›

definition *rephase-stats-assn* :: ‹*rephase-stats* ⇒ *rephase-stats* ⇒ -› **where**
‹*rephase-stats-assn* ≡ *word64-assn* ×$_a$ *word64-assn* ×$_a$ *word64-assn* ×$_a$*word64-assn* ×$_a$ *word64-assn* ×$_a$ *word64-assn*›

definition *empty-search-stats* **where**
‹*empty-search-stats* = (*0*,*0*,*0*,*0*,*0*,*0*,*0*,*0*,*0*)›

sepref-def *empty-search-stats-impl*
is ‹*uncurry0* (*RETURN empty-search-stats*)›
:: ‹*unit-assn*^{k} →$_a$ *search-stats-assn*›
unfolding *search-stats-assn-def empty-search-stats-def*
by *sepref*

definition *empty-binary-stats* :: ‹*inprocessing-binary-stats*› **where**
‹*empty-binary-stats* = (*0*,*0*,*0*)›

sepref-def *empty-binary-stats-impl*
is ‹*uncurry0* (*RETURN empty-binary-stats*)›

:: ‹*unit-assn*^{k} $\rightarrow_a$ *binary-stats-assn*›
unfolding *binary-stats-assn-def empty-binary-stats-def*
by *sepref*

definition *empty-subsumption-stats* :: ‹*inprocessing-subsumption-stats*› **where**
‹*empty-subsumption-stats* = (*0*,*0*,*0*,*0*,*0*)›

sepref-def *empty-subsumption-stats-impl*
is ‹*uncurry0* (*RETURN empty-subsumption-stats*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *subsumption-stats-assn*›
unfolding *subsumption-stats-assn-def empty-subsumption-stats-def*
by *sepref*

definition *empty-pure-lits-stats* :: ‹*inprocessing-pure-lits-stats*› **where**
‹*empty-pure-lits-stats* = (*0*,*0*)›

sepref-def *empty-pure-lits-stats-impl*
is ‹*uncurry0* (*RETURN empty-pure-lits-stats*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *pure-lits-stats-assn*›
unfolding *pure-lits-stats-assn-def empty-pure-lits-stats-def*
by *sepref*

definition *lbd-size-limit-assn* **where**
‹*lbd-size-limit-assn* = *uint32-nat-assn* $\times_a$ *sint64-nat-assn*›

definition *empty-lsize-limit-stats* :: ‹*lbd-size-limit-stats*› **where**
‹*empty-lsize-limit-stats* = (*0*,*0*)›

sepref-def *empty-lsize-limit-stats-impl*
is ‹*uncurry0* (*RETURN empty-lsize-limit-stats*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *lbd-size-limit-assn*›
unfolding *lbd-size-limit-assn-def empty-lsize-limit-stats-def*
apply (*rewrite at* ‹(-, ⌑)› *snat-const-fold*[**where** *'a=64*])
apply (*rewrite at* ‹(⌑, -)› *unat-const-fold*[**where** *'a=32*])
by *sepref*

definition *ema-init-bottom* :: ‹*ema*› **where**
‹*ema-init-bottom* = *ema-init 0*›

sepref-def *ema-init-bottom-impl*
is ‹*uncurry0* (*RETURN ema-init-bottom*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *ema-assn*›
unfolding *ema-init-bottom-def* **by** *sepref*

lemma *stats-bottom*:
‹(*uncurry0* (*return*$_M$ *0*), *uncurry0* (*RETURN 0*)) ∈ *unit-assn*^{k} $\rightarrow_a$ *word64-assn*›
‹(*uncurry0* (*return*$_M$ *0*), *uncurry0* (*RETURN 0*)) ∈ *unit-assn*^{k} $\rightarrow_a$ *word32-assn*›
by (*sepref-to-hoare*; *vcg*; *fail*)+

definition *empty-rephase-stats* :: ‹*rephase-stats*› **where**
‹*empty-rephase-stats* = (*0*,*0*,*0*,*0*,*0*,*0*)›

sepref-def *empty-rephase-stats-impl*
is ‹*uncurry0* (*RETURN empty-rephase-stats*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *rephase-stats-assn*›
unfolding *empty-rephase-stats-def rephase-stats-assn-def* **by** *sepref*

schematic-goal *mk-free-search-stats-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE search-stats-assn ?fr*› **and**
mk-free-binary-stats-assn[*sepref-frame-free-rules*]: ‹*MK-FREE binary-stats-assn ?fr2*› **and**
mk-free-subsumption-stats-assn[*sepref-frame-free-rules*]: ‹*MK-FREE subsumption-stats-assn ?fr3*› **and**
mk-free-ema-assn[*sepref-frame-free-rules*]: ‹*MK-FREE ema-assn ?fr4*›**and**
mk-free-pure-lits-stats-assn[*sepref-frame-free-rules*]: ‹*MK-FREE pure-lits-stats-assn ?fr5*› **and**
mk-free-rephase-stats-assn[*sepref-frame-free-rules*]: ‹*MK-FREE rephase-stats-assn ?fr6*›
unfolding *search-stats-assn-def binary-stats-assn-def subsumption-stats-assn-def*
pure-lits-stats-assn-def rephase-stats-assn-def
by *synthesize-free+*

sepref-def *free-search-stats-assn*
is ‹*mop-free*›
:: ‹*search-stats-assn*^{d} →$_a$ *unit-assn*›
by *sepref*

sepref-def *free-binary-stats-assn*
is ‹*mop-free*›
:: ‹*binary-stats-assn*^{d} →$_a$ *unit-assn*›
by *sepref*

sepref-def *free-subsumption-stats-assn*
is ‹*mop-free*›
:: ‹*subsumption-stats-assn*^{d} →$_a$ *unit-assn*›
by *sepref*

sepref-def *free-pure-lits-stats-assn*
is ‹*mop-free*›
:: ‹*pure-lits-stats-assn*^{d} →$_a$ *unit-assn*›
by *sepref*

sepref-def *free-ema-assn*
is ‹*mop-free*›
:: ‹*ema-assn*^{d} →$_a$ *unit-assn*›
by *sepref*

sepref-def *free-word64-assn*
is ‹*mop-free*›
:: ‹*word64-assn*^{d} →$_a$ *unit-assn*›
by *sepref*

sepref-def *free-word32-assn*
is ‹*mop-free*›
:: ‹*word32-assn*^{d} →$_a$ *unit-assn*›
by *sepref*

sepref-def *free-lbd-size-limit-assn*
is ‹*mop-free*›
:: ‹*lbd-size-limit-assn*^{d} →$_a$ *unit-assn*›
unfolding *lbd-size-limit-assn-def*
by *sepref*

sepref-def *free-rephase-stats-assn*
is ‹*mop-free*›
:: ‹*rephase-stats-assn*^{d} →$_a$ *unit-assn*›

by *sepref*

lemma *mop-free-hnr'*: ‹(*f*, *mop-free*) ∈ R^d →$_a$ *unit-assn* ⟹ *MK-FREE R f*›
unfolding *mop-free-def*
apply (*rule MK-FREEI*)
apply *simp*
subgoal for *a c*
apply (*drule hfrefD*[*of* - - - - - - - *a c*])
apply (*rule TrueI*)
apply (*rule TrueI*)
apply (*drule hn-refineD*)
apply *simp*
apply (*rule htriple-pure-preI*[*of*])
unfolding *fst-conv snd-conv hf-pres.simps*
apply (*clarsimp*
simp: *hn-ctxt-def pure-def sep-algebra-simps invalid-assn-def*)
done
done

type-synonym *isasat-stats-assn* = ‹(*search-stats*, *inprocessing-binary-stats*, *inprocessing-subsumption-stats*, *ema*,
inprocessing-pure-lits-stats, *32 word* × *64 word*, *rephase-stats*, *64 word*,
64 word, *64 word*,*64 word*, *64 word*,
64 word, *64 word*, *32 word*, *64 word*) *tuple16*›

definition *isasat-stats-assn* :: ‹*isasat-stats* ⇒ *isasat-stats-assn* ⇒ - ⇒ *bool*› **where**
‹*isasat-stats-assn* = *tuple16-assn search-stats-assn binary-stats-assn subsumption-stats-assn ema-assn pure-lits-stats-assn lbd-size-limit-assn rephase-stats-assn word64-assn word64-assn word64-assn word64-assn word64-assn word64-assn word64-assn word32-assn word64-assn*›

definition *extract-search-strategy-stats* :: ‹*isasat-stats* ⇒ -› **where**
‹*extract-search-strategy-stats* = *tuple16-ops.remove-a empty-search-stats*›

definition *extract-binary-stats* :: ‹*isasat-stats* ⇒ -› **where**
‹*extract-binary-stats* = *tuple16-ops.remove-b empty-binary-stats*›

definition *extract-subsumption-stats* :: ‹*isasat-stats* ⇒ -› **where**
‹*extract-subsumption-stats* = *tuple16-ops.remove-c empty-subsumption-stats*›

definition *extract-avg-lbd* :: ‹*isasat-stats* ⇒ -› **where**
‹*extract-avg-lbd* = *tuple16-ops.remove-d ema-init-bottom*›

definition *extract-pure-lits-stats* :: ‹*isasat-stats* ⇒ -› **where**
‹*extract-pure-lits-stats* = *tuple16-ops.remove-e empty-pure-lits-stats*›

definition *extract-lbd-size-limit-stats* :: ‹*isasat-stats* ⇒ -› **where**
‹*extract-lbd-size-limit-stats* = *tuple16-ops.remove-f empty-lsize-limit-stats*›

definition *extract-rephase-stats* :: ‹*isasat-stats* ⇒ -› **where**
‹*extract-rephase-stats* = *tuple16-ops.remove-g empty-rephase-stats*›

global-interpretation *tuple16* **where**
a-assn = *search-stats-assn* **and**
b-assn = *binary-stats-assn* **and**
c-assn = *subsumption-stats-assn* **and**

d-assn = *ema-assn* **and**
e-assn = *pure-lits-stats-assn* **and**
f-assn = *lbd-size-limit-assn* **and**
g-assn = *rephase-stats-assn* **and**
h-assn = *word64-assn* **and**
i-assn = *word64-assn* **and**
j-assn = *word64-assn* **and**
k-assn = *word64-assn* **and**
l-assn = *word64-assn* **and**
m-assn = *word64-assn* **and**
n-assn = *word64-assn* **and**
o-assn = *word32-assn* **and**
p-assn = *word64-assn* **and**
a-default = *empty-search-stats* **and**
a = *empty-search-stats-impl* **and**
b-default = *empty-binary-stats* **and**
b = *empty-binary-stats-impl* **and**
c-default = *empty-subsumption-stats* **and**
c = *empty-subsumption-stats-impl* **and**
d-default = *ema-init-bottom* **and**
d = *ema-init-bottom-impl* **and**
e-default = *empty-pure-lits-stats* **and**
e = *empty-pure-lits-stats-impl* **and**
f-default = ‹*empty-lsize-limit-stats*› **and**
f = ‹*empty-lsize-limit-stats-impl*› **and**
g-default = ‹*empty-rephase-stats*› **and**
g = ‹*empty-rephase-stats-impl*› **and**
h-default = ‹*0*› **and**
h = ‹*Mreturn 0*› **and**
i-default = ‹*0*› **and**
i = ‹*Mreturn 0*› **and**
j-default = ‹*0*› **and**
j = ‹*Mreturn 0*› **and**
k-default = ‹*0*› **and**
k = ‹*Mreturn 0*› **and**
l-default = ‹*0*› **and**
l = ‹*Mreturn 0*› **and**
m-default = ‹*0*› **and**
m = ‹*Mreturn 0*› **and**
n-default = ‹*0*› **and**
n = ‹*Mreturn 0*› **and**
ko-default = ‹*0*› **and**
ko = ‹*Mreturn 0*› **and**
p-default = ‹*0*› **and**
p = ‹*Mreturn 0*› **and**
a-free = *free-search-stats-assn* **and**
b-free = *free-binary-stats-assn* **and**
c-free = *free-subsumption-stats-assn* **and**
d-free = *free-ema-assn* **and**
e-free = *free-pure-lits-stats-assn* **and**
f-free = *free-lbd-size-limit-assn* **and**
g-free = *free-rephase-stats-assn* **and**
h-free = *free-word64-assn* **and**
i-free = *free-word64-assn* **and**
j-free = *free-word64-assn* **and**
k-free = *free-word64-assn* **and**

l-free = *free-word64-assn* **and**
m-free = *free-word64-assn* **and**
n-free = *free-word64-assn* **and**
o-free = *free-word32-assn* **and**
p-free = *free-word64-assn*
rewrites ‹*isasat-assn* = *isasat-stats-assn*› **and**
‹*remove-a* = *extract-search-strategy-stats*› **and**
‹*remove-b* = *extract-binary-stats*› **and**
‹*remove-c* = *extract-subsumption-stats*› **and**
‹*remove-d* = *extract-avg-lbd*› **and**
‹*remove-e* = *extract-pure-lits-stats*› **and**
‹*remove-f* = *extract-lbd-size-limit-stats*› **and**
‹*remove-g* = *extract-rephase-stats*›
apply *unfold-locales*
apply (*rule empty-search-stats-impl.refine empty-binary-stats-impl.refine*
empty-subsumption-stats-impl.refine ema-init-bottom-impl.refine empty-pure-lits-stats-impl.refine
stats-bottom free-search-stats-assn.refine[*THEN mop-free-hnr'*]
free-binary-stats-assn.refine[*THEN mop-free-hnr'*]
free-subsumption-stats-assn.refine[*THEN mop-free-hnr'*]
free-ema-assn.refine[*THEN mop-free-hnr'*]
free-pure-lits-stats-assn.refine[*THEN mop-free-hnr'*]
empty-lsize-limit-stats-impl.refine
free-pure-lits-stats-assn.refine[*THEN mop-free-hnr'*, *unfolded lbd-size-limit-assn-def*[*symmetric*] *pure-lits-stats-assn-def*
free-word64-assn.refine[*THEN mop-free-hnr'*]
free-word32-assn.refine[*THEN mop-free-hnr'*]
free-lbd-size-limit-assn.refine[*THEN mop-free-hnr'*]
free-rephase-stats-assn.refine[*THEN mop-free-hnr'*]
empty-rephase-stats-impl.refine
)+
subgoal unfolding *isasat-stats-assn-def tuple16-ops.isasat-assn-def* **..**
subgoal unfolding *extract-search-strategy-stats-def* **..**
subgoal unfolding *extract-binary-stats-def* **..**
subgoal unfolding *extract-subsumption-stats-def* **..**
subgoal unfolding *extract-avg-lbd-def* **..**
subgoal unfolding *extract-pure-lits-stats-def* **..**
subgoal unfolding *extract-lbd-size-limit-stats-def* **..**
subgoal unfolding *extract-rephase-stats-def* **..**
done

sepref-register
remove-a remove-b remove-c remove-d
remove-e remove-f remove-g remove-h
remove-i remove-j remove-k remove-l
remove-m remove-n remove-o remove-p

fun *tuple16-rel* :: ‹
('*a* × -) *set* ⇒
('*b* × -) *set* ⇒
('*c* × -) *set* ⇒
('*d* × -) *set* ⇒
('*e* × -) *set* ⇒
('*f* × -) *set* ⇒
('*g* × -) *set* ⇒
('*h* × -) *set* ⇒
('*i* × -) *set* ⇒

('j × -) set ⇒
('k × -) set ⇒
('l × -) set ⇒
('m × -) set ⇒
('n × -) set ⇒
('o × -) set ⇒
('p × -) set ⇒
(('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p) tuple16 × -)set› **where**
‹tuple16-rel a-assn b-assn' c-assn d-assn e-assn f-assn g-assn h-assn i-assn j-assn k-assn l-assn m-assn n-assn o-assn p-assn =
{(S, T). case (S, T) of
(Tuple16 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena,
Tuple16 M' N' D' i' W' ivmtf' icount' ccach' lbd' outl' heur' stats' aivdom' clss' opts' arena')
⇒
((M, M')∈a-assn ∧ (N, N')∈b-assn' ∧ (D, D')∈c-assn ∧ (i,i')∈d-assn ∧
(W, W')∈e-assn ∧ (ivmtf, ivmtf')∈f-assn ∧ (icount, icount')∈ g-assn ∧ (ccach,ccach')∈ h-assn∧
(lbd,lbd')∈i-assn ∧ (outl,outl')∈ j-assn∧ (heur,heur')∈k-assn ∧ (stats,stats')∈ l-assn∧
(aivdom,aivdom')∈ m-assn∧ (clss,clss')∈ n-assn∧ (opts,opts')∈o-assn ∧ (arena,arena')∈p-assn)
}›

lemma *tuple16-assn-tuple16-rel*:
‹tuple16-assn (pure A) (pure B)(pure C)(pure D)(pure E)(pure F)(pure G)(pure H)(pure I)(pure J)(pure K)(pure L)(pure M)(pure N)(pure KO)(pure P) =
pure (tuple16-rel A B C D E F G H I J K L M N KO P)›
by (auto intro!: ext simp: pure-def import-param-3(1) split: tuple16.splits)

lemma *pure-keep-detroy*: $(pure\ R)^k = (pure\ R)^d$
by (auto intro!: ext simp: invalid-pure-recover)
lemma *is-pure R* $\Longrightarrow R^k = R^d$
by (metis is-pure-conv pure-keep-detroy)

lemma *isasat-stats-assn-pure-keep*:
‹$isasat\text{-}stats\text{-}assn^d = isasat\text{-}stats\text{-}assn^k$›
unfolding isasat-stats-assn-def tuple16-assn-tuple16-rel search-stats-assn-def lbd-size-limit-assn-def prod-assn-pure-conv pure-lits-stats-assn-def subsumption-stats-assn-def rephase-stats-assn-def binary-stats-assn-def pure-lits-stats-assn-def pure-keep-detroy **..**

lemmas [unfolded isasat-stats-assn-pure-keep, sepref-fr-rules] =
remove-a-code.refine
remove-b-code.refine
remove-c-code.refine
remove-d-code.refine
remove-e-code.refine
remove-f-code.refine
remove-g-code.refine

named-theorems stats-extractors ‹Definition of all functions modifying the state›

lemmas [stats-extractors] =
extract-search-strategy-stats-def
extract-binary-stats-def
extract-subsumption-stats-def
extract-avg-lbd-def
extract-pure-lits-stats-def

tuple16-ops.remove-a-def
tuple16-ops.remove-b-def
tuple16-ops.remove-c-def
tuple16-ops.remove-d-def
tuple16-ops.remove-e-def
tuple16-ops.remove-f-def
tuple16-ops.remove-g-def
tuple16-ops.update-a-def
tuple16-ops.update-b-def
tuple16-ops.update-c-def
tuple16-ops.update-d-def
tuple16-ops.update-e-def
tuple16-ops.update-f-def
tuple16-ops.update-g-def

We do some cheating to simplify code generation, instead of using our alternative definitions as for the states.

lemma *stats-code-unfold*:
‹*get-search-stats x = fst (extract-search-strategy-stats x)*›
‹*get-binary-stats x = fst (extract-binary-stats x)*›
‹*get-subsumption-stats x = fst (extract-subsumption-stats x)*›
‹*get-pure-lits-stats x = fst (extract-pure-lits-stats x)*›
‹*get-avg-lbd-stats x = fst (extract-avg-lbd x)*›
‹*get-lsize-limit-stats x = fst (extract-lbd-size-limit-stats x)*›
‹*get-rephase-stats x = fst (extract-rephase-stats x)*›
‹*set-propagation-stats a x = update-a a x*›
‹*set-binary-stats b x = update-b b x*›
‹*set-subsumption-stats c x = update-c c x*›
‹*set-avg-lbd-stats lbd x = update-d lbd x*›
‹*set-pure-lits-stats e x = update-e e x*›
‹*set-lsize-limit-stats f x = update-f f x*›
‹*set-rephase-stats g x = update-g g x*›
by (*cases x*; *auto simp*: *get-search-stats-def get-avg-lbd-stats-def*
set-avg-lbd-stats-def set-propagation-stats-def set-binary-stats-def get-rephase-stats-def
set-subsumption-stats-def set-pure-lits-stats-def get-lsize-limit-stats-def
extract-lbd-size-limit-stats-def set-rephase-stats-def extract-rephase-stats-def
get-subsumption-stats-def get-pure-lits-stats-def set-lsize-limit-stats-def
get-binary-stats-def stats-extractors; *fail*)+

lemma *Mreturn-comp-Tuple16*:
‹(*Mreturn* o_{16} *Tuple16*) *a b c d e f g h i j k l m n ko p* =
Mreturn (*Tuple16 a b c d e f g h i j k l m n ko p*)›
by *auto*

lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
remove-a-code-alt-def[*unfolded tuple16.remove-a-code-alt-def Mreturn-comp-Tuple16*]
remove-b-code-alt-def[*unfolded tuple16.remove-b-code-alt-def Mreturn-comp-Tuple16*]
remove-c-code-alt-def[*unfolded tuple16.remove-c-code-alt-def Mreturn-comp-Tuple16*]
remove-d-code-alt-def[*unfolded tuple16.remove-d-code-alt-def Mreturn-comp-Tuple16*]
remove-e-code-alt-def[*unfolded tuple16.remove-e-code-alt-def Mreturn-comp-Tuple16*]
remove-f-code-alt-def[*unfolded tuple16.remove-f-code-alt-def Mreturn-comp-Tuple16*]
remove-g-code-alt-def[*unfolded tuple16.remove-g-code-alt-def Mreturn-comp-Tuple16*]

lemma [*safe-constraint-rules*]: ‹*CONSTRAINT is-pure isasat-stats-assn*›
unfolding *isasat-stats-assn-def tuple16-assn-tuple16-rel search-stats-assn-def*
prod-assn-pure-conv pure-lits-stats-assn-def subsumption-stats-assn-def lbd-size-limit-assn-def

binary-stats-assn-def pure-lits-stats-assn-def pure-keep-detroy rephase-stats-assn-def **by** *auto*

lemma *id-unat*[*sepref-fr-rules*]:
‹(*Mreturn o id*, *RETURN o unat*) ∈ *word32-assn*^{k} →$_a$ *uint32-nat-assn*›
apply *sepref-to-hoare*
apply *vcg*
by (*auto simp*: *ENTAILS-def unat-rel-def unat.rel-def br-def pred-lift-merge-simps pred-lift-def pure-true-conv*)

lemma [*sepref-fr-rules*]:
‹*CONSTRAINT is-pure B* ⟹ (*Mreturn o* (λ(*a*,*b*). *a*), *RETURN o fst*) ∈ (*A* ×$_a$ *B*)^d →$_a$ *A*›
apply *sepref-to-hoare*
apply *vcg*
apply (*auto simp*: *ENTAILS-def*)
by (*smt* (*verit*, *best*) *entails-def fri-startI-extended*(*3*) *is-pure-iff-pure-assn pure-part-pure-eq pure-part-split-conj pure-true-conv sep-conj-aci*(*5*))

sepref-def *Search-Stats-conflicts-impl*
is ‹*RETURN o Search-Stats-conflicts*›
:: ‹*search-stats-assn*^{k} →$_a$ *word64-assn*›
unfolding *search-stats-assn-def Search-Stats-conflicts-def*
by *sepref*

sepref-def *Search-Stats-units-since-gcs-impl*
is ‹*RETURN o Search-Stats-units-since-gcs*›
:: ‹*search-stats-assn*^{k} →$_a$ *word64-assn*›
unfolding *search-stats-assn-def Search-Stats-units-since-gcs-def*
by *sepref*

sepref-def *Search-Stats-reset-units-since-gc-impl*
is ‹*RETURN o Search-Stats-reset-units-since-gc*›
:: ‹*search-stats-assn*^{k} →$_a$ *search-stats-assn*›
unfolding *search-stats-assn-def Search-Stats-reset-units-since-gc-def*
by *sepref*

sepref-def *Search-Stats-fixed-impl*
is ‹*RETURN o Search-Stats-fixed*›
:: ‹*search-stats-assn*^{k} →$_a$ *word64-assn*›
unfolding *search-stats-assn-def Search-Stats-fixed-def*
by *sepref*

sepref-def *add-lbd-stats-impl*
is ‹*uncurry* (*RETURN oo add-lbd*)›
:: ‹*word32-assn*^{k} *$_a$ *isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
unfolding *add-lbd-def stats-code-unfold*
by *sepref*

sepref-def *get-conflict-count-stats-impl*
is ‹(*RETURN o get-conflict-count*)›
:: ‹*isasat-stats-assn*^{k} →$_a$ *word-assn*›
unfolding *stats-conflicts-def stats-code-unfold*
by *sepref*

sepref-def *units-since-last-GC-stats-impl*
is ‹(*RETURN o units-since-last-GC*)›

:: ‹*isasat-stats-assn*^{k} $\rightarrow_a$ *word-assn*›
unfolding *units-since-last-GC-def stats-code-unfold*
by *sepref*

sepref-def *reset-units-since-last-GC-stats-impl*
is ‹(*RETURN o reset-units-since-last-GC*)›
:: ‹*isasat-stats-assn*^{d} $\rightarrow_a$ *isasat-stats-assn*›
unfolding *reset-units-since-last-GC-def stats-code-unfold*
by *sepref*

sepref-def *Search-Stats-incr-irred-impl*
is ‹*RETURN o Search-Stats-incr-irred*›
:: ‹*search-stats-assn*^{k} $\rightarrow_a$ *search-stats-assn*›
unfolding *search-stats-assn-def Search-Stats-incr-irred-def*
by *sepref*

sepref-def *Search-Stats-decr-irred-impl*
is ‹*RETURN o Search-Stats-decr-irred*›
:: ‹*search-stats-assn*^{k} $\rightarrow_a$ *search-stats-assn*›
unfolding *search-stats-assn-def Search-Stats-decr-irred-def*
by *sepref*

sepref-def *Search-Stats-incr-propagation-impl*
is ‹*RETURN o Search-Stats-incr-propagation*›
:: ‹*search-stats-assn*^{k} $\rightarrow_a$ *search-stats-assn*›
unfolding *search-stats-assn-def Search-Stats-incr-propagation-def*
by *sepref*

sepref-def *Search-Stats-incr-propagation-by-impl*
is ‹*uncurry* (*RETURN oo Search-Stats-incr-propagation-by*)›
:: ‹*word64-assn*^{k} $*_a$ *search-stats-assn*^{k} $\rightarrow_a$ *search-stats-assn*›
unfolding *search-stats-assn-def Search-Stats-incr-propagation-by-def*
by *sepref*

sepref-def *Search-Stats-set-not-conflict-until-impl*
is ‹*uncurry* (*RETURN oo Search-Stats-set-no-conflict-until*)›
:: ‹*word64-assn*^{k} $*_a$ *search-stats-assn*^{k} $\rightarrow_a$ *search-stats-assn*›
unfolding *search-stats-assn-def Search-Stats-set-no-conflict-until-def*
by *sepref*

sepref-def *Search-Stats-no-conflict-until-impl*
is ‹*RETURN o Search-Stats-no-conflict-until*›
:: ‹*search-stats-assn*^{k} $\rightarrow_a$ *word64-assn*›
unfolding *search-stats-assn-def Search-Stats-no-conflict-until-def*
by *sepref*

sepref-def *Search-Stats-incr-conflicts-impl*
is ‹*RETURN o Search-Stats-incr-conflicts*›
:: ‹*search-stats-assn*^{k} $\rightarrow_a$ *search-stats-assn*›
unfolding *search-stats-assn-def Search-Stats-incr-conflicts-def*
by *sepref*

sepref-def *Search-Stats-incr-decisions-impl*
is ‹*RETURN o Search-Stats-incr-decisions*›
:: ‹*search-stats-assn*^{k} $\rightarrow_a$ *search-stats-assn*›
unfolding *search-stats-assn-def Search-Stats-incr-decisions-def*

by *sepref*

sepref-def *Search-Stats-incr-restarts-impl*
is ‹*RETURN o Search-Stats-incr-restarts*›
:: ‹*search-stats-assn*^{k} →$_a$ *search-stats-assn*›
unfolding *search-stats-assn-def Search-Stats-incr-restarts-def*
by *sepref*

sepref-def *Search-Stats-incr-reductions-impl*
is ‹*RETURN o Search-Stats-incr-reductions*›
:: ‹*search-stats-assn*^{k} →$_a$ *search-stats-assn*›
unfolding *search-stats-assn-def Search-Stats-incr-reductions-def*
by *sepref*

sepref-def *Search-Stats-incr-fixed-impl*
is ‹*RETURN o Search-Stats-incr-fixed*›
:: ‹*search-stats-assn*^{k} →$_a$ *search-stats-assn*›
unfolding *search-stats-assn-def Search-Stats-incr-fixed-def*
by *sepref*

sepref-def *Search-Stats-incr-fixed-by-impl*
is ‹*uncurry* (*RETURN oo Search-Stats-incr-fixed-by*)›
:: ‹*word64-assn*^{k} ×$_a$ *search-stats-assn*^{k} →$_a$ *search-stats-assn*›
unfolding *search-stats-assn-def Search-Stats-incr-fixed-by-def*
by *sepref*

sepref-def *Search-Stats-incr-gcs-impl*
is ‹*RETURN o Search-Stats-incr-gcs*›
:: ‹*search-stats-assn*^{k} →$_a$ *search-stats-assn*›
unfolding *search-stats-assn-def Search-Stats-incr-gcs-def*
by *sepref*

sepref-def *Search-Stats-incr-gcs-by-impl*
is ‹*uncurry* (*RETURN oo Search-Stats-incr-units-since-gc-by*)›
:: ‹*word64-assn*^{k} ×$_a$ *search-stats-assn*^{k} →$_a$ *search-stats-assn*›
unfolding *search-stats-assn-def Search-Stats-incr-units-since-gc-by-def*
by *sepref*

sepref-def *Search-Stats-incr-units-since-gc-impl*
is ‹*RETURN o Search-Stats-incr-units-since-gc*›
:: ‹*search-stats-assn*^{k} →$_a$ *search-stats-assn*›
unfolding *search-stats-assn-def Search-Stats-incr-units-since-gc-def*
by *sepref*

sepref-def *Pure-lits-Stats-incr-rounds-impl*
is ‹*RETURN o Pure-lits-Stats-incr-rounds*›
:: ‹*pure-lits-stats-assn*^{k} →$_a$ *pure-lits-stats-assn*›
unfolding *pure-lits-stats-assn-def Pure-lits-Stats-incr-rounds-def*
by *sepref*

sepref-def *Pure-lits-Stats-incr-removed-impl*
is ‹*RETURN o Pure-lits-Stats-incr-removed*›
:: ‹*pure-lits-stats-assn*^{k} →$_a$ *pure-lits-stats-assn*›
unfolding *pure-lits-stats-assn-def Pure-lits-Stats-incr-removed-def*
by *sepref*

sepref-def *Binary-Stats-incr-units-impl*
is ‹*RETURN o Binary-Stats-incr-units*›
:: ‹*binary-stats-assn*^{k} →$_a$ *binary-stats-assn*›
unfolding *binary-stats-assn-def Binary-Stats-incr-units-def*
by *sepref*

sepref-def *Binary-Stats-incr-removed-def*
is ‹*RETURN o Binary-Stats-incr-removed*›
:: ‹*binary-stats-assn*^{k} →$_a$ *binary-stats-assn*›
unfolding *Binary-Stats-incr-removed-def binary-stats-assn-def*
by *sepref*

sepref-def *Search-Stats-restarts-impl*
is ‹*RETURN o Search-Stats-restarts*›
:: ‹*search-stats-assn*^{k} →$_a$ *word64-assn*›
unfolding *search-stats-assn-def Search-Stats-restarts-def*
by *sepref*

sepref-def *Search-Stats-reductions-impl*
is ‹*RETURN o Search-Stats-reductions*›
:: ‹*search-stats-assn*^{k} →$_a$ *word64-assn*›
unfolding *search-stats-assn-def Search-Stats-reductions-def*
by *sepref*

sepref-def *Search-Stats-irred-impl*
is ‹*RETURN o Search-Stats-irred*›
:: ‹*search-stats-assn*^{k} →$_a$ *word64-assn*›
unfolding *search-stats-assn-def Search-Stats-irred-def*
by *sepref*

sepref-def *Search-Stats-propagations-impl*
is ‹*RETURN o Search-Stats-propagations*›
:: ‹*search-stats-assn*^{k} →$_a$ *word64-assn*›
unfolding *search-stats-assn-def Search-Stats-propagations-def*
by *sepref*

sepref-def *Search-Stats-gcs-impl*
is ‹*RETURN o Search-Stats-gcs*›
:: ‹*search-stats-assn*^{k} →$_a$ *word64-assn*›
unfolding *search-stats-assn-def Search-Stats-gcs-def*
by *sepref*

sepref-register *Search-Stats-fixed Search-Stats-incr-irred Search-Stats-decr-irred*
Search-Stats-incr-propagation Search-Stats-incr-conflicts
Search-Stats-incr-decisions Search-Stats-incr-restarts
Search-Stats-incr-reductions Search-Stats-incr-fixed Search-Stats-incr-gcs
Pure-lits-Stats-incr-rounds Pure-lits-Stats-incr-removed
Binary-Stats-incr-removed Binary-Stats-incr-units
Search-Stats-reductions Search-Stats-restarts
Search-Stats-irred Search-Stats-propagations Search-Stats-gcs

sepref-def *Search-Stats-decisions-impl*
is ‹*RETURN o Search-Stats-decisions*›
:: ‹*search-stats-assn*^{k} →$_a$ *word64-assn*›
unfolding *search-stats-assn-def Search-Stats-decisions-def*
by *sepref*

sepref-def *Binary-stats-rounds-impl*
  is ‹*RETURN o Binary-Stats-rounds*›
  :: ‹*binary-stats-assn*^{k} $\rightarrow_a$ *word64-assn*›
  unfolding *binary-stats-assn-def Binary-Stats-rounds-def*
  by *sepref*

sepref-def *Binary-stats-units-impl*
  is ‹*RETURN o Binary-Stats-units*›
  :: ‹*binary-stats-assn*^{k} $\rightarrow_a$ *word64-assn*›
  unfolding *binary-stats-assn-def Binary-Stats-units-def*
  by *sepref*

sepref-def *Binary-stats-removed-impl*
  is ‹*RETURN o Binary-Stats-removed*›
  :: ‹*binary-stats-assn*^{k} $\rightarrow_a$ *word64-assn*›
  unfolding *binary-stats-assn-def Binary-Stats-removed-def*
  by *sepref*

sepref-def *Pure-Lits-Stats-removed-impl*
  is ‹*RETURN o Pure-Lits-Stats-removed*›
  :: ‹*pure-lits-stats-assn*^{k} $\rightarrow_a$ *word64-assn*›
  unfolding *pure-lits-stats-assn-def Pure-Lits-Stats-removed-def*
  by *sepref*

sepref-def *Pure-Lits-Stats-removed-rounds-impl*
  is ‹*RETURN o Pure-Lits-Stats-rounds*›
  :: ‹*pure-lits-stats-assn*^{k} $\rightarrow_a$ *word64-assn*›
  unfolding *pure-lits-stats-assn-def Pure-Lits-Stats-rounds-def*
  by *sepref*

sepref-def *LSize-Stats-lbd-impl*
  is ‹*RETURN o LSize-Stats-lbd*›
  :: ‹*lbd-size-limit-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
  unfolding *LSize-Stats-lbd-def lbd-size-limit-assn-def*
  by *sepref*

sepref-def *LSize-Stats-size-impl*
  is ‹*RETURN o LSize-Stats-size*›
  :: ‹*lbd-size-limit-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*›
  unfolding *LSize-Stats-size-def lbd-size-limit-assn-def*
  by *sepref*

sepref-def *LSize-Stats-impl*
  is ‹*uncurry* (*RETURN oo LSize-Stats*)›
  :: ‹*uint32-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *lbd-size-limit-assn*›
  unfolding *LSize-Stats-def lbd-size-limit-assn-def*
  by *sepref*

sepref-register *Search-Stats-decisions Pure-Lits-Stats-rounds Pure-Lits-Stats-removed Binary-Stats-removed Binary-Stats-rounds Binary-Stats-units*
sepref-def *stats-decisions-impl*
  is ‹*RETURN o stats-decisions*› :: ‹*isasat-stats-assn*^{k} $\rightarrow_a$ *word64-assn*›
 unfolding *stats-decisions-def stats-code-unfold* **by** *sepref*

sepref-def *stats-irred-impl*
  is ‹*RETURN o stats-irred*› :: ‹*isasat-stats-assn*^{k} →$_a$ *word64-assn*›
 unfolding *stats-irred-def stats-code-unfold* **by** *sepref*

sepref-def *stats-binary-units-impl*
  is ‹*RETURN o stats-binary-units*› :: ‹*isasat-stats-assn*^{k} →$_a$ *word64-assn*›
 unfolding *stats-binary-units-def stats-code-unfold* **by** *sepref*

sepref-def *stats-binary-removed-impl*
  is ‹*RETURN o stats-binary-removed*› :: ‹*isasat-stats-assn*^{k} →$_a$ *word64-assn*›
 unfolding *stats-binary-removed-def stats-code-unfold* **by** *sepref*
sepref-def *stats-binary-rounds-impl*
  is ‹*RETURN o stats-binary-rounds*› :: ‹*isasat-stats-assn*^{k} →$_a$ *word64-assn*›
 unfolding *stats-binary-rounds-def stats-code-unfold* **by** *sepref*

sepref-def *stats-pure-lits-rounds-impl*
  is ‹*RETURN o stats-pure-lits-rounds*› :: ‹*isasat-stats-assn*^{k} →$_a$ *word64-assn*›
 unfolding *stats-pure-lits-rounds-def stats-code-unfold* **by** *sepref*

sepref-def *stats-pure-lits-removed-impl*
  is ‹*RETURN o stats-pure-lits-removed*› :: ‹*isasat-stats-assn*^{k} →$_a$ *word64-assn*›
 unfolding *stats-pure-lits-removed-def stats-code-unfold* **by** *sepref*

sepref-def *units-since-beginning-stats-impl*
  is ‹(*RETURN o units-since-beginning*)›
  :: ‹*isasat-stats-assn*^{k} →$_a$ *word-assn*›
  unfolding *units-since-beginning-def stats-code-unfold*
  by *sepref*

sepref-def *incr-irred-clss-stats-impl*
  is ‹*RETURN o incr-irred-clss*›
  :: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
  unfolding *incr-irred-clss-def stats-code-unfold*
  by *sepref*

sepref-def *decr-irred-clss-stats-impl*
  is ‹*RETURN o decr-irred-clss*›
  :: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
  unfolding *decr-irred-clss-def stats-code-unfold*
  by *sepref*

sepref-def *incr-propagation-stats-impl*
  is ‹*RETURN o incr-propagation*›
  :: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
  unfolding *incr-propagation-def stats-code-unfold*
  by *sepref*

sepref-def *incr-propagation-by-stats-impl*
  is ‹*uncurry* (*RETURN oo incr-propagation-by*)›
  :: ‹*word64-assn*^{k} *$_a$ *isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
  unfolding *incr-propagation-by-def stats-code-unfold*
  by *sepref*

sepref-def *set-not-conflict-until-stats-impl*
  is ‹*uncurry* (*RETURN oo set-no-conflict-until*)›
  :: ‹*word64-assn*^{k} *$_a$ *isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›

unfolding *set-no-conflict-until-def stats-code-unfold*
by *sepref*

sepref-def *no-conflict-until-stats-impl*
is ‹*RETURN o no-conflict-until*›
:: ‹*isasat-stats-assn*^{k} →$_a$ *word64-assn*›
unfolding *no-conflict-until-def stats-code-unfold*
by *sepref*

sepref-def *incr-conflict-stats-impl*
is ‹*RETURN o incr-conflict*›
:: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
unfolding *incr-conflict-def stats-code-unfold*
by *sepref*

sepref-def *incr-decision-stats-impl*
is ‹*RETURN o incr-decision*›
:: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
unfolding *incr-decision-def stats-code-unfold*
by *sepref*

sepref-def *incr-restart-stats-impl*
is ‹*RETURN o incr-restart*›
:: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
unfolding *incr-restart-def stats-code-unfold*
by *sepref*

sepref-def *incr-reduction-stats-impl*
is ‹*RETURN o incr-reduction*›
:: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
unfolding *incr-reduction-def stats-code-unfold*
by *sepref*

sepref-def *incr-uset-stats-impl*
is ‹*RETURN o incr-uset*›
:: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
unfolding *incr-uset-def stats-code-unfold*
by *sepref*

sepref-def *incr-uset-by-stats-impl*
is ‹*uncurry* (*RETURN oo incr-uset-by*)›
:: ‹*word64-assn*^{k} *$_a$ *isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
unfolding *incr-uset-by-def stats-code-unfold*
by *sepref*

sepref-def *incr-GC-stats-impl*
is ‹*RETURN o incr-GC*›
:: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
unfolding *incr-GC-def stats-code-unfold*
by *sepref*

sepref-def *stats-conflicts-impl*
is ‹*RETURN o stats-conflicts*›
:: ‹*isasat-stats-assn*^{k} →$_a$ *word-assn*›
unfolding *stats-conflicts-def stats-code-unfold*
by *sepref*

sepref-def *incr-units-since-last-GC-impl*
is ‹*RETURN o incr-units-since-last-GC*›
:: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
unfolding *incr-units-since-last-GC-def stats-code-unfold*
by *sepref*

sepref-def *incr-units-since-last-GC-by-impl*
is ‹*uncurry* (*RETURN oo incr-units-since-last-GC-by*)›
:: ‹*word64-assn*^{k} *$*_a$ *isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
unfolding *incr-units-since-last-GC-by-def stats-code-unfold*
by *sepref*

sepref-def *incr-purelit-rounds-impl*
is ‹*RETURN o incr-purelit-rounds*›
:: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
unfolding *incr-purelit-rounds-def stats-code-unfold*
by *sepref*

sepref-def *incr-purelit-elim-stats-impl*
is ‹*RETURN o incr-purelit-elim*›
:: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
unfolding *incr-purelit-elim-def stats-code-unfold*
by *sepref*

sepref-def *incr-binary-red-removed-impl*
is ‹*RETURN o incr-binary-red-removed*›
:: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
unfolding *incr-binary-red-removed-def stats-code-unfold*
by *sepref*

sepref-def *incr-binary-unit-derived-impl*
is ‹*RETURN o incr-binary-unit-derived*›
:: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
unfolding *incr-binary-unit-derived-def stats-code-unfold*
by *sepref*

sepref-def *get-reduction-count-impl*
is ‹*RETURN o get-reduction-count*›
:: ‹*isasat-stats-assn*^{k} →$_a$ *word-assn*›
unfolding *get-reduction-count-def stats-code-unfold*
by *sepref*

sepref-def *get-restart-count-impl*
is ‹*RETURN o get-restart-count*›
:: ‹*isasat-stats-assn*^{k} →$_a$ *word-assn*›
unfolding *get-restart-count-def stats-code-unfold*
by *sepref*

sepref-def *get-irredundant-count-impl*
is ‹*RETURN o irredundant-clss*›
:: ‹*isasat-stats-assn*^{k} →$_a$ *word-assn*›
unfolding *irredundant-clss-def stats-code-unfold*
by *sepref*

sepref-def *stats-propagations-impl*
  is ‹*RETURN o stats-propagations*›
  :: ‹*isasat-stats-assn*^{k} →$_a$ *word-assn*›
  unfolding *stats-propagations-def stats-code-unfold*
  by *sepref*

sepref-def *stats-restarts-impl*
  is ‹*RETURN o stats-restarts*›
  :: ‹*isasat-stats-assn*^{k} →$_a$ *word-assn*›
  unfolding *stats-restarts-def stats-code-unfold*
  by *sepref*

sepref-def *stats-reductions-impl*
  is ‹*RETURN o stats-reductions*›
  :: ‹*isasat-stats-assn*^{k} →$_a$ *word-assn*›
  unfolding *stats-reductions-def stats-code-unfold*
  by *sepref*

sepref-def *stats-fixed-impl*
  is ‹*RETURN o stats-fixed*›
  :: ‹*isasat-stats-assn*^{k} →$_a$ *word-assn*›
  unfolding *stats-fixed-def stats-code-unfold*
  by *sepref*

sepref-def *stats-gcs-impl*
  is ‹*RETURN o stats-gcs*›
  :: ‹*isasat-stats-assn*^{k} →$_a$ *word-assn*›
  unfolding *stats-gcs-def stats-code-unfold*
  by *sepref*

sepref-def *stats-avg-lbd-mpl*
  is ‹*RETURN o stats-avg-lbd*›
  :: ‹*isasat-stats-assn*^{k} →$_a$ *ema-assn*›
  unfolding *stats-avg-lbd-def stats-code-unfold*
  by *sepref*

sepref-register *LSize-Stats-lbd LSize-Stats-size LSize-Stats*
sepref-def *stats-lbd-limit-impl*
  is ‹*RETURN o stats-lbd-limit*›
  :: ‹*isasat-stats-assn*^{k} →$_a$ *uint32-nat-assn*›
  unfolding *stats-lbd-limit-def stats-code-unfold*
  by *sepref*

sepref-def *stats-size-limit-impl*
  is ‹*RETURN o stats-size-limit*›
  :: ‹*isasat-stats-assn*^{k} →$_a$ *sint64-nat-assn*›
  unfolding *stats-size-limit-def stats-code-unfold*
  by *sepref*

sepref-def *Subsumption-Stats-incr-strengthening-impl*
  is ‹*RETURN o Subsumption-Stats-incr-strengthening*›
  :: ‹*subsumption-stats-assn*^{d} →$_a$ *subsumption-stats-assn*›
  unfolding *Subsumption-Stats-incr-strengthening-def subsumption-stats-assn-def*
  by *sepref*

sepref-def *incr-forward-strengethening-impl*

is ‹*RETURN o incr-forward-strengethening*›
:: ‹*isasat-stats-assn*^{d} $\rightarrow_a$ *isasat-stats-assn*›
unfolding *incr-forward-strengethening-def stats-code-unfold*
by *sepref*

sepref-def *Subsumption-Stats-incr-subsumed-impl*
is ‹*RETURN o Subsumption-Stats-incr-subsumed*›
:: ‹*subsumption-stats-assn*^{d} $\rightarrow_a$ *subsumption-stats-assn*›
unfolding *Subsumption-Stats-incr-subsumed-def subsumption-stats-assn-def*
by *sepref*

sepref-def *Subsumption-Stats-incr-tried-impl*
is ‹*RETURN o Subsumption-Stats-incr-tried*›
:: ‹*subsumption-stats-assn*^{d} $\rightarrow_a$ *subsumption-stats-assn*›
unfolding *Subsumption-Stats-incr-tried-def subsumption-stats-assn-def*
by *sepref*

sepref-def *Subsumption-Stats-incr-rounds-impl*
is ‹*RETURN o Subsumption-Stats-incr-rounds*›
:: ‹*subsumption-stats-assn*^{d} $\rightarrow_a$ *subsumption-stats-assn*›
unfolding *Subsumption-Stats-incr-rounds-def subsumption-stats-assn-def*
by *sepref*

sepref-def *incr-forward-subsumed-impl*
is ‹*RETURN o incr-forward-subsumed*›
:: ‹*isasat-stats-assn*^{d} $\rightarrow_a$ *isasat-stats-assn*›
unfolding *incr-forward-subsumed-def stats-code-unfold*
by *sepref*

sepref-def *incr-forward-rounds-impl*
is ‹*RETURN o incr-forward-rounds*›
:: ‹*isasat-stats-assn*^{d} $\rightarrow_a$ *isasat-stats-assn*›
unfolding *incr-forward-rounds-def stats-code-unfold*
by *sepref*

sepref-def *incr-forward-tried-impl*
is ‹*RETURN o incr-forward-tried*›
:: ‹*isasat-stats-assn*^{d} $\rightarrow_a$ *isasat-stats-assn*›
unfolding *incr-forward-tried-def stats-code-unfold*
by *sepref*

sepref-def *Subsumption-Stats-rounds-impl*
is ‹*RETURN o Subsumption-Stats-rounds*›
:: ‹*subsumption-stats-assn*^{k} $\rightarrow_a$ *word64-assn*›
unfolding *subsumption-stats-assn-def Subsumption-Stats-rounds-def*
by *sepref*
sepref-def *Subsumption-Stats-strengthened-impl*
is ‹*RETURN o Subsumption-Stats-strengthened*›
:: ‹*subsumption-stats-assn*^{k} $\rightarrow_a$ *word64-assn*›
unfolding *subsumption-stats-assn-def Subsumption-Stats-strengthened-def*
by *sepref*
sepref-def *Subsumption-Stats-subsumed-impl*
is ‹*RETURN o Subsumption-Stats-subsumed*›
:: ‹*subsumption-stats-assn*^{k} $\rightarrow_a$ *word64-assn*›
unfolding *subsumption-stats-assn-def Subsumption-Stats-subsumed-def*
by *sepref*

sepref-def *Subsumption-Stats-tried-impl*
  is ‹*RETURN o Subsumption-Stats-tried*›
  :: ‹*subsumption-stats-assn*^{k} →$_a$ *word64-assn*›
  unfolding *subsumption-stats-assn-def Subsumption-Stats-tried-def*
  by *sepref*

sepref-def *stats-forward-rounds-impl*
  is ‹*RETURN o stats-forward-rounds*›
  :: ‹*isasat-stats-assn*^{k} →$_a$ *word64-assn*›
  unfolding *stats-forward-rounds-def stats-code-unfold*
  by *sepref*

sepref-def *Rephase-Stats-incr-total-impl*
  is ‹*RETURN o Rephase-Stats-incr-total*›
  :: ‹*rephase-stats-assn*^{d} →$_a$ *rephase-stats-assn*›
  unfolding *rephase-stats-assn-def Rephase-Stats-incr-total-def*
  by *sepref*

sepref-def *Rephase-Stats-total-impl*
  is ‹*RETURN o Rephase-Stats-total*›
  :: ‹*rephase-stats-assn*^{d} →$_a$ *word64-assn*›
  unfolding *rephase-stats-assn-def Rephase-Stats-total-def*
  by *sepref*

sepref-register *stats-forward-tried stats-forward-subsumed stats-forward-strengthened*

sepref-def *stats-forward-subsumed-impl*
  is ‹*RETURN o stats-forward-subsumed*›
  :: ‹*isasat-stats-assn*^{d} →$_a$ *word64-assn*›
  unfolding *stats-forward-subsumed-def stats-code-unfold*
  by *sepref*

sepref-def *stats-forward-tried-impl*
  is ‹*RETURN o stats-forward-tried*›
  :: ‹*isasat-stats-assn*^{d} →$_a$ *word64-assn*›
  unfolding *stats-forward-tried-def stats-code-unfold*
  by *sepref*

sepref-def *stats-forward-strengthened-impl*
  is ‹*RETURN o stats-forward-strengthened*›
  :: ‹*isasat-stats-assn*^{d} →$_a$ *word64-assn*›
  unfolding *stats-forward-strengthened-def stats-code-unfold*
  by *sepref*

lemmas [*llvm-inline*] = *Mreturn-comp-Tuple16*

sepref-register *empty-stats*
sepref-def *empty-stats-impl*
  is ‹*uncurry0* (*RETURN empty-stats*)›
  :: ‹*unit-assn*^{k} →$_a$ *isasat-stats-assn*›
  unfolding *empty-stats-def empty-search-stats-def*[*symmetric*]
  apply (*subst empty-rephase-stats-def*[*symmetric*])
  unfolding *empty-subsumption-stats-def*[*symmetric*]
  unfolding *empty-binary-stats-def*[*symmetric*]
  apply (*subst empty-pure-lits-stats-def*[*symmetric*])

apply (*subst empty-lsize-limit-stats-def*[*symmetric*])
by *sepref*

definition *empty-search-stats-clss* **where**
‹*empty-search-stats-clss n* = (*0*,*0*,*0*,*0*,*0*,*0*,*0*,*n*,*0*)›

sepref-def *empty-search-stats-clss-impl*
is ‹(*RETURN o empty-search-stats-clss*)›
:: ‹*word64-assn*^{k} →$_a$ *search-stats-assn*›
unfolding *search-stats-assn-def empty-search-stats-clss-def*
by *sepref*

sepref-def *empty-stats-clss-impl*
is ‹(*RETURN o empty-stats-clss*)›
:: ‹*word64-assn*^{k} →$_a$ *isasat-stats-assn*›
unfolding *empty-stats-clss-def empty-search-stats-clss-def*[*symmetric*]
apply (*subst empty-rephase-stats-def*[*symmetric*])
unfolding *empty-subsumption-stats-def*[*symmetric*]
unfolding *empty-binary-stats-def*[*symmetric*]
apply (*subst empty-pure-lits-stats-def*[*symmetric*])
apply (*subst empty-lsize-limit-stats-def*[*symmetric*])
by *sepref*

sepref-register *Rephase-Stats-incr-total Rephase-Stats-total stats-rephase incr-rephase-total*
sepref-def *stats-rephase-impl*
is ‹*RETURN o stats-rephase*›
:: ‹*isasat-stats-assn*^{k} →$_a$ *word64-assn*›
unfolding *stats-rephase-def stats-code-unfold*
by *sepref*

sepref-def *incr-rephase-total-impl*
is ‹*RETURN o incr-rephase-total*›
:: ‹*isasat-stats-assn*^{k} →$_a$ *isasat-stats-assn*›
unfolding *incr-rephase-total-def stats-code-unfold*
by *sepref*

export-llvm *empty-stats-impl*

sepref-register *unset-fully-propagated-heur is-fully-propagated-heur set-fully-propagated-heur*

abbreviation (*input*) ‹*restart-info-rel* ≡ *word64-rel* ×$_r$ *word64-rel* ×$_r$ *word64-rel* ×$_r$ *word64-rel* ×$_r$ *word64-rel*›
abbreviation (*input*) *restart-info-assn* **where**
‹*restart-info-assn* ≡ *word64-assn* ×$_a$ *word64-assn* ×$_a$ *word64-assn* ×$_a$ *word64-assn* ×$_a$ *word64-assn*›

lemma *restart-info-params*[*sepref-import-param*]:
(*incr-conflict-count-since-last-restart*,*incr-conflict-count-since-last-restart*) ∈
restart-info-rel → *restart-info-rel*
(*restart-info-update-lvl-avg*,*restart-info-update-lvl-avg*) ∈
word32-rel → *restart-info-rel* → *restart-info-rel*
‹(*restart-info-init*,*restart-info-init*) ∈ *restart-info-rel*›
‹(*restart-info-restart-done*,*restart-info-restart-done*) ∈ *restart-info-rel* → *restart-info-rel*›
by *auto*

lemmas [*llvm-inline*] =

incr-conflict-count-since-last-restart-def
restart-info-update-lvl-avg-def
restart-info-init-def
restart-info-restart-done-def

abbreviation (*input*) ‹*schedule-info-rel* ≡ *word64-rel* ×$_r$ *word64-rel* ×$_r$ *word64-rel*›
abbreviation (*input*) *schedule-info-assn* **where**
‹*schedule-info-assn* ≡ *word64-assn* ×$_a$ *word64-assn* ×$_a$ *word64-assn*›

lemma *schedule-info-params*[*sepref-import-param*]:
(*next-pure-lits-schedule-info*, *next-pure-lits-schedule-info*) ∈
schedule-info-rel → *word64-rel*
(*schedule-next-pure-lits-info*, *schedule-next-pure-lits-info*) ∈
schedule-info-rel → *schedule-info-rel*
(*next-reduce-schedule-info*, *next-reduce-schedule-info*) ∈ *schedule-info-rel* → *word64-rel*
(*schedule-next-reduce-info*, *schedule-next-reduce-info*) ∈
word64-rel → *schedule-info-rel* → *schedule-info-rel*
by (*auto*)

sepref-register *FOCUSED-MODE STABLE-MODE DEFAULT-INIT-PHASE*

sepref-def *FOCUSED-MODE-impl*
is ‹*uncurry0* (*RETURN FOCUSED-MODE*)›
:: ‹*unit-assn*^{k} →$_a$ *word-assn*›
unfolding *FOCUSED-MODE-def*
by *sepref*

sepref-def *STABLE-MODE-impl*
is ‹*uncurry0* (*RETURN STABLE-MODE*)›
:: ‹*unit-assn*^{k} →$_a$ *word-assn*›
unfolding *STABLE-MODE-def*
by *sepref*

definition *lcount-assn* :: ‹*clss-size* ⇒ - ⇒ *assn*› **where**
‹*lcount-assn* ≡ *uint64-nat-assn* ×$_a$ *uint64-nat-assn* ×$_a$ *uint64-nat-assn* ×$_a$ *uint64-nat-assn* ×$_a$ *uint64-nat-assn*›

lemma [*safe-constraint-rules*]:
‹*CONSTRAINT Sepref-Basic.is-pure lcount-assn*›
unfolding *lcount-assn-def*
by *auto*

sepref-def *clss-size-lcount-fast-code*
is ‹*RETURN o clss-size-lcount*›
:: ‹*lcount-assn*^{k} →$_a$ *uint64-nat-assn*›
unfolding *clss-size-lcount-def lcount-assn-def*
by *sepref*

sepref-register *clss-size-resetUS*

lemma *clss-size-resetUS-alt-def*:
‹*RETURN o clss-size-resetUS* =
(λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *lcountU0*). *RETURN* (*lcount*, *lcountUE*, *lcountUEk*, *0*, *lcountU0*))›
by (*auto simp*: *clss-size-resetUS-def*)

sepref-def *clss-size-resetUS-fast-code*

is ‹*RETURN o clss-size-resetUS*›
:: ‹*lcount-assn*^{d} →$_a$ *lcount-assn*›
unfolding *clss-size-resetUS-alt-def lcount-assn-def*
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *clss-size-incr-lcountUS-alt-def*:
‹*RETURN o clss-size-incr-lcountUS* =
(λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *lcountU0*). *RETURN* (*lcount*, *lcountUE*, *lcountUEk*, *lcountUS* + *1*, *lcountU0*))›
by (*auto simp*: *clss-size-incr-lcountUS-def*)

sepref-def *clss-size-incr-lcountUS-fast-code*
is ‹*RETURN o clss-size-incr-lcountUS*›
:: ‹[λ*S*. *clss-size-lcountUS S* < *unat64-max*]$_a$ *lcount-assn*^{d} → *lcount-assn*›
unfolding *clss-size-incr-lcountUS-alt-def lcount-assn-def clss-size-lcountUS-def*
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *clss-size-resetU0-alt-def*:
‹*RETURN o clss-size-resetU0* =
(λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *lcountU0*). *RETURN* (*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *0*))›
by (*auto simp*: *clss-size-resetU0-def*)

sepref-def *clss-size-resetU0-fast-code*
is ‹*RETURN o clss-size-resetU0*›
:: ‹*lcount-assn*^{d} →$_a$ *lcount-assn*›
unfolding *clss-size-resetU0-alt-def lcount-assn-def*
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *clss-size-incr-lcountU0-alt-def*:
‹*RETURN o clss-size-incr-lcountU0* =
(λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *lcountU0*). *RETURN* (*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *lcountU0+1*))›
by (*auto simp*: *clss-size-incr-lcountU0-def*)

sepref-def *clss-size-incr-lcountU0-fast-code*
is ‹*RETURN o clss-size-incr-lcountU0*›
:: ‹[λ*S*. *clss-size-lcountU0 S* < *unat64-max*]$_a$ *lcount-assn*^{d} → *lcount-assn*›
unfolding *clss-size-incr-lcountU0-alt-def lcount-assn-def clss-size-lcountU0-def*
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *clss-size-resetUE-alt-def*:
‹*RETURN o clss-size-resetUE* =
(λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *lcountU0*). *RETURN* (*lcount*, *0*, *lcountUEk*, *lcountUS*, *lcountU0*))›
by (*auto simp*: *clss-size-resetUE-def*)

sepref-def *clss-size-resetUE-fast-code*
is ‹*RETURN o clss-size-resetUE*›
:: ‹*lcount-assn*^{d} →$_a$ *lcount-assn*›
unfolding *clss-size-resetUE-alt-def lcount-assn-def*
apply (*annot-unat-const* ‹*TYPE*(*64*)›)

by *sepref*

lemma *clss-size-incr-lcountUE-alt-def*:
‹*RETURN o clss-size-incr-lcountUE* =
(λ(*lcount*, *lcountUE*, *lcountUS*). *RETURN* (*lcount*, *lcountUE* + *1*, *lcountUS*))›
by (*auto simp*: *clss-size-incr-lcountUE-def*)

sepref-def *clss-size-incr-lcountUE-fast-code*
is ‹*RETURN o clss-size-incr-lcountUE*›
:: ‹[λ*S*. *clss-size-lcountUE S* < *unat64-max*]$_a$ *lcount-assn*^{d} → *lcount-assn*›
unfolding *clss-size-incr-lcountUE-alt-def lcount-assn-def clss-size-lcountUE-def*
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *clss-size-incr-lcountUEk-alt-def*:
‹*RETURN o clss-size-incr-lcountUEk* =
(λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*). *RETURN* (*lcount*, *lcountUE*, *lcountUEk* + *1*, *lcountUS*))›
by (*auto simp*: *clss-size-incr-lcountUEk-def*)

sepref-def *clss-size-incr-lcountUEk-fast-code*
is ‹*RETURN o clss-size-incr-lcountUEk*›
:: ‹[λ*S*. *clss-size-lcountUEk S* < *unat64-max*]$_a$ *lcount-assn*^{d} → *lcount-assn*›
unfolding *clss-size-incr-lcountUEk-alt-def lcount-assn-def clss-size-lcountUEk-def*
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
by *sepref*

schematic-goal *mk-free-lookup-clause-rel-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE lcount-assn ?fr*›
unfolding *lcount-assn-def*
by (*rule free-thms sepref-frame-free-rules*)+

lemma *clss-size-lcountUE-alt-def*:
‹*RETURN o clss-size-lcountUE* = (λ(*lcount*, *lcountUE*, *lcountUS*). *RETURN lcountUE*)›
by (*auto simp*: *clss-size-lcountUE-def*)

sepref-def *clss-size-lcountUE-fast-code*
is ‹*RETURN o clss-size-lcountUE*›
:: ‹*lcount-assn*^{k} →$_a$ *uint64-nat-assn*›
unfolding *lcount-assn-def clss-size-lcountUE-alt-def clss-size-lcount-def*
by *sepref*

lemma *clss-size-lcountUS-alt-def*:
‹*RETURN o clss-size-lcountUS* = (λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *lcountU0*). *RETURN lcountUS*)›
by (*auto simp*: *clss-size-lcountUS-def*)

sepref-def *clss-size-lcountUSt-fast-code*
is ‹*RETURN o clss-size-lcountUS*›
:: ‹*lcount-assn*^{k} →$_a$ *uint64-nat-assn*›
unfolding *lcount-assn-def clss-size-lcountUS-alt-def clss-size-lcount-def*
by *sepref*

lemma *clss-size-lcountU0-alt-def*:
‹*RETURN o clss-size-lcountU0* = (λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *lcountU0*). *RETURN lcountU0*)›
by (*auto simp*: *clss-size-lcountU0-def*)

sepref-def *clss-size-lcountU0-fast-code*
is ‹*RETURN o clss-size-lcountU0*›
:: ‹*lcount-assn*^{k} →$_a$ *uint64-nat-assn*›
unfolding *lcount-assn-def clss-size-lcountU0-alt-def clss-size-lcount-def*
by *sepref*

lemma *clss-size-incr-allcount-alt-def*:
‹*RETURN o clss-size-allcount* =
(λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *lcountU0*). *RETURN* (*lcount* + *lcountUE* + *lcountUEk* + *lcountUS* + *lcountU0*))›
by (*auto simp*: *clss-size-allcount-def*)

sepref-def *clss-size-allcount-fast-code*
is ‹*RETURN o clss-size-allcount*›
:: ‹[λ*S*. *clss-size-allcount S* < *max-snat 64*]$_a$ *lcount-assn*^{d} → *uint64-nat-assn*›
unfolding *clss-size-incr-allcount-alt-def lcount-assn-def clss-size-allcount-def*
by *sepref*

lemma *clss-size-decr-lcount-alt-def*:
‹*RETURN o clss-size-decr-lcount* =
(λ(*lcount*, *lcountUE*, *lcountUS*). *RETURN* (*lcount* − *1*, *lcountUE*, *lcountUS*))›
by (*auto simp*: *clss-size-decr-lcount-def*)

sepref-def *clss-size-decr-lcount-fast-code*
is ‹*RETURN o clss-size-decr-lcount*›
:: ‹[λ*S*. *clss-size-lcount S* ≥ *1*]$_a$ *lcount-assn*^{d} → *lcount-assn*›
unfolding *lcount-assn-def clss-size-decr-lcount-alt-def clss-size-lcount-def*
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *emag-get-value-alt-def*:
‹*ema-get-value* = (λ(*a*, *b*, *c*, *d*). *a*)›
by *auto*

sepref-def *ema-get-value-impl*
is ‹*RETURN o ema-get-value*›
:: ‹*ema-assn*^{k} →$_a$ *word-assn*›
unfolding *emag-get-value-alt-def*
by *sepref*

lemma *emag-extract-value-alt-def*:
‹*ema-extract-value* = (λ(*a*, *b*, *c*, *d*). *a* >> *EMA-FIXPOINT-SIZE*)›
by *auto*

sepref-def *ema-extract-value-impl*
is ‹*RETURN o ema-extract-value*›
:: ‹*ema-assn*^{k} →$_a$ *word-assn*›
unfolding *emag-extract-value-alt-def EMA-FIXPOINT-SIZE-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *schedule-next-pure-lits-info-impl*
is ‹*RETURN o schedule-next-pure-lits-info*›
:: ‹*schedule-info-assn*^{k} →$_a$ *schedule-info-assn*›

unfolding *schedule-next-pure-lits-info-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *next-pure-lits-schedule-info-impl*
is ‹*RETURN o next-pure-lits-schedule-info*›
:: ‹*schedule-info-assn*^{k} →$_a$ *word64-assn*›
unfolding *next-pure-lits-schedule-info-def*
by *sepref*

sepref-def *schedule-next-reduce-info-impl*
is ‹*uncurry* (*RETURN oo schedule-next-reduce-info*)›
:: ‹*word64-assn*^{k} *$_a$ *schedule-info-assn*^{k} →$_a$ *schedule-info-assn*›
unfolding *schedule-next-reduce-info-def*
by *sepref*

sepref-def *next-reduce-schedule-info-impl*
is ‹*RETURN o next-reduce-schedule-info*›
:: ‹*schedule-info-assn*^{k} →$_a$ *word64-assn*›
unfolding *next-reduce-schedule-info-def*
by *sepref*

sepref-def *schedule-next-subsume-info-impl*
is ‹*uncurry* (*RETURN oo schedule-next-subsume-info*)›
:: ‹*word64-assn*^{k} *$_a$ *schedule-info-assn*^{k} →$_a$ *schedule-info-assn*›
unfolding *schedule-next-subsume-info-def*
by *sepref*

sepref-def *next-subsume-schedule-info-impl*
is ‹*RETURN o next-subsume-schedule-info*›
:: ‹*schedule-info-assn*^{k} →$_a$ *word64-assn*›
unfolding *next-subsume-schedule-info-def*
by *sepref*

type-synonym *heur-assn* = ‹(*ema* × *ema* × *restart-info* × *64 word* ×
(*phase-saver-assn* × *64 word* × *phase-saver′-assn* × *64 word* × *phase-saver′-assn* × *64 word* × *64 word* × *64 word*) ×
reluctant-rel-assn × *1 word* × *phase-saver-assn* × (*64 word* × *64 word* × *64 word*) × *ema* × *ema*)›

definition *heuristic-int-assn* :: ‹*restart-heuristics* ⇒ *heur-assn* ⇒ *assn*› **where**
‹*heuristic-int-assn* = *ema-assn* ×$_a$
ema-assn ×$_a$
restart-info-assn ×$_a$
word64-assn ×$_a$ *phase-heur-assn* ×$_a$ *reluctant-assn* ×$_a$ *bool1-assn* ×$_a$ *phase-saver-assn* ×$_a$
schedule-info-assn ×$_a$ *ema-assn* ×$_a$ *ema-assn*›

abbreviation *heur-int-rel* :: ‹(*restart-heuristics* × *restart-heuristics*) *set*› **where**
‹*heur-int-rel* ≡ *Id*›

abbreviation *heur-rel* :: ‹(*restart-heuristics* × *isasat-restart-heuristics*) *set*› **where**
‹*heur-rel* ≡ ⟨*heur-int-rel*⟩*code-hider-rel*›

definition *heuristic-assn* :: ‹*isasat-restart-heuristics* ⇒ - ⇒ *assn*› **where**
‹*heuristic-assn* = *code-hider-assn heuristic-int-assn heur-int-rel*›

lemma *heuristic-assn-alt-def*:
‹*heuristic-assn* = *hr-comp heuristic-int-assn heur-rel*›
unfolding *heuristic-assn-def code-hider-assn-def* **by** *auto*

context
notes [*fcomp-norm-unfold*] = *heuristic-assn-def*[*symmetric*] *heuristic-assn-alt-def*[*symmetric*]
begin

lemma *set-zero-wasted-stats-set-zero-wasted-stats*:
‹(*set-zero-wasted-stats*, *set-zero-wasted*) ∈ *heur-rel* → *heur-rel*› **and**
heuristic-reluctant-tick-stats-heuristic-reluctant-tick:
‹(*heuristic-reluctant-tick-stats*, *heuristic-reluctant-tick*) ∈ *heur-rel* → *heur-rel*› **and**
heuristic-reluctant-enable-stats-heuristic-reluctant-enable:
‹(*heuristic-reluctant-enable-stats*,*heuristic-reluctant-enable*) ∈ *heur-rel* → *heur-rel*› **and**
heuristic-reluctant-disable-stats-heuristic-reluctant-disable:
‹(*heuristic-reluctant-disable-stats*,*heuristic-reluctant-disable*) ∈ *heur-rel* → *heur-rel*› **and**
heuristic-reluctant-triggered-stats-heuristic-reluctant-triggered:
‹(*heuristic-reluctant-triggered-stats*,*heuristic-reluctant-triggered*) ∈ *heur-rel* → *heur-rel* ×$_f$ *bool-rel*›
and
heuristic-reluctant-triggered2-stats-heuristic-reluctant-triggered2:
‹(*heuristic-reluctant-triggered2-stats*,*heuristic-reluctant-triggered2*) ∈ *heur-rel* → *bool-rel*› **and**
heuristic-reluctant-untrigger-stats-heuristic-reluctant-untrigger:
‹(*heuristic-reluctant-untrigger-stats*, *heuristic-reluctant-untrigger*) ∈ *heur-rel* → *heur-rel*› **and**
end-of-rephasing-phase-heur-stats-end-of-rephasing-phase-heur:
‹(*end-of-rephasing-phase-heur-stats*, *end-of-rephasing-phase-heur*) ∈ *heur-rel* → *word64-rel*› **and**
is-fully-propagated-heur-stats-is-fully-propagated-heur:
‹(*is-fully-propagated-heur-stats*, *is-fully-propagated-heur*) ∈ *heur-rel* → *bool-rel*› **and**
set-fully-propagated-heur-stats-set-fully-propagated-heur:
‹(*set-fully-propagated-heur-stats*, *set-fully-propagated-heur*) ∈ *heur-rel* → *heur-rel*›**and**
unset-fully-propagated-heur-stats-unset-fully-propagated-heur:
‹(*unset-fully-propagated-heur-stats*, *unset-fully-propagated-heur*) ∈ *heur-rel* → *heur-rel*› **and**
restart-info-restart-done-heur-stats-restart-info-restart-done-heur:
‹(*restart-info-restart-done-heur-stats*, *restart-info-restart-done-heur*) ∈ *heur-rel* → *heur-rel*› **and**
set-zero-wasted-stats-set-zero-wasted:
‹(*set-zero-wasted-stats*, *set-zero-wasted*) ∈ *heur-rel* → *heur-rel*› **and**
wasted-of-stats-wasted-of:
‹(*wasted-of-stats*, *wasted-of*) ∈ *heur-rel* → *word64-rel*› **and**
slow-ema-of-stats-slow-ema-of:
‹(*slow-ema-of-stats*, *slow-ema-of*) ∈ *heur-rel* → *ema-rel*› **and**
fast-ema-of-stats-fast-ema-of:
‹(*fast-ema-of-stats*, *fast-ema-of*) ∈ *heur-rel* → *ema-rel*› **and**
current-restart-phase-stats-current-restart-phase:
‹(*current-restart-phase-stats*, *current-restart-phase*) ∈ *heur-rel* → *word-rel*› **and**
incr-wasted-stats-incr-wasted:
‹(*incr-wasted-stats*, *incr-wasted*) ∈ *word-rel* → *heur-rel* → *heur-rel*› **and**
current-rephasing-phase-stats-current-rephasing-phase:
‹(*current-rephasing-phase-stats*, *current-rephasing-phase*) ∈ *heur-rel* → *word-rel*› **and**
get-next-phase-heur-stats-get-next-phase-heur:
‹(*uncurry2* (*get-next-phase-heur-stats*), *uncurry2* (*get-next-phase-heur*))
∈ *Id* ×$_f$ *Id* ×$_f$ *heur-rel* →$_f$ ⟨*bool-rel*⟩*nres-rel*› **and**
get-conflict-count-since-last-restart-stats-get-conflict-count-since-last-restart-stats:
‹(*get-conflict-count-since-last-restart-stats*, *get-conflict-count-since-last-restart*)
∈ *heur-rel* → *word-rel*› **and**
schedule-next-pure-lits-stats-schedule-next-pure-lits:
‹(*schedule-next-pure-lits-stats*, *schedule-next-pure-lits*) ∈ *heur-rel* → *heur-rel*› **and**
next-pure-lits-schedule-next-pure-lits-schedule-stats:

‹(*next-pure-lits-schedule-info-stats*, *next-pure-lits-schedule*) ∈ *heur-rel* → *word64-rel*› **and**
schedule-next-reduce-stats-schedule-next-reduce:
‹(*schedule-next-reduce-stats*, *schedule-next-reduce*) ∈ *word64-rel* → *heur-rel* → *heur-rel*› **and**
next-reduce-schedule-next-reduce-schedule-stats:
‹(*next-reduce-schedule-info-stats*, *next-reduce-schedule*) ∈ *heur-rel* → *word64-rel*› **and**
schedule-next-subsume-stats-schedule-next-subsume:
‹(*schedule-next-subsume-stats*, *schedule-next-subsume*) ∈ *word64-rel* → *heur-rel* → *heur-rel*› **and**
next-subsume-schedule-next-subsume-schedule-stats:
‹(*next-subsume-schedule-info-stats*, *next-subsume-schedule*) ∈ *heur-rel* → *word64-rel*› **and**
swap-emas-stats-swap-emas:
‹(*swap-emas-stats*, *swap-emas*) ∈ *heur-rel* → *heur-rel*›
by (*auto simp*: *set-zero-wasted-def code-hider-rel-def heuristic-reluctant-tick-def*
heuristic-reluctant-enable-def heuristic-reluctant-triggered-def apfst-def map-prod-def
heuristic-reluctant-disable-def heuristic-reluctant-triggered2-def is-fully-propagated-heur-def
end-of-rephasing-phase-heur-def unset-fully-propagated-heur-def restart-info-restart-done-heur-def
heuristic-reluctant-untrigger-def set-fully-propagated-heur-def wasted-of-def get-next-phase-heur-def
slow-ema-of-def fast-ema-of-def current-restart-phase-def incr-wasted-def current-rephasing-phase-def
get-conflict-count-since-last-restart-def next-pure-lits-schedule-def
schedule-next-pure-lits-def schedule-next-pure-lits-stats-def next-reduce-schedule-def
schedule-next-reduce-def schedule-next-reduce-stats-def next-subsume-schedule-def
schedule-next-subsume-def schedule-next-subsume-stats-def swap-emas-def
intro!: *frefI nres-relI*
split: *prod.splits*)

lemma *set-zero-wasted-stats-alt-def*:
‹*set-zero-wasted-stats*= (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ).
(*fast-ema*, *slow-ema*, *res-info*, *0*, φ))›
by *auto*

sepref-def *set-zero-wasted-stats-impl*
is ‹*RETURN o set-zero-wasted-stats*›
:: ‹*heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
unfolding *heuristic-int-assn-def set-zero-wasted-stats-alt-def*
by *sepref*

sepref-def *heuristic-reluctant-tick-stats-impl*
is ‹*RETURN o heuristic-reluctant-tick-stats*›
:: ‹*heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
unfolding *heuristic-int-assn-def heuristic-reluctant-tick-stats-def*
by *sepref*

sepref-def *heuristic-reluctant-enable-stats-impl*
is ‹*RETURN o heuristic-reluctant-enable-stats*›
:: ‹*heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
unfolding *heuristic-int-assn-def heuristic-reluctant-enable-stats-def*
by *sepref*

sepref-def *heuristic-reluctant-disable-stats-impl*
is ‹*RETURN o heuristic-reluctant-disable-stats*›
:: ‹*heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
unfolding *heuristic-int-assn-def heuristic-reluctant-disable-stats-def*
by *sepref*

sepref-def *heuristic-reluctant-triggered-stats-impl*
is ‹*RETURN o heuristic-reluctant-triggered-stats*›

:: ‹*heuristic-int-assn*^{d} $\rightarrow_a$ *heuristic-int-assn* $\times_a$ *bool1-assn*›
unfolding *heuristic-reluctant-triggered-stats-def heuristic-int-assn-def*
by *sepref*

sepref-def *heuristic-reluctant-triggered2-stats-impl*
is ‹*RETURN o heuristic-reluctant-triggered2-stats*›
:: ‹*heuristic-int-assn*^{k} $\rightarrow_a$ *bool1-assn*›
unfolding *heuristic-reluctant-triggered2-stats-def heuristic-int-assn-def*
by *sepref*

sepref-def *heuristic-reluctant-untrigger-stats-impl*
is ‹*RETURN o heuristic-reluctant-untrigger-stats*›
:: ‹*heuristic-int-assn*^{d} $\rightarrow_a$ *heuristic-int-assn*›
unfolding *heuristic-int-assn-def heuristic-reluctant-untrigger-stats-def*
by *sepref*

sepref-def *end-of-rephasing-phase-impl* [*llvm-inline*]
is ‹*RETURN o end-of-rephasing-phase*›
:: ‹*phase-heur-assn*^{k} $\rightarrow_a$ *word64-assn*›
unfolding *end-of-rephasing-phase-def phase-heur-assn-def*
by *sepref*

sepref-def *end-of-rephasing-phase-heur-stats-impl*
is ‹*RETURN o end-of-rephasing-phase-heur-stats*›
:: ‹*heuristic-int-assn*^{k} $\rightarrow_a$ *word64-assn*›
unfolding *heuristic-int-assn-def end-of-rephasing-phase-heur-stats-def*
by *sepref*

sepref-def *is-fully-propagated-heur-stats-impl*
is ‹*RETURN o is-fully-propagated-heur-stats*›
:: ‹*heuristic-int-assn*^{k} $\rightarrow_a$ *bool1-assn*›
unfolding *heuristic-int-assn-def is-fully-propagated-heur-stats-def*
by *sepref*

sepref-def *set-fully-propagated-heur-stats-impl*
is ‹*RETURN o set-fully-propagated-heur-stats*›
:: ‹*heuristic-int-assn*^{d} $\rightarrow_a$ *heuristic-int-assn*›
unfolding *heuristic-int-assn-def set-fully-propagated-heur-stats-def*
by *sepref*

sepref-def *unset-fully-propagated-heur-stats-impl*
is ‹*RETURN o unset-fully-propagated-heur-stats*›
:: ‹*heuristic-int-assn*^{d} $\rightarrow_a$ *heuristic-int-assn*›
unfolding *heuristic-int-assn-def unset-fully-propagated-heur-stats-def*
by *sepref*

sepref-def *restart-info-restart-done-heur-stats-impl*
is ‹*RETURN o restart-info-restart-done-heur-stats*›
:: ‹*heuristic-int-assn*^{d} $\rightarrow_a$ *heuristic-int-assn*›
unfolding *heuristic-int-assn-def restart-info-restart-done-heur-stats-def*
by *sepref*

sepref-def *set-zero-wasted-impl*
is ‹*RETURN o set-zero-wasted-stats*›

:: ‹*heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
unfolding *heuristic-int-assn-def set-zero-wasted-stats-alt-def*
by *sepref*

lemma *wasted-of-stats-alt-def*:
‹*RETURN o wasted-of-stats* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ). *RETURN wasted*)›
by *auto*

sepref-def *wasted-of-stats-impl*
is ‹*RETURN o wasted-of-stats*›
:: ‹*heuristic-int-assn*^{k} →$_a$ *word64-assn*›
unfolding *heuristic-int-assn-def wasted-of-stats-alt-def*
by *sepref*

lemma *slow-ema-of-stats-alt-def*:
‹*RETURN o slow-ema-of-stats* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ). *RETURN slow-ema*)›
and
fast-ema-of-stats-alt-def:
‹*RETURN o fast-ema-of-stats* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ). *RETURN fast-ema*)›
by *auto*

sepref-def *slow-ema-of-stats-impl*
is ‹*RETURN o slow-ema-of-stats*›
:: ‹*heuristic-int-assn*^{k} →$_a$ *ema-assn*›
unfolding *heuristic-int-assn-def slow-ema-of-stats-alt-def*
by *sepref*

sepref-def *fast-ema-of-stats-impl*
is ‹*RETURN o fast-ema-of-stats*›
:: ‹*heuristic-int-assn*^{k} →$_a$ *ema-assn*›
unfolding *heuristic-int-assn-def fast-ema-of-stats-alt-def*
by *sepref*

lemma *current-restart-phase-stats-alt-def*:
‹*RETURN o current-restart-phase-stats* =
(λ(*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase*), *wasted*, φ). *RETURN restart-phase*)›
by *auto*

sepref-def *current-restart-phase-impl*
is ‹*RETURN o current-restart-phase-stats*›
:: ‹*heuristic-int-assn*^{k} →$_a$ *word-assn*›
unfolding *heuristic-int-assn-def current-restart-phase-stats-alt-def*
by *sepref*

lemma *incr-wasted-stats-stats-alt-def*:
‹*RETURN oo incr-wasted-stats* =
(λ*waste* (*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ). *RETURN* (*fast-ema*, *slow-ema*, *res-info*, *wasted* + *waste*, φ))›
by (*auto intro*!: *ext*)

sepref-def *incr-wasted-stats-impl*
is ‹*uncurry* (*RETURN oo incr-wasted-stats*)›
:: ‹*word64-assn*^{k} *$_a$ *heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
unfolding *heuristic-int-assn-def incr-wasted-stats-stats-alt-def*
by *sepref*

sepref-def *swap-emas-stats-impl*
is ‹*RETURN o swap-emas-stats*›
:: ‹*heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
unfolding *heuristic-int-assn-def swap-emas-stats-def*
by *sepref*

sepref-def *current-rephasing-phase-stats-impl*
is ‹*RETURN o current-rephasing-phase-stats*›
:: ‹*heuristic-int-assn*^{k} →$_a$ *word-assn*›
unfolding *heuristic-int-assn-def current-rephasing-phase-stats-def*
phase-current-rephasing-phase-def phase-heur-assn-def
by *sepref*

sepref-def *get-next-phase-heur-stats-impl*
is ‹*uncurry2 get-next-phase-heur-stats*›
:: ‹*bool1-assn*^{k} *∗$_a$ atom-assn*^{k} *∗$_a$ heuristic-int-assn*^{k} →$_a$ *bool1-assn*›
unfolding *get-next-phase-heur-stats-def heuristic-int-assn-def*
by *sepref*

lemma *get-conflict-count-since-last-restart-stats-alt-def*:
‹*get-conflict-count-since-last-restart-stats* =
(λ(*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase*), *wasted*, φ). *ccount*)›
by *auto*

sepref-def *get-conflict-count-since-last-restart-stats-impl*
is ‹*RETURN o get-conflict-count-since-last-restart-stats*›
:: ‹*heuristic-int-assn*^{k} →$_a$ *word64-assn*›
unfolding *get-conflict-count-since-last-restart-stats-alt-def heuristic-int-assn-def*
by *sepref*

lemma *next-pure-lits-schedule-info-stats-alt-def*:
‹*next-pure-lits-schedule-info-stats* =
(λ(*fast-ema*, *slow-ema*, -, *wasted*, φ, -,-,*lits*, (*inprocess-schedule*, -), *other-fema*, *other-sema*). *inprocess-schedule*)›
unfolding *next-pure-lits-schedule-info-stats-def next-pure-lits-schedule-info-def*
by *auto*

sepref-def *next-pure-lits-schedule-info-stats-impl*
is ‹*RETURN o next-pure-lits-schedule-info-stats*›
:: ‹*heuristic-int-assn*^{k} →$_a$ *word64-assn*›
unfolding *next-pure-lits-schedule-info-stats-alt-def heuristic-int-assn-def*
by *sepref*

sepref-def *schedule-next-pure-lits-stats-impl*
is ‹*RETURN o schedule-next-pure-lits-stats*›
:: ‹*heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
unfolding *schedule-next-pure-lits-stats-def heuristic-int-assn-def*
by *sepref*

lemma *next-reduce-schedule-info-stats-alt-def*:
‹*next-reduce-schedule-info-stats* =
(λ(*fast-ema*, *slow-ema*, -, *wasted*, φ, -,-,*lits*, (*inprocess-schedule*, *reduce-schedule*, -), -). *reduce-schedule*)›
unfolding *next-reduce-schedule-info-stats-def next-reduce-schedule-info-def*
by (*auto intro*!: *ext*)

sepref-def *next-reduce-schedule-info-stats-impl*
is ‹*RETURN o next-reduce-schedule-info-stats*›
:: ‹*heuristic-int-assn*^{k} →$_a$ *word64-assn*›
unfolding *next-reduce-schedule-info-stats-alt-def heuristic-int-assn-def*
by *sepref*

sepref-def *schedule-next-reduce-stats-impl*
is ‹*uncurry* (*RETURN oo schedule-next-reduce-stats*)›
:: ‹*word64-assn*^{k} *$_a$ *heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
unfolding *schedule-next-reduce-stats-def heuristic-int-assn-def*
by *sepref*

lemma *next-subsume-schedule-info-stats-alt-def*:
‹*next-subsume-schedule-info-stats* =
(λ(*fast-ema*, *slow-ema*, -, *wasted*, φ, -,-,*lits*, (*inprocess-schedule*, *reduce-schedule*, *subsume-schedule*),
-). *subsume-schedule*)›
unfolding *next-subsume-schedule-info-stats-def next-subsume-schedule-info-def*
by (*auto intro*!: *ext*)

sepref-def *next-subsume-schedule-info-stats-impl*
is ‹*RETURN o next-subsume-schedule-info-stats*›
:: ‹*heuristic-int-assn*^{k} →$_a$ *word64-assn*›
unfolding *next-subsume-schedule-info-stats-alt-def heuristic-int-assn-def*
by *sepref*

sepref-def *schedule-next-subsume-stats-impl*
is ‹*uncurry* (*RETURN oo schedule-next-subsume-stats*)›
:: ‹*word64-assn*^{k} *$_a$ *heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
unfolding *schedule-next-subsume-stats-def heuristic-int-assn-def*
by *sepref*

lemma *hn-id-pure*:
‹*CONSTRAINT is-pure A* ⟹ (*Mreturn*, *RETURN o id*) ∈ *A*^{k} →$_a$ *A*›
apply *sepref-to-hoare*
apply *vcg*
apply (*auto simp*: *ENTAILS-def is-pure-conv pure-def*)
by (*smt* (*z3*) *entails-def entails-lift-extract-simps*(*1*) *pure-true-conv sep-conj-empty'*)

lemmas *heur-refine*[*sepref-fr-rules*] =
set-zero-wasted-stats-impl.refine[*FCOMP set-zero-wasted-stats-set-zero-wasted-stats*]
heuristic-reluctant-tick-stats-impl.refine[*FCOMP heuristic-reluctant-tick-stats-heuristic-reluctant-tick*]
heuristic-reluctant-enable-stats-impl.refine[*FCOMP heuristic-reluctant-enable-stats-heuristic-reluctant-enable*]
heuristic-reluctant-disable-stats-impl.refine[*FCOMP heuristic-reluctant-disable-stats-heuristic-reluctant-disable*]
heuristic-reluctant-triggered-stats-impl.refine[*FCOMP heuristic-reluctant-triggered-stats-heuristic-reluctant-triggered*]
heuristic-reluctant-triggered2-stats-impl.refine[*FCOMP heuristic-reluctant-triggered2-stats-heuristic-reluctant-triggered2*
heuristic-reluctant-untrigger-stats-impl.refine[*FCOMP heuristic-reluctant-untrigger-stats-heuristic-reluctant-untrigger*]
end-of-rephasing-phase-heur-stats-impl.refine[*FCOMP end-of-rephasing-phase-heur-stats-end-of-rephasing-phase-heur*]
is-fully-propagated-heur-stats-impl.refine[*FCOMP is-fully-propagated-heur-stats-is-fully-propagated-heur*]
set-fully-propagated-heur-stats-impl.refine[*FCOMP set-fully-propagated-heur-stats-set-fully-propagated-heur*]
unset-fully-propagated-heur-stats-impl.refine[*FCOMP unset-fully-propagated-heur-stats-unset-fully-propagated-heur*]
restart-info-restart-done-heur-stats-impl.refine[*FCOMP restart-info-restart-done-heur-stats-restart-info-restart-done-heu*
set-zero-wasted-impl.refine[*FCOMP set-zero-wasted-stats-set-zero-wasted*]
wasted-of-stats-impl.refine[*FCOMP wasted-of-stats-wasted-of*]
current-restart-phase-impl.refine[*FCOMP current-restart-phase-stats-current-restart-phase*]
slow-ema-of-stats-impl.refine[*FCOMP slow-ema-of-stats-slow-ema-of*]

fast-ema-of-stats-impl.refine[*FCOMP fast-ema-of-stats-fast-ema-of*]
incr-wasted-stats-impl.refine[*FCOMP incr-wasted-stats-incr-wasted*]
swap-emas-stats-impl.refine[*FCOMP swap-emas-stats-swap-emas*]
current-rephasing-phase-stats-impl.refine[*FCOMP current-rephasing-phase-stats-current-rephasing-phase*]
get-next-phase-heur-stats-impl.refine[*FCOMP get-next-phase-heur-stats-get-next-phase-heur*]
get-conflict-count-since-last-restart-stats-impl.refine[*FCOMP get-conflict-count-since-last-restart-stats-get-conflict-count*
schedule-next-pure-lits-stats-impl.refine[*FCOMP schedule-next-pure-lits-stats-schedule-next-pure-lits*]
next-pure-lits-schedule-info-stats-impl.refine[*FCOMP next-pure-lits-schedule-next-pure-lits-schedule-stats*]
schedule-next-reduce-stats-impl.refine[*FCOMP schedule-next-reduce-stats-schedule-next-reduce*]
next-reduce-schedule-info-stats-impl.refine[*FCOMP next-reduce-schedule-next-reduce-schedule-stats*]
schedule-next-subsume-stats-impl.refine[*FCOMP schedule-next-subsume-stats-schedule-next-subsume*]
next-subsume-schedule-info-stats-impl.refine[*FCOMP next-subsume-schedule-next-subsume-schedule-stats*]
hn-id[*of heuristic-int-assn*, *FCOMP get-content-hnr*[*of heur-int-rel*]]
hn-id[*of heuristic-int-assn*, *FCOMP Constructor-hnr*[*of heur-int-rel*]]

lemmas *get-restart-heuristics-pure-rule* =
hn-id-pure[*of heuristic-int-assn*, *FCOMP get-content-hnr*[*of heur-int-rel*]]

end

sepref-register *set-zero-wasted-stats save-phase-heur-stats heuristic-reluctant-tick-stats*
heuristic-reluctant-tick is-fully-propagated-heur-stats get-content next-pure-lits-schedule-info-stats
schedule-next-pure-lits-stats

lemma *mop-save-phase-heur-stats-alt-def*:
‹*mop-save-phase-heur-stats* = (λ *L b* (*fast-ema*, *slow-ema*, *res-info*, *wasted*, (φ, *target*, *best*), *rel*). *do* {
ASSERT(*L* < *length* φ);
RETURN (*fast-ema*, *slow-ema*, *res-info*, *wasted*, (φ[*L* := *b*], *target*,
best), *rel*)})›
unfolding *mop-save-phase-heur-stats-def save-phase-heur-def save-phase-heur-pre-stats-def save-phase-heur-stats-def*
by (*auto intro*!: *ext*)

sepref-def *mop-save-phase-heur-stats-impl*
is ‹*uncurry2* (*mop-save-phase-heur-stats*)›
:: ‹*atom-assn*^{k} ∗$_a$ *bool1-assn*^{k} ∗$_a$ *heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
supply [[*goals-limit=1*]]
unfolding *mop-save-phase-heur-stats-alt-def save-phase-heur-stats-def save-phase-heur-pre-stats-def*
phase-heur-assn-def mop-save-phase-heur-def heuristic-int-assn-def
apply *annot-all-atm-idxs*
by *sepref*

lemma *mop-save-phase-heur-alt-def*:
‹*mop-save-phase-heur L b S* = *do* {
let S = *get-restart-heuristics S*;
S ← *mop-save-phase-heur-stats L b S*;
RETURN (*Restart-Heuristics S*)
}›
unfolding *Let-def mop-save-phase-heur-def mop-save-phase-heur-def save-phase-heur-def*
mop-save-phase-heur-stats-def save-phase-heur-pre-def
by *auto*

sepref-def *mop-save-phase-heur-impl*
is ‹*uncurry2* (*mop-save-phase-heur*)›
:: ‹*atom-assn*^{k} ∗$_a$ *bool1-assn*^{k} ∗$_a$ *heuristic-assn*^{d} →$_a$ *heuristic-assn*›
supply [[*goals-limit=1*]]

unfolding *mop-save-phase-heur-alt-def save-phase-heur-def save-phase-heur-pre-def phase-heur-assn-def mop-save-phase-heur-def*
apply *annot-all-atm-idxs*
by *sepref*

schematic-goal *mk-free-heuristic-int-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE heuristic-int-assn ?fr*›
unfolding *heuristic-int-assn-def code-hider-assn-def*
by *synthesize-free*

schematic-goal *mk-free-heuristic-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE heuristic-assn ?fr*›
unfolding *heuristic-assn-def code-hider-assn-def*
by *synthesize-free*

lemma [*safe-constraint-rules*]: ‹*CONSTRAINT is-pure A* ⟹ *CONSTRAINT is-pure* (*code-hider-assn A B*)›
unfolding *code-hider-assn-def* **by** (*auto simp*: *hr-comp-is-pure*)

lemma *clss-size-incr-lcount-alt-def*:
‹*RETURN o clss-size-incr-lcount* =
(λ(*lcount*, *lcountUE*, *lcountUS*). *RETURN* (*lcount* + *1*, *lcountUE*, *lcountUS*))›
by (*auto simp*: *clss-size-incr-lcount-def*)

lemma *clss-size-lcountUEk-alt-def*:
‹*RETURN o clss-size-lcountUEk* = (λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*). *RETURN lcountUEk*)›
by (*auto simp*: *clss-size-lcountUEk-def*)

sepref-def *clss-size-lcountUEk-fast-code*
is ‹*RETURN o clss-size-lcountUEk*›
:: ‹*lcount-assn*^{k} →$_a$ *uint64-nat-assn*›
unfolding *lcount-assn-def clss-size-lcountUEk-alt-def clss-size-lcount-def*
by *sepref*

sepref-register *clss-size-incr-lcount*
sepref-def *clss-size-incr-lcount-fast-code*
is ‹*RETURN o clss-size-incr-lcount*›
:: ‹[λ*S*. *clss-size-lcount S* ≤ *max-snat 64*]$_a$ *lcount-assn*^{d} → *lcount-assn*›
unfolding *clss-size-incr-lcount-alt-def lcount-assn-def clss-size-lcount-def*
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *end-of-restart-phase-impl*
is ‹*RETURN o end-of-restart-phase-stats*›
:: ‹*heuristic-int-assn*^{k} →$_a$ *word-assn*›
unfolding *end-of-restart-phase-stats-def heuristic-int-assn-def*
by *sepref*

lemma *end-of-restart-phase-stats-end-of-restart-phase*:
‹(*end-of-restart-phase-stats*, *end-of-restart-phase*) ∈ *heur-rel* → *word-rel*›
by (*auto simp*: *end-of-restart-phase-def code-hider-rel-def*)

lemmas *end-of-restart-phase-impl-refine*[*sepref-fr-rules*] =
end-of-restart-phase-impl.refine[*FCOMP end-of-restart-phase-stats-end-of-restart-phase*,
unfolded heuristic-assn-alt-def[*symmetric*]]

lemma *incr-restart-phase-end-stats-alt-def*:
‹*incr-restart-phase-end-stats* = (λ*end-of-phase* (*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase*, -, *length-phase*), *wasted*).
(*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase* + *length-phase*, (*length-phase* ∗ *3*) >> *1*), *wasted*))›
by (*auto intro*!: *ext*)

sepref-def *incr-restart-phase-end-stats-impl* [*llvm-inline*]
is ‹*uncurry* (*RETURN oo incr-restart-phase-end-stats*)›
:: ‹*word64-assn*^{k} ∗$_a$ *heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
supply[[*goals-limit=1*]]
unfolding *heuristic-int-assn-def incr-restart-phase-end-stats-alt-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *incr-restart-phase-end-impl*
is ‹*uncurry* (*RETURN oo incr-restart-phase-end*)›
:: ‹*word64-assn*^{k} ∗$_a$ *heuristic-assn*^{d} →$_a$ *heuristic-assn*›
supply[[*goals-limit=1*]]
unfolding *incr-restart-phase-end-def*
by *sepref*

lemma *incr-restart-phase-stats-alt-def*:
‹*incr-restart-phase-stats* =
(λ(*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase*, *end-of-phase*), *wasted*, *φ*).
(*fast-ema*, *slow-ema*, (*ccount*, *ema-lvl*, *restart-phase XOR 1*, *end-of-phase*), *wasted*, *φ*))›
by (*auto*)

sepref-def *incr-restart-phase-stats-impl*
is ‹*RETURN o incr-restart-phase-stats*›
:: ‹*heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
unfolding *heuristic-int-assn-def incr-restart-phase-stats-alt-def*
by *sepref*

sepref-def *incr-restart-phase-impl*
is ‹*RETURN o incr-restart-phase*›
:: ‹*heuristic-assn*^{d} →$_a$ *heuristic-assn*›
unfolding *incr-restart-phase-def*
by *sepref*

sepref-def *reset-added-heur-stats2-impl*
is *reset-added-heur-stats2*
:: ‹*heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
unfolding *reset-added-heur-stats2-def heuristic-int-assn-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *reset-added-heur-stats2-reset-added-heur-stats*:
‹(*reset-added-heur-stats2*, *RETURN o reset-added-heur-stats*) ∈ *heur-int-rel* → ⟨*heur-int-rel*⟩*nres-rel*›
unfolding *fref-param1*
apply (*intro frefI fref-param1 nres-relI*)
apply (*subst comp-def*)
apply (*rule reset-added-heur-stats2-reset-added-heur-stats*[*THEN order-trans*])
by *simp*

lemmas *reset-added-heur-stats-impl-refine*[*sepref-fr-rules*] =
reset-added-heur-stats2-impl.refine[*FCOMP reset-added-heur-stats2-reset-added-heur-stats*]

sepref-register *reset-added-heur mop-reset-added-heur is-marked-added-heur*

sepref-def *is-marked-added-heur-stats-impl*
is ‹*uncurry* (*mop-is-marked-added-heur-stats*)›
:: ‹*heuristic-int-assn*^{k} $*_a$ *atom-assn*^{k} $\rightarrow_a$ *bool1-assn*›
supply [[*eta-contract=false*]]
unfolding *is-marked-added-heur-stats-def is-marked-added-heur-pre-stats-def*
heuristic-int-assn-def mop-is-marked-added-heur-stats-def case-prod-app
apply (*rewrite at* ‹- ! -› *annot-index-of-atm*)
by *sepref*

lemma *mop-mark-added-heur-stats-alt-def*:
‹*mop-mark-added-heur-stats L b* =(λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant*, *fully-proped*, *lits-st*). *do* {
ASSERT(*mark-added-heur-pre-stats L b* (*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant*, *fully-proped*, *lits-st*));
RETURN (*mark-added-heur-stats L b* (*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant*, *fully-proped*, *lits-st*))
})›
by (*auto simp*: *mop-mark-added-heur-stats-def*)

sepref-def *mop-mark-added-heur-stats-impl*
is ‹*uncurry2 mop-mark-added-heur-stats*›
:: ‹*atom-assn*^{k} $*_a$ *bool1-assn*^{k} $*_a$ *heuristic-int-assn*^{d} $\rightarrow_a$ *heuristic-int-assn*›
unfolding *heuristic-int-assn-def mop-mark-added-heur-stats-alt-def*
mark-added-heur-stats-def prod.simps mark-added-heur-pre-stats-def
apply (*rewrite at* ‹-[- := -]› *annot-index-of-atm*)
by *sepref*

lemma *mop-is-marked-added-heur-stats-alt-def*:
‹*mop-is-marked-added-heur-stats* =(λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant*, *fully-proped*, *lits-st*, *schedule*) *L*. *do* {
ASSERT(*is-marked-added-heur-pre-stats* (*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant*, *fully-proped*, *lits-st*, *schedule*) *L*);
RETURN (*is-marked-added-heur-stats* (*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, *reluctant*, *fully-proped*, *lits-st*, *schedule*) *L*)
})›
by (*auto simp*: *mop-is-marked-added-heur-stats-def intro*!: *ext*)

sepref-def *mop-is-marked-added-heur-stats-impl*
is ‹*uncurry mop-is-marked-added-heur-stats*›
:: ‹*heuristic-int-assn*^{k} $*_a$ *atom-assn*^{k} $\rightarrow_a$ *bool1-assn*›
unfolding *heuristic-int-assn-def mop-is-marked-added-heur-stats-alt-def*
is-marked-added-heur-stats-def prod.simps is-marked-added-heur-pre-stats-def
apply (*rewrite at* ‹- ! -› *annot-index-of-atm*)
by *sepref*

context
notes [*fcomp-norm-unfold*] = *heuristic-assn-def*[*symmetric*] *heuristic-assn-alt-def*[*symmetric*]
begin

lemma *reset-added-heur-stats-reset-added-heur*:
  ‹(*reset-added-heur-stats*, *reset-added-heur*) ∈ *heur-rel* → *heur-rel*› **and**
  is-marked-added-heur-stats-is-marked-added-heur:
  ‹(*is-marked-added-heur-stats*, *is-marked-added-heur*) ∈ *heur-rel* → *nat-rel* → *bool-rel*›
  by (*auto simp*: *reset-added-heur-def code-hider-rel-def is-marked-added-heur-def*
    mop-is-marked-added-heur-stats-def)

lemmas *reset-added-heur-refine*[*sepref-fr-rules*] =
  reset-added-heur-stats-impl-refine[*FCOMP reset-added-heur-stats-reset-added-heur*]

lemma *mop-is-marked-added-heur-stats-is-marked-added-heur*:
  ‹(*uncurry mop-is-marked-added-heur-stats*, *uncurry* (*RETURN oo is-marked-added-heur*)) ∈
  [λ(*S*, *l*). *is-marked-added-heur-pre-stats* (*get-restart-heuristics S*) *l*]$_f$
  heur-rel ×$_f$ *nat-rel* → ⟨*bool-rel*⟩*nres-rel*› **and**
  mop-mark-added-heur-stats-mop-mark-added-heur:
  ‹(*uncurry2 mop-mark-added-heur-stats*, *uncurry2* (*RETURN ooo mark-added-heur*)) ∈
  [λ((*l*,*b*), *S*). *mark-added-heur-pre l b S*]$_f$
  nat-rel ×$_f$ *bool-rel* ×$_f$ *heur-rel* → ⟨*heur-rel*⟩*nres-rel*› **and**
  mop-is-marked-added-heur-stats-mop-is-marked-added-heur:
  ‹(*uncurry mop-is-marked-added-heur-stats*, *uncurry* (*RETURN oo is-marked-added-heur*)) ∈
  [λ(*l*, *S*). *is-marked-added-heur-pre l S*]$_f$
   heur-rel ×$_f$ *nat-rel* → ⟨*bool-rel*⟩*nres-rel*›
  by (*auto intro*!: *frefI nres-relI*
    simp: *reset-added-heur-def code-hider-rel-def is-marked-added-heur-def*
    mop-mark-added-heur-stats-def mop-is-marked-added-heur-stats-def
    mop-is-marked-added-heur-stats-def is-marked-added-heur-pre-def
    mark-added-heur-pre-def
    mop-mark-added-heur-stats-def mark-added-heur-def)

lemmas *is-marked-added-heur-stats-impl-refine*[*refine*] =
  is-marked-added-heur-stats-impl.refine[*FCOMP mop-is-marked-added-heur-stats-is-marked-added-heur*]

lemmas *mark-added-heur-impl-refine*[*refine*] =
  mop-mark-added-heur-stats-impl.refine[*FCOMP mop-mark-added-heur-stats-mop-mark-added-heur*]

lemmas *is-marked-added-heur-refine*[*refine*] =
  mop-is-marked-added-heur-stats-impl.refine[*FCOMP mop-is-marked-added-heur-stats-mop-is-marked-added-heur*]
end

sepref-def *mop-reset-added-heur-impl*
  is ‹*mop-reset-added-heur*›
  :: ‹*heuristic-assn*^{d} →$_a$ *heuristic-assn*›
  unfolding *mop-reset-added-heur-def*
  by *sepref*

end
theory *Watched-Literals-VMTF*
  imports *IsaSAT-Literals*
begin

5.4.3 Variable-Move-to-Front

Specification

type-synonym *'v abs-vmtf-ns* = ‹*'v set* × *'v set*›

datatype ('v, 'n) *vmtf-node* = *VMTF-Node* (*stamp* : 'n) (*get-prev*: ‹'v *option*›) (*get-next*: ‹'v *option*›)
type-synonym *nat-vmtf-node* = ‹(*nat*, *nat*) *vmtf-node*›

inductive *vmtf-ns* :: ‹*nat list* ⇒ *nat* ⇒ *nat-vmtf-node list* ⇒ *bool*› **where**
Nil: ‹*vmtf-ns* [] *st xs*› |
Cons1: ‹*a* < *length xs* ⟹ *m* ≥ *n* ⟹ *xs* ! *a* = *VMTF-Node* (*n*::*nat*) *None None* ⟹ *vmtf-ns* [*a*] *m xs*›
|
Cons: ‹*vmtf-ns* (*b* # *l*) *m xs* ⟹ *a* < *length xs* ⟹ *xs* ! *a* = *VMTF-Node n None* (*Some b*) ⟹
a ≠ *b* ⟹ *a* ∉ *set l* ⟹ *n* > *m* ⟹
xs' = *xs*[*b* := *VMTF-Node* (*stamp* (*xs*!*b*)) (*Some a*) (*get-next* (*xs*!*b*))] ⟹ *n'* ≥ *n* ⟹
vmtf-ns (*a* # *b* # *l*) *n' xs'*›

inductive-cases *vmtf-nsE*: ‹*vmtf-ns xs st zs*›

lemma *vmtf-ns-le-length*: ‹*vmtf-ns l m xs* ⟹ *i* ∈ *set l* ⟹ *i* < *length xs*›
apply (*induction rule*: *vmtf-ns.induct*)
subgoal by (*auto intro*: *vmtf-ns.intros*)
subgoal by (*auto intro*: *vmtf-ns.intros*)
subgoal by (*auto intro*: *vmtf-ns.intros*)
done

lemma *vmtf-ns-distinct*: ‹*vmtf-ns l m xs* ⟹ *distinct l*›
apply (*induction rule*: *vmtf-ns.induct*)
subgoal by (*auto intro*: *vmtf-ns.intros*)
subgoal by (*auto intro*: *vmtf-ns.intros*)
subgoal by (*auto intro*: *vmtf-ns.intros*)
done

lemma *vmtf-ns-eq-iff*:
assumes
‹∀ *i* ∈ *set l*. *xs* ! *i* = *zs* ! *i*› **and**
‹∀ *i* ∈ *set l*. *i* < *length xs* ∧ *i* < *length zs*›
shows ‹*vmtf-ns l m zs* ⟷ *vmtf-ns l m xs*› (**is** ‹*?A* ⟷ *?B*›)
proof −
have ‹*vmtf-ns l m xs*›
if
‹*vmtf-ns l m zs*› **and**
‹(∀ *i* ∈ *set l*. *xs* ! *i* = *zs* ! *i*)› **and**
‹(∀ *i* ∈ *set l*. *i* < *length xs* ∧ *i* < *length zs*)›
for *xs zs*
using *that*
proof (*induction arbitrary*: *xs rule*: *vmtf-ns.induct*)
case (*Nil st xs zs*)
then show *?case* **by** (*auto intro*: *vmtf-ns.intros*)
next
case (*Cons1 a xs n zs*)
show *?case* **by** (*rule vmtf-ns.Cons1*) (*use Cons1* **in** ‹*auto intro*: *vmtf-ns.intros*›)
next
case (*Cons b l m xs c n zs n' zs'*) **note** *vmtf-ns* = *this*(*1*) **and** *a-le-y* = *this*(*2*) **and** *zs-a* = *this*(*3*)
and *ab* = *this*(*4*) **and** *a-l* = *this*(*5*) **and** *mn* = *this*(*6*) **and** *xs'* = *this*(*7*) **and** *nn'* = *this*(*8*) **and**
IH = *this*(*9*) **and** *H* = *this*(*10*−)
have ‹*vmtf-ns* (*c* # *b* # *l*) *n' zs*›
by (*rule vmtf-ns.Cons*[*OF Cons.hyps*])
have [*simp*]: ‹*b* < *length xs*› ‹*b* < *length zs*›
using *H xs'* **by** *auto*

```
    have [simp]: ‹b ∉ set l›
      using vmtf-ns-distinct[OF vmtf-ns] by auto
    then have K: ‹∀ i∈set l. zs ! i = (if b = i then x else xs ! i) =
       (∀ i∈set l. zs ! i = xs ! i)› for x
       using H(2)
       by (simp add: H(1) xs')
    have next-xs-b: ‹get-next (xs ! b) = None› if ‹l = []›
      using vmtf-ns unfolding that by (auto simp: elim!: vmtf-nsE)
    have prev-xs-b: ‹get-prev (xs ! b) = None›
      using vmtf-ns by (auto elim: vmtf-nsE)
    have vmtf-ns-zs: ‹vmtf-ns (b # l) m (zs'[b := xs!b])›
      apply (rule IH)
      subgoal using H(1) ab next-xs-b prev-xs-b H unfolding xs' by (auto simp: K)
      subgoal using H(2) ab next-xs-b prev-xs-b unfolding xs' by (auto simp: K)
      done
    have ‹zs' ! b = VMTF-Node (stamp (xs ! b)) (Some c) (get-next (xs ! b))›
      using H(1) unfolding xs' by auto
    show ?case
      apply (rule vmtf-ns.Cons[OF vmtf-ns-zs, of - n])
      subgoal using a-le-y xs' H(2) by auto
      subgoal using ab zs-a xs' H(1) by (auto simp: K)
      subgoal using ab .
      subgoal using a-l .
      subgoal using mn .
      subgoal using ab xs' H(1) by (metis H(2) insert-iff list.set(2) list-update-id
          list-update-overwrite nth-list-update-eq)
      subgoal using nn' .
      done
  qed
  then show ?thesis
    using assms by metis
qed

lemmas vmtf-ns-eq-iffI = vmtf-ns-eq-iff[THEN iffD1]

lemma vmtf-ns-stamp-increase: ‹vmtf-ns xs p zs ⟹ p ≤ p' ⟹ vmtf-ns xs p' zs›
  apply (induction rule: vmtf-ns.induct)
  subgoal by (auto intro: vmtf-ns.intros)
  subgoal by (rule vmtf-ns.Cons1) (auto intro!: vmtf-ns.intros)
  subgoal by (auto intro: vmtf-ns.intros)
  done

lemma vmtf-ns-single-iff: ‹vmtf-ns [a] m xs ⟷ (a < length xs ∧ m ≥ stamp (xs ! a) ∧
    xs ! a = VMTF-Node (stamp (xs ! a)) None None)›
  by (auto 5 5 elim!: vmtf-nsE intro: vmtf-ns.intros)

lemma vmtf-ns-append-decomp:
  assumes ‹vmtf-ns (axs @ [ax, ay] @ azs) an ns›
  shows ‹(vmtf-ns (axs @ [ax]) an (ns[ax:= VMTF-Node (stamp (ns!ax)) (get-prev (ns!ax)) None]) ∧
    vmtf-ns (ay # azs) (stamp (ns!ay)) (ns[ay:= VMTF-Node (stamp (ns!ay)) None (get-next (ns!ay))])
∧
    stamp (ns!ax) > stamp (ns!ay))›
  using assms
proof (induction ‹axs @ [ax, ay] @ azs› an ns arbitrary: axs ax ay azs rule: vmtf-ns.induct)
  case (Nil st xs)
  then show ?case by simp
```

```
next
  case (Cons1 a xs m n)
  then show ?case by auto
next
  case (Cons b l m xs a n xs' n') note vmtf-ns = this(1) and IH = this(2) and a-le-y = this(3) and
    zs-a = this(4) and ab = this(5) and a-l = this(6) and mn = this(7) and xs' = this(8) and
    nn' = this(9) and decomp = this(10-)
  have b-le-xs: ‹b < length xs›
    using vmtf-ns by (auto intro: vmtf-ns-le-length simp: xs')
  show ?case
  proof (cases ‹axs›)
    case [simp]: Nil
    then have [simp]: ‹ax = a› ‹ay = b› ‹azs = l›
      using decomp by auto
    show ?thesis
    proof (cases l)
      case Nil
      then show ?thesis
        using vmtf-ns xs' a-le-y zs-a ab a-l mn nn' by (cases ‹xs ! b›)
          (auto simp: vmtf-ns-single-iff)
    next
      case (Cons al als) note l = this
      have vmtf-ns-b: ‹vmtf-ns [b] m (xs[b := VMTF-Node (stamp (xs ! b)) (get-prev (xs ! b)) None])›
and
        vmtf-ns-l: ‹vmtf-ns (al # als) (stamp (xs ! al))
          (xs[al := VMTF-Node (stamp (xs ! al)) None (get-next (xs ! al))])› and
        stamp-al-b: ‹stamp (xs ! al) < stamp (xs ! b)›
        using IH[of Nil b al als] unfolding l by auto
      have ‹vmtf-ns [a] n' (xs'[a := VMTF-Node (stamp (xs' ! a)) (get-prev (xs' ! a)) None])›
        using a-le-y xs' ab mn nn' zs-a by (auto simp: vmtf-ns-single-iff)
      have al-b[simp]: ‹al ≠ b› and b-als: ‹b ∉ set als›
        using vmtf-ns unfolding l by (auto dest: vmtf-ns-distinct)
      have al-le-xs: ‹al < length xs›
        using vmtf-ns vmtf-ns-l by (auto intro: vmtf-ns-le-length simp: l xs')
      have xs-al: ‹xs ! al = VMTF-Node (stamp (xs ! al)) (Some b) (get-next (xs ! al))›
        using vmtf-ns unfolding l by (auto 5 5 elim: vmtf-nsE)
      have xs-b: ‹xs ! b = VMTF-Node (stamp (xs ! b)) None (get-next (xs ! b))›
        using vmtf-ns-b vmtf-ns xs' by (cases ‹xs ! b›) (auto elim: vmtf-nsE simp: l vmtf-ns-single-iff)

      have ‹vmtf-ns (b # al # als) (stamp (xs' ! b))
          (xs'[b := VMTF-Node (stamp (xs' ! b)) None (get-next (xs' ! b))])›
        apply (rule vmtf-ns.Cons[OF vmtf-ns-l, of - ‹stamp (xs' ! b)›])
        subgoal using b-le-xs by auto
        subgoal using xs-b vmtf-ns-b vmtf-ns xs' by (cases ‹xs ! b›)
            (auto elim: vmtf-nsE simp: l vmtf-ns-single-iff)
        subgoal using al-b by blast
        subgoal using b-als .
        subgoal using xs' b-le-xs stamp-al-b by (simp add:)
        subgoal using ab unfolding xs' by (simp add: b-le-xs al-le-xs xs-al[symmetric]
              xs-b[symmetric])
        subgoal by simp
        done
      moreover have ‹vmtf-ns [a] n'
          (xs'[a := VMTF-Node (stamp (xs' ! a)) (get-prev (xs' ! a)) None])›
        using ab a-le-y mn nn' zs-a by (auto simp: vmtf-ns-single-iff xs')
      moreover have ‹stamp (xs' ! b) < stamp (xs' ! a)›
```

```
    using b-le-xs ab mn vmtf-ns-b zs-a by (auto simp add: xs' vmtf-ns-single-iff)
   ultimately show ?thesis
    unfolding l by (simp add: l)
  qed
next
  case (Cons aaxs axs') note axs = this
  have [simp]: ‹aaxs = a› and bl: ‹b # l = axs' @ [ax, ay] @ azs›
   using decomp unfolding axs by simp-all
  have
   vmtf-ns-axs': ‹vmtf-ns (axs' @ [ax]) m
    (xs[ax := VMTF-Node (stamp (xs ! ax)) (get-prev (xs ! ax)) None])› and
   vmtf-ns-ay: ‹vmtf-ns (ay # azs) (stamp (xs ! ay))
    (xs[ay := VMTF-Node (stamp (xs ! ay)) None (get-next (xs ! ay))])› and
   stamp: ‹stamp (xs ! ay) < stamp (xs ! ax)›
   using IH[OF bl] by fast+
  have b-ay: ‹b ≠ ay›
   using bl vmtf-ns-distinct[OF vmtf-ns] by (cases axs') auto
  have vmtf-ns-ay': ‹vmtf-ns (ay # azs) (stamp (xs' ! ay))
    (xs[ay := VMTF-Node (stamp (xs ! ay)) None (get-next (xs ! ay))])›
   using vmtf-ns-ay xs' b-ay by (auto)
  have [simp]: ‹ay < length xs›
    using vmtf-ns by (auto intro: vmtf-ns-le-length simp: bl xs')
  have in-azs-noteq-b: ‹i ∈ set azs ⟹ i ≠ b› for i
   using vmtf-ns-distinct[OF vmtf-ns] bl by (cases axs') (auto simp: xs' b-ay)
  have a-ax[simp]: ‹a ≠ ax›
   using ab a-l bl by (cases axs') (auto simp: xs' b-ay)
  have ‹vmtf-ns (axs @ [ax]) n'
    (xs'[ax := VMTF-Node (stamp (xs' ! ax)) (get-prev (xs' ! ax)) None])›
  proof (cases axs')
   case Nil
   then have [simp]: ‹ax = b›
    using bl by auto
   have ‹vmtf-ns [ax] m (xs[ax := VMTF-Node (stamp (xs ! ax)) (get-prev (xs ! ax)) None])›
    using vmtf-ns-axs' unfolding axs Nil by simp
   then have ‹vmtf-ns (aaxs # ax # []) n'
     (xs'[ax := VMTF-Node (stamp (xs' ! ax)) (get-prev (xs' ! ax)) None])›
    apply (rule vmtf-ns.Cons[of - - - - - n])
    subgoal using a-le-y by auto
    subgoal using zs-a a-le-y ab by auto
    subgoal using ab by auto
    subgoal by simp
    subgoal using mn .
    subgoal using zs-a a-le-y ab xs' b-le-xs by auto
    subgoal using nn' .
    done
   then show ?thesis
    using vmtf-ns-axs' unfolding axs Nil by simp
  next
   case (Cons aaaxs' axs'')
   have [simp]: ‹aaaxs' = b›
    using bl unfolding Cons by auto
   have ‹vmtf-ns (aaaxs' # axs'' @ [ax]) m
     (xs[ax := VMTF-Node (stamp (xs ! ax)) (get-prev (xs ! ax)) None])›
    using vmtf-ns-axs' unfolding axs Cons by simp
   then have ‹vmtf-ns (a # aaaxs' # axs'' @ [ax]) n'
     (xs'[ax := VMTF-Node (stamp (xs' ! ax)) (get-prev (xs' ! ax)) None])›
```

```
      apply (rule vmtf-ns.Cons[of - - - - - n])
      subgoal using a-le-y by auto
      subgoal using zs-a a-le-y a-ax ab by (auto simp del: ‹a ≠ ax›)
      subgoal using ab by auto
      subgoal using a-l bl unfolding Cons by simp
      subgoal using mn .
      subgoal using zs-a a-le-y ab xs' b-le-xs by (auto simp: list-update-swap)
      subgoal using nn' .
      done
    then show ?thesis
      unfolding axs Cons by simp
  qed
  moreover have ‹vmtf-ns (ay # azs) (stamp (xs' ! ay))
      (xs'[ay := VMTF-Node (stamp (xs' ! ay)) None (get-next (xs' ! ay))])›
    apply (rule vmtf-ns-eq-iffI[OF - - vmtf-ns-ay'])
    subgoal using vmtf-ns-distinct[OF vmtf-ns] bl b-le-xs in-azs-noteq-b by (auto simp: xs' b-ay)
    subgoal using vmtf-ns-le-length[OF vmtf-ns] bl unfolding xs' by auto
    done
  moreover have ‹stamp (xs' ! ay) < stamp (xs' ! ax)›
    using stamp unfolding axs xs' by (auto simp: b-le-xs b-ay)
  ultimately show ?thesis
    unfolding axs xs' by fast
 qed
qed

lemma vmtf-ns-append-rebuild:
 assumes
  ‹(vmtf-ns (axs @ [ax]) an ns) › and
  ‹vmtf-ns (ay # azs) (stamp (ns!ay)) ns› and
  ‹stamp (ns!ax) > stamp (ns!ay)› and
  ‹distinct (axs @ [ax, ay] @ azs)›
 shows ‹vmtf-ns (axs @ [ax, ay] @ azs) an
  (ns[ax := VMTF-Node (stamp (ns!ax)) (get-prev (ns!ax)) (Some ay) ,
    ay := VMTF-Node (stamp (ns!ay)) (Some ax) (get-next (ns!ay))])›
 using assms
proof (induction ‹axs @ [ax]› an ns arbitrary: axs ax ay azs rule: vmtf-ns.induct)
 case (Nil st xs)
 then show ?case by simp
next
 case (Cons1 a xs m n) note a-le-xs = this(1) and nm = this(2) and xs-a = this(3) and a = this(4)
  and vmtf-ns = this(5) and stamp = this(6) and dist = this(7)
 have a-ax: ‹ax = a›
  using a by simp

 have vmtf-ns-ay': ‹vmtf-ns (ay # azs) (stamp (xs ! ay)) (xs[ax := VMTF-Node n None (Some ay)])›
  apply (rule vmtf-ns-eq-iffI[OF - - vmtf-ns])
  subgoal using dist a-ax a-le-xs by auto
  subgoal using vmtf-ns vmtf-ns-le-length by auto
  done

 then have ‹vmtf-ns (ax # ay # azs) m (xs[ax := VMTF-Node n None (Some ay),
    ay := VMTF-Node (stamp (xs ! ay)) (Some ax) (get-next (xs ! ay))])›
  apply (rule vmtf-ns.Cons[of - - - - - ‹stamp (xs ! a)›])
  subgoal using a-le-xs unfolding a-ax by auto
  subgoal using xs-a a-ax a-le-xs by auto
  subgoal using dist by auto
```

```
    subgoal using dist by auto
    subgoal using stamp by (simp add: a-ax)
    subgoal using a-ax a-le-xs dist by auto
    subgoal by (simp add: nm xs-a)
    done
  then show ?case
    using a-ax a xs-a by auto
next
  case (Cons b l m xs a n xs' n') note vmtf-ns = this(1) and IH = this(2) and a-le-y = this(3) and
    zs-a = this(4) and ab = this(5) and a-l = this(6) and mn = this(7) and xs' = this(8) and
    nn' = this(9) and decomp = this(10) and vmtf-ns-ay = this(11) and stamp = this(12) and
    dist = this(13)

  have dist-b: ‹distinct ((a # b # l) @ ay # azs)›
    using dist unfolding decomp by auto
  then have b-ay: ‹b ≠ ay›
    by auto
  have b-le-xs: ‹b < length xs›
    using vmtf-ns vmtf-ns-le-length by auto
  have a-ax: ‹a ≠ ax› and a-ay: ‹a ≠ ay›
    using dist-b decomp dist by (cases axs; auto)+
  have vmtf-ns-ay': ‹vmtf-ns (ay # azs) (stamp (xs ! ay)) xs›
    apply (rule vmtf-ns-eq-iffI[of - - xs'])
    subgoal using xs' b-ay dist-b b-le-xs by auto
    subgoal using vmtf-ns-le-length[OF vmtf-ns-ay] xs' by auto
    subgoal using xs' b-ay dist-b b-le-xs vmtf-ns-ay xs' by auto
    done

  have ‹vmtf-ns (tl axs @ [ax, ay] @ azs) m
        (xs[ax := VMTF-Node (stamp (xs ! ax)) (get-prev (xs ! ax)) (Some ay),
            ay := VMTF-Node (stamp (xs ! ay)) (Some ax) (get-next (xs ! ay))])›
    apply (rule IH)
    subgoal using decomp by (cases axs) auto
    subgoal using vmtf-ns-ay' .
    subgoal using stamp xs' b-ay b-le-xs by (cases ‹ax = b›) auto
    subgoal using dist by (cases axs) auto
    done
  moreover have ‹tl axs @ [ax, ay] @ azs = b # l @ ay # azs›
    using decomp by (cases axs) auto
  ultimately have vmtf-ns-tl-axs: ‹vmtf-ns (b # l @ ay # azs) m
        (xs[ax := VMTF-Node (stamp (xs ! ax)) (get-prev (xs ! ax)) (Some ay),
            ay := VMTF-Node (stamp (xs ! ay)) (Some ax) (get-next (xs ! ay))])›
    by metis

  then have ‹vmtf-ns (a # b # l @ ay # azs) n'
     (xs'[ax := VMTF-Node (stamp (xs' ! ax)) (get-prev (xs' ! ax)) (Some ay),
          ay := VMTF-Node (stamp (xs' ! ay)) (Some ax) (get-next (xs' ! ay))])›
    apply (rule vmtf-ns.Cons[of - - - - - ‹stamp (xs ! a)›])
    subgoal using a-le-y by simp
    subgoal using zs-a a-le-y a-ax a-ay by auto
    subgoal using ab .
    subgoal using dist-b by auto
    subgoal using mn by (simp add: zs-a)
    subgoal using zs-a a-le-y a-ax a-ay b-ay b-le-xs unfolding xs'
      by (auto simp: list-update-swap)
    subgoal using stamp xs' nn' b-ay b-le-xs zs-a by auto
```

```
    done
  then show ?case
    by (metis append.assoc append-Cons append-Nil decomp)
qed
```

It is tempting to remove the *update-x*. However, it leads to more complicated reasoning later: What happens if x is not in the list, but its successor is? Moreover, it is unlikely to really make a big difference (performance-wise).

```
definition ns-vmtf-dequeue :: ‹nat ⇒ nat-vmtf-node list ⇒ nat-vmtf-node list› where
‹ns-vmtf-dequeue y xs =
  (let x = xs ! y;
   u-prev =
      (case get-prev x of None ⇒ xs
      | Some a ⇒ xs[a:= VMTF-Node (stamp (xs!a)) (get-prev (xs!a)) (get-next x)]);
   u-next =
      (case get-next x of None ⇒ u-prev
      | Some a ⇒ u-prev[a:= VMTF-Node (stamp (u-prev!a)) (get-prev x) (get-next (u-prev!a))]);
    u-x = u-next[y:= VMTF-Node (stamp (u-next!y)) None None]
    in
   u-x)
  ›

lemma vmtf-ns-different-same-neq: ‹vmtf-ns (b # c # l') m xs ⟹ vmtf-ns (c # l') m xs ⟹ False›
  apply (cases l')
  subgoal by (force elim: vmtf-nsE)
  subgoal for x xs
    apply (subst (asm) vmtf-ns.simps)
    apply (subst (asm)(2) vmtf-ns.simps)
    by (metis (no-types, lifting) vmtf-node.inject length-list-update list.discI list-tail-coinc
        nth-list-update-eq nth-list-update-neq option.discI)
  done

lemma vmtf-ns-last-next:
  ‹vmtf-ns (xs @ [x]) m ns ⟹ get-next (ns ! x) = None›
  apply (induction ‹xs @ [x]› m ns arbitrary: xs x rule: vmtf-ns.induct)
  subgoal by auto
  subgoal by auto
  subgoal for b l m xs a n xs' n' xsa x
    by (cases ‹xs ! b›; cases ‹x = b›; cases xsa)
       (force simp: vmtf-ns-le-length)+
  done

lemma vmtf-ns-hd-prev:
  ‹vmtf-ns (x # xs) m ns ⟹ get-prev (ns ! x) = None›
  apply (induction ‹x # xs› m ns arbitrary: xs x rule: vmtf-ns.induct)
  subgoal by auto
  subgoal by auto
  done

lemma vmtf-ns-last-mid-get-next:
  ‹vmtf-ns (xs @ [x, y] @ zs) m ns ⟹ get-next (ns ! x) = Some y›
  apply (induction ‹xs @ [x, y] @ zs› m ns arbitrary: xs x rule: vmtf-ns.induct)
  subgoal by auto
  subgoal by auto
  subgoal for b l m xs a n xs' n' xsa x
```

```
    by (cases ‹xs ! b›; cases ‹x = b›; cases xsa)
       (force simp: vmtf-ns-le-length)+
  done

lemma vmtf-ns-last-mid-get-next-option-hd:
  ‹vmtf-ns (xs @ x # zs) m ns ⟹ get-next (ns ! x) = option-hd zs›
  using vmtf-ns-last-mid-get-next[of xs x ‹hd zs› ‹tl zs› m ns]
  vmtf-ns-last-next[of xs x]
  by (cases zs) auto

lemma vmtf-ns-last-mid-get-prev:
  assumes ‹vmtf-ns (xs @ [x, y] @ zs) m ns›
  shows ‹get-prev (ns ! y) = Some x›
    using assms
proof (induction ‹xs @ [x, y] @ zs› m ns arbitrary: xs x rule: vmtf-ns.induct)
  case (Nil st xs)
  then show ?case by auto
next
  case (Cons1 a xs m n)
  then show ?case by auto
next
  case (Cons b l m xxs a n xxs' n') note vmtf-ns = this(1) and IH = this(2) and a-le-y = this(3) and
    zs-a = this(4) and ab = this(5) and a-l = this(6) and mn = this(7) and xs' = this(8) and
    nn' = this(9) and decomp = this(10)
  show ?case
  proof (cases xs)
    case Nil
    then show ?thesis using Cons vmtf-ns-le-length by auto
  next
    case (Cons aaxs axs')
    then have b-l: ‹b # l = tl xs @ [x, y] @ zs›
      using decomp by auto
    then have ‹get-prev (xxs ! y) = Some x›
      by (rule IH)
    moreover have ‹x ≠ y›
      using vmtf-ns-distinct[OF vmtf-ns] b-l by auto
    moreover have ‹b ≠ y›
      using vmtf-ns-distinct[OF vmtf-ns] decomp by (cases axs') (auto simp add: Cons)
    moreover have ‹y < length xxs› ‹b < length xxs›
      using vmtf-ns-le-length[OF vmtf-ns, unfolded b-l] vmtf-ns-le-length[OF vmtf-ns] by auto
    ultimately show ?thesis
      unfolding xs' by auto
  qed
qed

lemma vmtf-ns-last-mid-get-prev-option-last:
  ‹vmtf-ns (xs @ x # zs) m ns ⟹ get-prev (ns ! x) = option-last xs›
  using vmtf-ns-last-mid-get-prev[of ‹butlast xs› ‹last xs› ‹x› ‹zs› m ns]
  by (cases xs rule: rev-cases) (auto elim: vmtf-nsE)

lemma length-ns-vmtf-dequeue[simp]: ‹length (ns-vmtf-dequeue x ns) = length ns›
  unfolding ns-vmtf-dequeue-def by (auto simp: Let-def split: option.splits)

lemma vmtf-ns-skip-fst:
  assumes vmtf-ns: ‹vmtf-ns (x # y' # zs') m ns›
  shows ‹∃ n. vmtf-ns (y' # zs') n (ns[y' := VMTF-Node (stamp (ns ! y')) None (get-next (ns ! y'))])
```

```
∧
    m ≥ n›
  using assms
proof (rule vmtf-nsE, goal-cases)
  case 1
  then show ?case by simp
next
  case (2 a n)
  then show ?case by simp
next
  case (3 b l m xs a n)
  moreover have ‹get-prev (xs ! b) = None›
    using 3(3) by (fastforce elim: vmtf-nsE)
  moreover have ‹b < length xs›
    using 3(3) vmtf-ns-le-length by auto
  ultimately show ?case
    by (cases ‹xs ! b›) (auto simp: eq-commute[of ‹xs ! b›])
qed

definition vmtf-ns-notin where
  ‹vmtf-ns-notin l m xs ⟷ (∀ i<length xs. i∉set l ⟶ (get-prev (xs ! i) = None ∧
      get-next (xs ! i) = None))›

lemma vmtf-ns-notinI:
  ‹(⋀i. i <length xs ⟹ i∉set l ⟹ get-prev (xs ! i) = None ∧
      get-next (xs ! i) = None) ⟹ vmtf-ns-notin l m xs›
  by (auto simp: vmtf-ns-notin-def)

lemma stamp-ns-vmtf-dequeue:
  ‹axs < length zs ⟹ stamp (ns-vmtf-dequeue x zs ! axs) = stamp (zs ! axs)›
  by (cases ‹zs ! (the (get-next (zs ! x)))›; cases ‹(the (get-next (zs ! x))) = axs›;
      cases ‹(the (get-prev (zs ! x))) = axs›; cases ‹zs ! x›)
    (auto simp: nth-list-update' ns-vmtf-dequeue-def Let-def split: option.splits)

lemma sorted-many-eq-append: ‹sorted (xs @ [x, y]) ⟷ sorted (xs @ [x]) ∧ x ≤ y›
  using sorted-append[of ‹xs @ [x]› ‹[y]›] sorted-append[of xs ‹[x]›]
  by force

lemma vmtf-ns-stamp-sorted:
  assumes ‹vmtf-ns l m ns›
  shows ‹sorted (map (λa. stamp (ns!a)) (rev l)) ∧ (∀ a ∈ set l. stamp (ns!a) ≤ m)›
  using assms
proof (induction rule: vmtf-ns.induct)
  case (Cons b l m xs a n xs' n') note vmtf-ns = this(1) and IH = this(9) and a-le-y = this(2) and
    zs-a = this(3) and ab = this(4) and a-l = this(5) and mn = this(6) and xs' = this(7) and
    nn' = this(8)
  have H:
  ‹map (λaa. stamp (xs[b := VMTF-Node (stamp (xs ! b)) (Some a) (get-next (xs ! b))] ! aa)) (rev l) =
     map (λa. stamp (xs ! a)) (rev l)›
    apply (rule map-cong)
    subgoal by auto
    subgoal using vmtf-ns-distinct[OF vmtf-ns] vmtf-ns-le-length[OF vmtf-ns] by auto
    done
  have [simp]: ‹stamp (xs[b := VMTF-Node (stamp (xs ! b)) (Some a) (get-next (xs ! b))] ! b) =
     stamp (xs ! b)›
    using vmtf-ns-distinct[OF vmtf-ns] vmtf-ns-le-length[OF vmtf-ns] by (cases ‹xs ! b›) auto
```

```
  have ‹stamp (xs[b := VMTF-Node (stamp (xs ! b)) (Some a) (get-next (xs ! b))] ! aa) ≤ n'›
    if ‹aa ∈ set l› for aa
    apply (cases ‹aa = b›)
    subgoal using Cons by auto
    subgoal using vmtf-ns-distinct[OF vmtf-ns] vmtf-ns-le-length[OF vmtf-ns] IH nn' mn that by auto
    done
  then show ?case
    using Cons by (auto simp: H sorted-many-eq-append)
qed auto

lemma vmtf-ns-ns-vmtf-dequeue:
  assumes vmtf-ns: ‹vmtf-ns l m ns› and notin: ‹vmtf-ns-notin l m ns› and valid: ‹x < length ns›
  shows ‹vmtf-ns (remove1 x l) m (ns-vmtf-dequeue x ns)›
proof (cases ‹x ∈ set l›)
  case False
  then have H: ‹remove1 x l = l›
    by (simp add: remove1-idem)
  have simp-is-stupid[simp]: ‹a ∈ set l ⟹ x ∉ set l ⟹ a ≠ x› ‹a ∈ set l ⟹ x ∉ set l ⟹ x ≠ a›
for a x
    by auto
  have
      ‹get-prev (ns ! x) = None › and
      ‹get-next (ns ! x) = None›
    using notin False valid unfolding vmtf-ns-notin-def by auto
  then have vmtf-ns-eq: ‹(ns-vmtf-dequeue x ns) ! a = ns ! a› if ‹a ∈ set l› for a
    using that False valid unfolding vmtf-ns-notin-def ns-vmtf-dequeue-def
    by (cases ‹ns ! (the (get-prev (ns ! x)))›; cases ‹ns ! (the (get-next (ns ! x)))›)
      (auto simp: Let-def split: option.splits)
  show ?thesis
    unfolding H
    apply (rule vmtf-ns-eq-iffI[OF - - vmtf-ns])
    subgoal using vmtf-ns-eq by blast
    subgoal using vmtf-ns-le-length[OF vmtf-ns] by auto
    done
next
  case True
  then obtain xs zs where
    l: ‹l = xs @ x # zs›
    by (meson split-list)
  have r-l: ‹remove1 x l = xs @ zs›
    using vmtf-ns-distinct[OF vmtf-ns] unfolding l by (simp add: remove1-append)
  have dist: ‹distinct l›
    using vmtf-ns-distinct[OF vmtf-ns] .
  have le-length: ‹i ∈ set l ⟹ i < length ns› for i
    using vmtf-ns-le-length[OF vmtf-ns] .
  consider
    (xs-zs-empty) ‹xs = []› and ‹zs = []› |
    (xs-nempty-zs-empty) x' xs' where ‹xs = xs' @ [x']› and ‹zs = []› |
    (xs-empty-zs-nempty) y' zs' where ‹xs = []› and ‹zs = y' # zs'› |
    (xs-zs-nempty) x' y' xs' zs' where  ‹xs = xs' @ [x']› and ‹zs = y' # zs'›
    by (cases xs rule: rev-cases; cases zs)
  then show ?thesis
  proof cases
    case xs-zs-empty
    then show ?thesis
      using vmtf-ns by (auto simp: r-l intro: vmtf-ns.intros)
```

```
  next
    case xs-empty-zs-nempty note xs = this(1) and zs = this(2)
    have [simp]: ‹x ≠ y'› ‹y' ≠ x› ‹x ∉ set zs'›
      using dist unfolding l xs zs by auto
    have prev-next: ‹get-prev (ns ! x) = None› ‹get-next (ns ! x) = option-hd zs›
      using vmtf-ns unfolding l xs zs
      by (cases zs; auto 5 5 simp: option-hd-def elim: vmtf-nsE; fail)+
    then have vmtf': ‹vmtf-ns (y' # zs') m (ns[y' := VMTF-Node (stamp (ns ! y')) None (get-next (ns
! y'))])›
      using vmtf-ns unfolding r-l unfolding l xs zs
      by (auto simp: ns-vmtf-dequeue-def Let-def nth-list-update' zs
          split: option.splits
          intro: vmtf-ns.intros vmtf-ns-stamp-increase dest: vmtf-ns-skip-fst)
    show ?thesis
      apply (rule vmtf-ns-eq-iffI[of - -
            ‹(ns[y' := VMTF-Node (stamp (ns ! y')) None (get-next (ns ! y'))])› m])
      subgoal
        using prev-next unfolding r-l unfolding l xs zs
        by (cases ‹ns ! x›) (auto simp: ns-vmtf-dequeue-def Let-def nth-list-update')
      subgoal
        using prev-next le-length unfolding r-l unfolding l xs zs
        by (cases ‹ns ! x›) auto
      subgoal
        using vmtf' unfolding r-l unfolding l xs zs by auto
      done
  next
    case xs-nempty-zs-empty note xs = this(1) and zs = this(2)
    have [simp]: ‹x ≠ x'› ‹x' ≠ x› ‹x' ∉ set xs'› ‹x ∉ set xs'›
      using dist unfolding l xs zs by auto
    have prev-next: ‹get-prev (ns ! x) = Some x'› ‹get-next (ns ! x) = None›
      using vmtf-ns vmtf-ns-append-decomp[of xs' x' x zs m ns] unfolding l xs zs
      by (auto simp: vmtf-ns-single-iff intro: vmtf-ns-last-mid-get-prev)
    then have vmtf': ‹vmtf-ns (xs' @ [x']) m (ns[x' := VMTF-Node (stamp (ns ! x')) (get-prev (ns !
x')) None])›
      using vmtf-ns unfolding r-l unfolding l xs zs
      by (auto simp: ns-vmtf-dequeue-def Let-def vmtf-ns-append-decomp split: option.splits
          intro: vmtf-ns.intros)
    show ?thesis
      apply (rule vmtf-ns-eq-iffI[of - -
            ‹(ns[x' := VMTF-Node (stamp (ns ! x')) (get-prev (ns ! x')) None])› m])
      subgoal
        using prev-next unfolding r-l unfolding l xs zs
        by (cases ‹ns ! x'›) (auto simp: ns-vmtf-dequeue-def Let-def nth-list-update')
      subgoal
        using prev-next le-length unfolding r-l unfolding l xs zs
        by (cases ‹ns ! x›) auto
      subgoal
        using vmtf' unfolding r-l unfolding l xs zs by auto
      done
  next
    case xs-zs-nempty note xs = this(1) and zs = this(2)
    have vmtf-ns-x'-x: ‹vmtf-ns (xs' @ [x', x] @ (y' #  zs')) m ns› and
      vmtf-ns-x-y: ‹vmtf-ns ((xs' @ [x']) @ [x, y'] @ zs') m ns›
      using vmtf-ns unfolding l xs zs by simp-all
    from vmtf-ns-append-decomp[OF vmtf-ns-x'-x] have
      vmtf-ns-xs: ‹vmtf-ns (xs' @ [x']) m (ns[x' := VMTF-Node (stamp (ns ! x')) (get-prev (ns ! x'))
```

None])› **and**
vmtf-ns-zs: ‹*vmtf-ns* (*x* # *y′* # *zs′*) (*stamp* (*ns* ! *x*)) (*ns*[*x* := *VMTF-Node* (*stamp* (*ns* ! *x*)) *None* (*get-next* (*ns* ! *x*))])› **and**
stamp: ‹*stamp* (*ns* ! *x*) < *stamp* (*ns* ! *x′*)›
by *fast+*
have [*simp*]: ‹*y′* < *length ns*› ‹*x* < *length ns*› ‹*x* ≠ *y′*› ‹*x′* ≠ *y′*› ‹*x′* < *length ns*› ‹*y′* ≠ *x′*›
‹*x′* ≠ *x*› ‹*x* ≠ *x′*› ‹*y′* ≠ *x*›
and *x-zs′*: ‹*x* ∉ *set zs′*› ‹*x* ∉ *set xs′*› **and** *x′-zs′*: ‹*x′* ∉ *set zs′*› **and** *y′-xs′*: ‹*y′* ∉ *set xs′*›
using *vmtf-ns-distinct*[*OF vmtf-ns*] *vmtf-ns-le-length*[*OF vmtf-ns*] **unfolding** *l xs zs*
by *auto*
obtain *n* **where**
vmtf-ns-zs′: ‹*vmtf-ns* (*y′* # *zs′*) *n* (*ns*[*x* := *VMTF-Node* (*stamp* (*ns* ! *x*)) *None* (*get-next* (*ns* ! *x*)),
y′ := *VMTF-Node* (*stamp* (*ns*[*x* := *VMTF-Node* (*stamp* (*ns* ! *x*)) *None* (*get-next* (*ns* ! *x*))] ! *y′*)) *None*
(*get-next* (*ns*[*x* := *VMTF-Node* (*stamp* (*ns* ! *x*)) *None* (*get-next* (*ns* ! *x*))] ! *y′*))])› **and**
‹*n* ≤ *stamp* (*ns* ! *x*)›
using *vmtf-ns-skip-fst*[*OF vmtf-ns-zs*] **by** *blast*
then have *vmtf-ns-y′-zs′-x-y′*: ‹*vmtf-ns* (*y′* # *zs′*) *n* (*ns*[*x* := *VMTF-Node* (*stamp* (*ns* ! *x*)) *None* (*get-next* (*ns* ! *x*)),
y′ := *VMTF-Node* (*stamp* (*ns* ! *y′*)) *None* (*get-next* (*ns* ! *y′*))])›
by *auto*

define *ns′* **where** ‹*ns′* = *ns*[*x′* := *VMTF-Node* (*stamp* (*ns* ! *x′*)) (*get-prev* (*ns* ! *x′*)) *None*,
y′ := *VMTF-Node* (*stamp* (*ns* ! *y′*)) *None* (*get-next* (*ns* ! *y′*))]›
have *vmtf-ns-y′-zs′-y′*: ‹*vmtf-ns* (*y′* # *zs′*) *n* (*ns*[*y′* := *VMTF-Node* (*stamp* (*ns* ! *y′*)) *None* (*get-next* (*ns* ! *y′*))])›
apply (*rule vmtf-ns-eq-iffI*[*OF - - vmtf-ns-y′-zs′-x-y′*])
subgoal using *x-zs′* **by** *auto*
subgoal using *vmtf-ns-le-length*[*OF vmtf-ns*] **unfolding** *l xs zs* **by** *auto*
done
moreover have *stamp-y′-n*: ‹*stamp* (*ns*[*x′* := *VMTF-Node* (*stamp* (*ns* ! *x′*)) (*get-prev* (*ns* ! *x′*)) *None*] ! *y′*) ≤ *n*›
using *vmtf-ns-stamp-sorted*[*OF vmtf-ns-y′-zs′-y′*] *stamp* **unfolding** *l xs zs*
by (*auto simp*: *sorted-append*)
ultimately have *vmtf-ns-y′-zs′-y′*: ‹*vmtf-ns* (*y′* # *zs′*) (*stamp* (*ns′* ! *y′*))
(*ns*[*y′* := *VMTF-Node* (*stamp* (*ns* ! *y′*)) *None* (*get-next* (*ns* ! *y′*))])›
using *l vmtf-ns vmtf-ns-append-decomp xs-zs-nempty*(2) *ns′-def* **by** *auto*
have *vmtf-ns-y′-zs′-x′-y′*: ‹*vmtf-ns* (*y′* # *zs′*) (*stamp* (*ns′* ! *y′*)) *ns′*›
apply (*rule vmtf-ns-eq-iffI*[*OF - - vmtf-ns-y′-zs′-y′*])
subgoal using *dist le-length x′-zs′ ns′-def* **unfolding** *l xs zs* **by** *auto*
subgoal using *dist le-length x′-zs′ ns′-def* **unfolding** *l xs zs* **by** *auto*
done
have *vmtf-ns-xs′*: ‹*vmtf-ns* (*xs′* @ [*x′*]) *m ns′*›
apply (*rule vmtf-ns-eq-iffI*[*OF - - vmtf-ns-xs*])
subgoal using *y′-xs′ ns′-def* **by** *auto*
subgoal using *vmtf-ns-le-length*[*OF vmtf-ns-xs*] *ns′-def* **by** *auto*
done
have *vmtf-x′-y′*: ‹*vmtf-ns* (*xs′* @ [*x′*, *y′*] @ *zs′*) *m*
(*ns′*[*x′* := *VMTF-Node* (*stamp* (*ns′* ! *x′*)) (*get-prev* (*ns′* ! *x′*)) (*Some y′*),
y′ := *VMTF-Node* (*stamp* (*ns′* ! *y′*)) (*Some x′*) (*get-next* (*ns′* ! *y′*))])›
apply (*rule vmtf-ns-append-rebuild*[*OF vmtf-ns-xs′ vmtf-ns-y′-zs′-x′-y′*])
subgoal using *stamp-y′-n vmtf-ns-xs vmtf-ns-zs stamp* ‹*n* ≤ *stamp* (*ns* ! *x*)›
unfolding *ns′-def* **by** *auto*
subgoal by (*metis append.assoc append-Cons distinct-remove1 r-l self-append-conv2 vmtf-ns*
vmtf-ns-distinct xs zs)
done

```
    have ‹vmtf-ns (xs' @ [x', y'] @ zs') m
      (ns'[x' := VMTF-Node (stamp (ns' ! x')) (get-prev (ns' ! x')) (Some y'),
        y' := VMTF-Node (stamp (ns' ! y')) (Some x') (get-next (ns' ! y')),
        x := VMTF-Node (stamp (ns' ! x)) None None])›
      apply (rule vmtf-ns-eq-iffI[OF - - vmtf-x'-y'])
      subgoal
       using vmtf-ns-last-mid-get-next[OF vmtf-ns-x-y] vmtf-ns-last-mid-get-prev[OF vmtf-ns-x'-x] x-zs'
       by (cases ‹ns!x›; auto simp: nth-list-update' ns'-def)
      subgoal using le-length unfolding l xs zs ns'-def by auto
      done
    moreover have ‹xs' @ [x', y'] @ zs' = remove1 x l›
      unfolding r-l xs zs by auto
    moreover have ‹ns'[x' := VMTF-Node (stamp (ns' ! x')) (get-prev (ns' ! x')) (Some y'),
        y' := VMTF-Node (stamp (ns' ! y')) (Some x') (get-next (ns' ! y')),
        x := VMTF-Node (stamp (ns' ! x)) None None] = ns-vmtf-dequeue x ns›
      using vmtf-ns-last-mid-get-next[OF vmtf-ns-x-y] vmtf-ns-last-mid-get-prev[OF vmtf-ns-x'-x]
      list-update-swap[of x' y' - ‹- :: nat-vmtf-node›]
      unfolding ns'-def ns-vmtf-dequeue-def
      by (auto simp: Let-def)
    ultimately show ?thesis
      by force
  qed
qed

lemma vmtf-ns-hd-next:
   ‹vmtf-ns (x # a # list) m ns ⟹ get-next (ns ! x) = Some a›
  by (auto 5 5 elim: vmtf-nsE)

lemma vmtf-ns-notin-dequeue:
  assumes vmtf-ns: ‹vmtf-ns l m ns› and notin: ‹vmtf-ns-notin l m ns› and valid: ‹x < length ns›
  shows ‹vmtf-ns-notin (remove1 x l) m (ns-vmtf-dequeue x ns)›
proof (cases ‹x ∈ set l›)
  case False
  then have H: ‹remove1 x l = l›
    by (simp add: remove1-idem)
  have simp-is-stupid[simp]: ‹a ∈ set l ⟹ x ∉ set l ⟹ a ≠ x› ‹a ∈ set l ⟹ x ∉ set l ⟹ x ≠ a›
for a x
    by auto
  have
    ‹get-prev (ns ! x) = None› and
    ‹get-next (ns ! x) = None›
    using notin False valid unfolding vmtf-ns-notin-def by auto
  show ?thesis
    using notin valid False unfolding vmtf-ns-notin-def
    by (auto simp: vmtf-ns-notin-def ns-vmtf-dequeue-def Let-def H split: option.splits)
next
  case True
  then obtain xs zs where
    l: ‹l = xs @ x # zs›
    by (meson split-list)
  have r-l: ‹remove1 x l = xs @ zs›
    using vmtf-ns-distinct[OF vmtf-ns] unfolding l by (simp add: remove1-append)

  consider
    (xs-zs-empty) ‹xs = []› and ‹zs = []› |
    (xs-nempty-zs-empty) x' xs' where ‹xs = xs' @ [x']› and ‹zs = []› |
```

```
  (xs-empty-zs-nempty) y' zs' where ‹xs = []› and ‹zs = y' # zs'› |
  (xs-zs-nempty) x' y' xs' zs' where ‹xs = xs' @ [x']› and ‹zs = y' # zs'›
  by (cases xs rule: rev-cases; cases zs)
then show ?thesis
proof cases
  case xs-zs-empty
  then show ?thesis
    using notin vmtf-ns unfolding l apply (cases ‹ns ! x›)
      by (auto simp: vmtf-ns-notin-def ns-vmtf-dequeue-def Let-def vmtf-ns-single-iff
        split: option.splits)
next
  case xs-empty-zs-nempty note xs = this(1) and zs = this(1)
  have prev-next: ‹get-prev (ns ! x) = None› ‹get-next (ns ! x) = option-hd zs›
    using vmtf-ns unfolding l xs zs
    by (cases zs; auto simp: option-hd-def elim: vmtf-nsE dest: vmtf-ns-hd-next)+
  show ?thesis
    apply (rule vmtf-ns-notinI)
    apply (case-tac ‹i = x›)
    subgoal
      using vmtf-ns prev-next unfolding r-l unfolding l xs zs
      by (cases zs) (auto simp: ns-vmtf-dequeue-def Let-def
          vmtf-ns-notin-def vmtf-ns-single-iff
          split: option.splits)
    subgoal
      using vmtf-ns notin prev-next unfolding r-l unfolding l xs zs
      by (auto simp: ns-vmtf-dequeue-def Let-def
          vmtf-ns-notin-def vmtf-ns-single-iff
          split: option.splits
          intro: vmtf-ns.intros vmtf-ns-stamp-increase dest: vmtf-ns-skip-fst)
     done
next
  case xs-nempty-zs-empty note xs = this(1) and zs = this(2)
  have prev-next: ‹get-prev (ns ! x) = Some x'› ‹get-next (ns ! x) = None›
    using vmtf-ns vmtf-ns-append-decomp[of xs' x' x zs m ns] unfolding l xs zs
    by (auto simp: vmtf-ns-single-iff intro: vmtf-ns-last-mid-get-prev)
  then show ?thesis
    using vmtf-ns notin unfolding r-l unfolding l xs zs
    by (auto simp: ns-vmtf-dequeue-def Let-def vmtf-ns-append-decomp vmtf-ns-notin-def
        split: option.splits
        intro: vmtf-ns.intros)
next
  case xs-zs-nempty note xs = this(1) and zs = this(2)
  have vmtf-ns-x'-x: ‹vmtf-ns (xs' @ [x', x] @ (y' #  zs')) m ns› and
    vmtf-ns-x-y: ‹vmtf-ns ((xs' @ [x']) @ [x, y'] @ zs') m ns›
    using vmtf-ns unfolding l xs zs by simp-all
  have [simp]: ‹y' < length ns› ‹x < length ns› ‹x ≠ y'› ‹x' ≠ y'› ‹x' < length ns› ‹y' ≠ x'›
    ‹y' ≠ x› ‹y' ∉ set xs›  ‹y' ∉ set zs'›
    and x-zs': ‹x ∉ set zs'› and x'-zs': ‹x' ∉ set zs'› and y'-xs': ‹y' ∉ set xs'›
    using vmtf-ns-distinct[OF vmtf-ns] vmtf-ns-le-length[OF vmtf-ns] unfolding l xs zs
    by auto
  have ‹get-next (ns!x) = Some y'› ‹get-prev (ns!x) = Some x'›
    using vmtf-ns-last-mid-get-prev[OF vmtf-ns-x'-x] vmtf-ns-last-mid-get-next[OF vmtf-ns-x-y]
    by fast+
  then show ?thesis
    using notin x-zs' x'-zs' y'-xs' unfolding l xs zs
    by (auto simp: vmtf-ns-notin-def ns-vmtf-dequeue-def)
```

```
  qed
qed

lemma vmtf-ns-stamp-distinct:
  assumes ‹vmtf-ns l m ns›
  shows ‹distinct (map (λa. stamp (ns!a)) l)›
  using assms
proof (induction rule: vmtf-ns.induct)
  case (Cons b l m xs a n xs' n') note vmtf-ns = this(1) and IH = this(9) and a-le-y = this(2) and
    zs-a = this(3) and ab = this(4) and a-l = this(5) and mn = this(6) and xs' = this(7) and
    nn' = this(8)
  have [simp]: ‹map (λaa. stamp
              (if b = aa
               then VMTF-Node (stamp (xs ! aa)) (Some a) (get-next (xs ! aa))
               else xs ! aa)) l =
      map (λaa. stamp (xs ! aa)) l
     › for a
    apply (rule map-cong)
    subgoal ..
    subgoal using vmtf-ns-distinct[OF vmtf-ns] by auto
    done
  show ?case
    using Cons vmtf-ns-distinct[OF vmtf-ns] vmtf-ns-le-length[OF vmtf-ns]
    by (auto simp: sorted-many-eq-append leD vmtf-ns-stamp-sorted cong: if-cong)
qed auto

lemma vmtf-ns-thighten-stamp:
  assumes vmtf-ns: ‹vmtf-ns l m xs› and n: ‹∀ a∈set l. stamp (xs ! a) ≤ n›
  shows ‹vmtf-ns l n xs›
proof −
  consider
    (empty) ‹l = []› |
    (single) x where ‹l = [x]› |
    (more-than-two) x y ys where ‹l = x # y # ys›
    by (cases l; cases ‹tl l›) auto
  then show ?thesis
  proof cases
    case empty
    then show ?thesis by (auto intro: vmtf-ns.intros)
  next
    case (single x)
    then show ?thesis using n vmtf-ns by (auto simp: vmtf-ns-single-iff)
  next
    case (more-than-two x y ys) note l = this
    then have vmtf-ns': ‹vmtf-ns ([] @ [x, y] @ ys) m xs›
      using vmtf-ns by auto
    from vmtf-ns-append-decomp[OF this] have
      ‹vmtf-ns ([x]) m (xs[x := VMTF-Node (stamp (xs ! x)) (get-prev (xs ! x)) None])› and
      vmtf-ns-y-ys: ‹vmtf-ns (y # ys) (stamp (xs ! y))
        (xs[y := VMTF-Node (stamp (xs ! y)) None (get-next (xs ! y))])› and
      ‹stamp (xs ! y) < stamp (xs ! x)›
      by auto
    have [simp]: ‹x ≠ y› ‹x ∉ set ys› ‹x < length xs› ‹y < length xs›
      using vmtf-ns-distinct[OF vmtf-ns] vmtf-ns-le-length[OF vmtf-ns] unfolding l by auto
    show ?thesis
      unfolding l
```

```
    apply (rule vmtf-ns.Cons[OF vmtf-ns-y-ys, of - ‹stamp (xs ! x)›])
    subgoal using vmtf-ns-le-length[OF vmtf-ns] unfolding l by auto
    subgoal using vmtf-ns unfolding l by (cases ‹xs ! x›) (auto elim: vmtf-nsE)
    subgoal by simp
    subgoal by simp
    subgoal using vmtf-ns-stamp-sorted[OF vmtf-ns] vmtf-ns-stamp-distinct[OF vmtf-ns]
     by (auto simp: l sorted-many-eq-append)
    subgoal
      using vmtf-ns vmtf-ns-last-mid-get-prev[OF vmtf-ns']
      apply (cases ‹xs ! y›)
      by simp (auto simp: l eq-commute[of ‹xs ! y›])
    subgoal using n unfolding l by auto
    done
  qed
qed

lemma vmtf-ns-rescale:
  assumes
    ‹vmtf-ns l m xs› and
    ‹sorted (map (λa. st ! a) (rev l))› and ‹distinct (map (λa. st ! a) l)›
    ‹∀ a ∈ set l. get-prev (zs ! a) = get-prev (xs ! a)› and
    ‹∀ a ∈ set l. get-next (zs ! a) = get-next (xs ! a)› and
    ‹∀ a ∈ set l. stamp (zs ! a) = st ! a› and
    ‹length xs ≤ length zs› and
    ‹∀ a∈set l. a < length st› and
    m': ‹∀ a ∈ set l. st ! a < m'›
  shows ‹vmtf-ns l m' zs›
  using assms
proof (induction arbitrary: zs m' rule: vmtf-ns.induct)
  case (Nil st xs)
  then show ?case by (auto intro: vmtf-ns.intros)
next
  case (Cons1 a xs m n)
  then show ?case by (cases ‹zs ! a›) (auto simp: vmtf-ns-single-iff intro!: Max-ge nth-mem)
next
  case (Cons b l m xs a n xs' n' zs m') note vmtf-ns = this(1) and a-le-y = this(2) and
    zs-a = this(3) and ab = this(4) and a-l = this(5) and mn = this(6) and xs' = this(7) and
    nn' = this(8) and IH = this(9) and H = this(10−)
  have [simp]: ‹b < length xs› ‹b ≠ a› ‹a ≠ b› ‹b ∉ set l› ‹b < length zs› ‹a < length zs›
    using vmtf-ns-distinct[OF vmtf-ns] vmtf-ns-le-length[OF vmtf-ns] ab H(6) a-le-y unfolding xs'
    by force+

  have simp-is-stupid[simp]: ‹a ∈ set l ⟹ x ∉ set l ⟹ a ≠ x› ‹a ∈ set l ⟹ x ∉ set l ⟹ x ≠ a›
for a x
    by auto
  define zs' where ‹zs' ≡ (zs[b := VMTF-Node (st ! b) (get-prev (xs ! b)) (get-next (xs ! b)),
        a := VMTF-Node (st ! a) None (Some b)])›
  have zs-upd-zs: ‹zs = zs
    [b := VMTF-Node (st ! b) (get-prev (xs ! b)) (get-next (xs ! b)),
     a := VMTF-Node (st ! a) None (Some b),
     b := VMTF-Node (st ! b) (Some a) (get-next (xs ! b))]
    ›
    using H(2−5) xs' zs-a ‹b < length xs›
    by (metis list.set-intros(1) list.set-intros(2) list-update-id list-update-overwrite
      nth-list-update-eq nth-list-update-neq vmtf-node.collapse vmtf-node.sel(2,3))
```

```
  have vtmf-b-l: ‹vmtf-ns (b # l) m' zs'›
    unfolding zs'-def
    apply (rule IH)
    subgoal using H(1) by (simp add: sorted-many-eq-append)
    subgoal using H(2) by auto
    subgoal using H(3,4,5) xs' zs-a a-l ab by (auto split: if-splits)
    subgoal using H(4) xs' zs-a a-l ab by auto
    subgoal using H(5) xs' a-l ab by auto
    subgoal using H(6) xs' by auto
    subgoal using H(7) xs' by auto
    subgoal using H(8) by auto
    done
  then have ‹vmtf-ns (b # l) (stamp (zs' ! b)) zs'›
    by (rule vmtf-ns-thighten-stamp)
      (use vmtf-ns-stamp-sorted[OF vtmf-b-l] in ‹auto simp: sorted-append›)

  then show ?case
    apply (rule vmtf-ns.Cons[of - - - - - ‹st ! a›])
    unfolding zs'-def
    subgoal using a-le-y H(6) xs' by auto
    subgoal using a-le-y by auto
    subgoal using ab.
    subgoal using a-l .
    subgoal using nn' mn H(1,2) by (auto simp: sorted-many-eq-append)
    subgoal using zs-upd-zs by auto
    subgoal using H by (auto intro!: Max-ge nth-mem)
    done
qed

lemma vmtf-ns-last-prev:
  assumes vmtf: ‹vmtf-ns (xs @ [x]) m ns›
  shows ‹get-prev (ns ! x) = option-last xs›
proof (cases xs rule: rev-cases)
  case Nil
  then show ?thesis using vmtf by (cases ‹ns!x›) (auto simp: vmtf-ns-single-iff)
next
  case (snoc xs' y')
  then show ?thesis
    using vmtf-ns-last-mid-get-prev[of xs' y' x ‹[]› m ns] vmtf by auto
qed
```

Abstract Invariants Invariants

- The atoms of *xs* and *ys* are always disjoint.
- The atoms of *ys* are *always* set.
- The atoms of *xs* *can* be set but do not have to.

```
definition vmtf-\mathcal{L}_{all} :: ‹nat multiset ⇒ (nat, nat) ann-lits ⇒ nat abs-vmtf-ns ⇒ bool› where
‹vmtf-\mathcal{L}_{all} \mathcal{A} M ≡ λ(xs, ys).
  (∀ L∈ys. L ∈ atm-of ‘ lits-of-l M) ∧
  xs ∩ ys = {} ∧
  xs ∪ ys = atms-of (\mathcal{L}_{all} \mathcal{A})
  ›
```

abbreviation *abs-vmtf-ns-inv* :: ‹*nat multiset* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *nat abs-vmtf-ns* ⇒ *bool*› **where**
‹*abs-vmtf-ns-inv* $\mathcal{A}$ *M vm* ≡ *vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M vm*›

Implementation

type-synonym (**in** −) *vmtf* = ‹*nat-vmtf-node list* × *nat* × *nat* × *nat* × *nat option*›

We use the opposite direction of the VMTF paper: The latest added element *fst-As* is at the beginning.

definition *vmtf* :: ‹*nat multiset* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *vmtf set*› **where**
‹*vmtf* $\mathcal{A}$ *M* = {(*ns*, *m*, *fst-As*, *lst-As*, *next-search*).
  (∃ *xs′ ys′*.
    vmtf-ns (*ys′* @ *xs′*) *m ns* ∧ *fst-As* = *hd* (*ys′* @ *xs′*) ∧ *lst-As* = *last* (*ys′* @ *xs′*)
  ∧ *next-search* = *option-hd xs′*
  ∧ *vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* (*set xs′*, *set ys′*)
  ∧ *vmtf-ns-notin* (*ys′* @ *xs′*) *m ns*
  ∧ (∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*) ∧ (∀ *L*∈*set* (*ys′* @ *xs′*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$))
)}›

lemma *vmtf-consD*:
 assumes *vmtf*: ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*)) ∈ *vmtf* $\mathcal{A}$ *M*›
 shows ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*)) ∈ *vmtf* $\mathcal{A}$ (*L* # *M*)›
proof −
 obtain *xs′ ys′* **where**
  vmtf-ns: ‹*vmtf-ns* (*ys′* @ *xs′*) *m ns*› **and**
  fst-As: ‹*fst-As* = *hd* (*ys′* @ *xs′*)› **and**
  lst-As: ‹*lst-As* = *last* (*ys′* @ *xs′*)› **and**
  next-search: ‹*next-search* = *option-hd xs′*› **and**
  abs-vmtf: ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs′*, *set ys′*))› **and**
  notin: ‹*vmtf-ns-notin* (*ys′* @ *xs′*) *m ns*› **and**
  atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*› **and**
  ‹∀ *L*∈*set* (*ys′* @ *xs′*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
  using *vmtf* **unfolding** *vmtf-def* **by** *fast*
 moreover have ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ (*L* # *M*) ((*set xs′*, *set ys′*))›
  using *abs-vmtf* **unfolding** *vmtf-*$\mathcal{L}_{all}$*-def* **by** *auto*
 ultimately have ‹*vmtf-ns* (*ys′* @ *xs′*) *m ns* ∧
    fst-As = *hd* (*ys′* @ *xs′*) ∧
    lst-As = *last* (*ys′* @ *xs′*) ∧
    next-search = *option-hd xs′* ∧
    vmtf-$\mathcal{L}_{all}$ $\mathcal{A}$ (*L* # *M*) ((*set xs′*, *set ys′*)) ∧
    vmtf-ns-notin (*ys′* @ *xs′*) *m ns* ∧ (∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*) ∧
    (∀ *L*∈*set* (*ys′* @ *xs′*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
   by *fast*
 then show *?thesis*
  unfolding *vmtf-def* **by** *fast*
qed

lemma *vmtf-consD′*:
 assumes *vmtf*: ‹*x* ∈ *vmtf* $\mathcal{A}$ *M*›
 shows ‹*x* ∈ *vmtf* $\mathcal{A}$ (*L* # *M*)›
 using *vmtf-consD*
 by (*metis prod-cases5 vmtf*)

type-synonym (**in** −) *vmtf-option-fst-As* = ‹*nat-vmtf-node list* × *nat* × *nat option* × *nat option* × *nat option*›

definition (**in** −) *vmtf-dequeue* :: ‹*nat* ⇒ *vmtf* ⇒ *vmtf-option-fst-As*› **where**
‹*vmtf-dequeue* ≡ (λ*L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*).
(*let fst-As′* = (*if fst-As* = *L then get-next* (*ns* ! *L*) *else Some fst-As*);
next-search′ = *if next-search* = *Some L then get-next* (*ns* ! *L*) *else next-search*;
lst-As′ = *if lst-As* = *L then get-prev* (*ns* ! *L*) *else Some lst-As in*
(*ns-vmtf-dequeue L ns*, *m*, *fst-As′*, *lst-As′*, *next-search′*)))›

It would be better to distinguish whether L is set in M or not.

definition *vmtf-enqueue* :: ‹(*nat*, *nat*) *ann-lits* ⇒ *nat* ⇒ *vmtf-option-fst-As* ⇒ *vmtf*› **where**
‹*vmtf-enqueue* = (λ*M L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*).
(*case fst-As of*
None ⇒ (*ns*[*L* := *VMTF-Node m fst-As None*], *m*+*1*, *L*, *L*,
(*if defined-lit M* (*Pos L*) *then None else Some L*))
| *Some fst-As* ⇒
let fst-As′ = *VMTF-Node* (*stamp* (*ns*!*fst-As*)) (*Some L*) (*get-next* (*ns*!*fst-As*)) *in*
(*ns*[*L* := *VMTF-Node* (*m*+*1*) *None* (*Some fst-As*), *fst-As* := *fst-As′*],
m+*1*, *L*, *the lst-As*, (*if defined-lit M* (*Pos L*) *then next-search else Some L*))))›

definition (**in** −) *vmtf-en-dequeue* :: ‹(*nat*, *nat*) *ann-lits* ⇒ *nat* ⇒ *vmtf* ⇒ *vmtf*› **where**
‹*vmtf-en-dequeue* = (λ*M L vm*. *vmtf-enqueue M L* (*vmtf-dequeue L vm*))›

lemma *abs-vmtf-ns-bump-vmtf-dequeue*:
fixes *M*
assumes *vmtf*:‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*)) ∈ *vmtf* 𝒜 *M*› **and**
L: ‹*L* ∈ *atms-of* (ℒ*all* 𝒜)› **and**
dequeue: ‹(*ns′*, *m′*, *fst-As′*, *lst-As′*, *next-search′*) =
vmtf-dequeue L (*ns*, *m*, *fst-As*, *lst-As*, *next-search*)› **and**
𝒜*in*-*nempty*: ‹*isasat-input-nempty* 𝒜›
shows ‹∃ *xs′ ys′*. *vmtf-ns* (*ys′* @ *xs′*) *m′ ns′* ∧ *fst-As′* = *option-hd* (*ys′* @ *xs′*)
∧ *lst-As′* = *option-last* (*ys′* @ *xs′*)
∧ *next-search′* = *option-hd xs′*
∧ *next-search′* = (*if next-search* = *Some L then get-next* (*ns*!*L*) *else next-search*)
∧ *vmtf-*ℒ*all* 𝒜 *M* ((*insert L* (*set xs′*), *set ys′*))
∧ *vmtf-ns-notin* (*ys′* @ *xs′*) *m′ ns′* ∧
L ∉ *set* (*ys′* @ *xs′*) ∧ (∀ *L*∈*set* (*ys′* @ *xs′*). *L* ∈ *atms-of* (ℒ*all* 𝒜))›
unfolding *vmtf-def*
proof −
have *ns′*: ‹*ns′* = *ns-vmtf-dequeue L ns*› **and**
fst-As′: ‹*fst-As′* = (*if fst-As* = *L then get-next* (*ns* ! *L*) *else Some fst-As*)› **and**
lst-As′: ‹*lst-As′* = (*if lst-As* = *L then get-prev* (*ns* ! *L*) *else Some lst-As*)› **and**
m′m: ‹*m′* = *m*› **and**
next-search-L-next:
‹*next-search′* = (*if next-search* = *Some L then get-next* (*ns*!*L*) *else next-search*)›
using *dequeue* **unfolding** *vmtf-dequeue-def* **by** *auto*
obtain *xs ys* **where**
vmtf: ‹*vmtf-ns* (*ys* @ *xs*) *m ns*› **and**
notin: ‹*vmtf-ns-notin* (*ys* @ *xs*) *m ns*› **and**
next-search: ‹*next-search* = *option-hd xs*› **and**
abs-inv: ‹*vmtf-*ℒ*all* 𝒜 *M* ((*set xs*, *set ys*))› **and**
fst-As: ‹*fst-As* = *hd* (*ys* @ *xs*)› **and**
lst-As: ‹*lst-As* = *last* (*ys* @ *xs*)› **and**
atm-A: ‹∀ *L*∈*atms-of* (ℒ*all* 𝒜). *L* < *length ns*› **and**
L-ys-xs: ‹∀ *L*∈*set* (*ys* @ *xs*). *L* ∈ *atms-of* (ℒ*all* 𝒜)›

```
  using vmtf unfolding vmtf-def by auto
have [dest]: ‹xs = [] ⟹ ys = [] ⟹ False›
  using abs-inv 𝒜_in-nempty unfolding atms-of-ℒ_all-𝒜_in vmtf-ℒ_all-def
  by auto
let ?ys = ‹ys›
let ?xs = ‹xs›
have dist: ‹distinct (xs @ ys)›
  using vmtf-ns-distinct[OF vmtf] by auto
have xs-ys: ‹remove1 L ys @ remove1 L xs = remove1 L (ys @ xs)›
  using dist by (auto simp: remove1-append remove1-idem disjoint-iff-not-equal
     intro!: remove1-idem)
have atm-L-A: ‹L < length ns›
  using atm-A L by blast

have ‹vmtf-ns (remove1 L ys @ remove1 L xs) m' ns'›
  using vmtf-ns-ns-vmtf-dequeue[OF vmtf notin, of L] dequeue dist atm-L-A
  unfolding vmtf-dequeue-def by (auto split: if-splits simp: xs-ys)
moreover have next-search': ‹next-search' = option-hd (remove1 L xs)›
proof −
  have ‹[hd xs, hd (tl xs)] @ tl (tl xs) = xs›
    if ‹xs ≠ []› ‹tl xs ≠ []›
    apply (cases xs; cases ‹tl xs›)
     using that by (auto simp: tl-append split: list.splits)
  then have [simp]: ‹get-next (ns ! hd xs) = option-hd (remove1 (hd xs) xs)› if ‹xs ≠ []›
    using vmtf-ns-last-mid-get-next[of ‹?ys› ‹hd ?xs›
       ‹hd (tl ?xs)› ‹tl (tl ?xs)› m ns] vmtf vmtf-ns-distinct[OF vmtf] that
      distinct-remove1-last-butlast[of xs]
    by (cases xs; cases ‹tl xs›)
      (auto simp: tl-append vmtf-ns-last-next split: list.splits elim: vmtf-nsE)
  have ‹xs ≠ [] ⟹ xs ≠ [L] ⟹  L ≠ hd xs ⟹ hd xs = hd (remove1 L xs)›
    by (induction xs) (auto simp: remove1-Nil-iff)
  then have [simp]: ‹option-hd xs = option-hd (remove1 L xs)› if ‹L ≠ hd xs›
    using that vmtf-ns-distinct[OF vmtf]
    by (auto simp: option-hd-def remove1-Nil-iff)
  show ?thesis
    using dequeue dist atm-L-A next-search next-search unfolding vmtf-dequeue-def
    by (auto split: if-splits simp: xs-ys dest: last-in-set)
  qed
moreover {
  have ‹[hd ys, hd (tl ys)] @ tl (tl ys) = ys›
    if ‹ys ≠ []› ‹tl ys ≠ []›
     using that by (auto simp: tl-append split: list.splits)
  then have ‹get-next (ns ! hd (ys @ xs)) = option-hd (remove1 (hd (ys @ xs)) (ys @ xs))›
    if ‹ys @ xs ≠ []›
    using vmtf-ns-last-next[of ‹?xs @ butlast ?ys› ‹last ?ys›] that
    using vmtf-ns-last-next[of ‹butlast ?xs› ‹last ?xs›]  vmtf dist
      distinct-remove1-last-butlast[of ‹?ys @ ?xs›]
    by (cases ys; cases ‹tl ys›)
     (auto simp: tl-append vmtf-ns-last-prev remove1-append hd-append remove1-Nil-iff
        split: list.splits if-splits elim: vmtf-nsE)
  moreover have ‹hd ys ∉ set xs› if ‹ys ≠ []›
    using vmtf-ns-distinct[OF vmtf] that by (cases ys) auto
  ultimately have ‹fst-As' = option-hd (remove1 L (ys @ xs))›
    using dequeue dist atm-L-A fst-As vmtf-ns-distinct[OF vmtf] vmtf
    unfolding vmtf-dequeue-def
    apply (cases ys)
```

```
    subgoal by (cases xs) (auto simp: remove1-append option-hd-def remove1-Nil-iff split: if-splits)
    subgoal by (auto simp: remove1-append option-hd-def remove1-Nil-iff)
    done
  }
  moreover have ‹lst-As' = option-last (remove1 L (ys @ xs))›
    apply (cases ‹ys @ xs› rule: rev-cases)
    using lst-As vmtf-ns-distinct[OF vmtf] vmtf-ns-last-prev vmtf
    by (auto simp: lst-As' remove1-append simp del: distinct-append) auto
  moreover have ‹vmtf-\mathcal{L}_{all} \mathcal{A} M (insert L (set (remove1 L xs)), set (remove1 L ys))›
    using abs-inv L dist
    unfolding vmtf-\mathcal{L}_{all}-def by (auto dest: in-set-remove1D)
  moreover have ‹vmtf-ns-notin (remove1 L ?ys @ remove1 L ?xs) m' ns'›
    unfolding xs-ys ns'
    apply (rule vmtf-ns-notin-dequeue)
    subgoal using vmtf unfolding m'm .
    subgoal using notin unfolding m'm .
    subgoal using atm-L-A .
    done
  moreover have ‹∀ L∈atms-of (\mathcal{L}_{all} \mathcal{A}). L < length ns'›
    using atm-A unfolding ns' by auto
  moreover have ‹L ∉ set (remove1 L ys @ remove1 L xs)›
    using dist by auto
  moreover have ‹∀ L∈set (remove1 L (ys @ xs)). L ∈ atms-of (\mathcal{L}_{all} \mathcal{A})›
    using L-ys-xs by (auto dest: in-set-remove1D)
  ultimately show ?thesis
    using next-search-L-next
    apply −
    apply (rule exI[of - ‹remove1 L xs›])
    apply (rule exI[of - ‹remove1 L ys›])
    unfolding xs-ys
    by blast
qed

lemma vmtf-ns-get-prev-not-itself:
  ‹vmtf-ns xs m ns ⟹ L ∈ set xs ⟹ L < length ns ⟹ get-prev (ns ! L) ≠ Some L›
  apply (induction rule: vmtf-ns.induct)
  subgoal by auto
  subgoal by (auto simp: vmtf-ns-single-iff)
  subgoal by auto
  done

lemma vmtf-ns-get-next-not-itself:
  ‹vmtf-ns xs m ns ⟹ L ∈ set xs ⟹ L < length ns ⟹ get-next (ns ! L) ≠ Some L›
  apply (induction rule: vmtf-ns.induct)
  subgoal by auto
  subgoal by (auto simp: vmtf-ns-single-iff)
  subgoal by auto
  done

lemma abs-vmtf-ns-bump-vmtf-en-dequeue:
  fixes M
  assumes
    vmtf: ‹(ns, m, fst-As, lst-As, next-search) ∈ vmtf \mathcal{A} M› and
    L: ‹L ∈ atms-of (\mathcal{L}_{all} \mathcal{A})› and
    nempty: ‹isasat-input-nempty \mathcal{A}›
  shows ‹vmtf-en-dequeue M L (ns, m, fst-As, lst-As, next-search) ∈ vmtf \mathcal{A} M›
```

unfolding *vmtf-def*
proof *clarify*
fix *xxs yys zzs ns' m' fst-As' lst-As' next-search'*
assume *dequeue*: ‹*vmtf-en-dequeue M L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*) = (*ns'*, *m'*, *fst-As'*, *lst-As'*, *next-search'*)›
obtain *xs ys* **where**
vmtf-ns: ‹*vmtf-ns* (*ys* @ *xs*) *m ns*› **and**
notin: ‹*vmtf-ns-notin* (*ys* @ *xs*) *m ns*› **and**
next-search: ‹*next-search* = *option-hd xs*› **and**
abs-inv: ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* (*set xs*, *set ys*)› **and**
fst-As: ‹*fst-As* = *hd* (*ys* @ *xs*)› **and**
lst-As: ‹*lst-As* = *last* (*ys* @ *xs*)› **and**
atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*› **and**
ys-xs-$\mathcal{L}_{all}$: ‹∀ *L*∈*set* (*ys* @ *xs*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *assms* **unfolding** *vmtf-def* **by** *auto*
have *atm-L-A*: ‹*L* < *length ns*›
using *atm-A L* **by** *blast*

d stands for dequeue

obtain *nsd md fst-Asd lst-Asd next-searchd* **where**
de: ‹*vmtf-dequeue L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*) = (*nsd*, *md*, *fst-Asd*, *lst-Asd*, *next-searchd*) ›
by (*cases* ‹*vmtf-dequeue L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*)›)
obtain *xs' ys'* **where**
vmtf-ns': ‹*vmtf-ns* (*ys'* @ *xs'*) *md nsd*› **and**
fst-Asd: ‹*fst-Asd* = *option-hd* (*ys'* @ *xs'*)› **and**
lst-Asd: ‹*lst-Asd* = *option-last* (*ys'* @ *xs'*)› **and**
next-searchd-hd: ‹*next-searchd* = *option-hd xs'*› **and**
next-searchd-L-next:
‹*next-searchd* = (*if next-search* = *Some L then get-next* (*ns*!*L*) *else next-search*)› **and**
abs-l: ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* (*insert L* (*set xs'*), *set ys'*)› **and**
not-in: ‹*vmtf-ns-notin* (*ys'* @ *xs'*) *md nsd*› **and**
L-xs'-ys': ‹*L* ∉ *set* (*ys'* @ *xs'*)› **and**
L-xs'-ys'-$\mathcal{L}_{all}$: ‹∀ *L*∈*set* (*ys'* @ *xs'*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *abs-vmtf-ns-bump-vmtf-dequeue*[*OF vmtf L de*[*symmetric*] *nempty*] **by** *blast*

have [*simp*]: ‹*length ns'* = *length ns*› ‹*length nsd* = *length ns*›
using *dequeue de* **unfolding** *vmtf-en-dequeue-def comp-def vmtf-dequeue-def*
by (*auto simp add*: *vmtf-enqueue-def split*: *option.splits*)
have *nsd*: ‹*nsd* = *ns-vmtf-dequeue L ns*›
using *de* **unfolding** *vmtf-dequeue-def* **by** *auto*
have [*simp*]: ‹*fst-As* = *L*› **if** ‹*ys'* = []› **and** ‹*xs'* = []›
proof −
have *1*: ‹*set-mset* $\mathcal{A}$ = {*L*}›
using *abs-l* **unfolding** *that vmtf-*$\mathcal{L}_{all}$*-def* **by** (*auto simp*: *atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$)
show *?thesis*
using *vmtf-ns-distinct*[*OF vmtf-ns*] *ys-xs-*$\mathcal{L}_{all}$ *abs-inv*
unfolding *atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *1 fst-As vmtf-*$\mathcal{L}_{all}$*-def*
by (*cases* ‹*ys* @ *xs*›) *auto*
qed
have *fst-As'*: ‹*fst-As'* = *L*› **and** *m'*: ‹*m'* = *md* + *1*› **and**
lst-As': ‹*fst-Asd* ≠ *None* ⟶ *lst-As'* = *the* (*lst-Asd*)›
‹*fst-Asd* = *None* ⟶ *lst-As'* = *L*›
using *dequeue* **unfolding** *vmtf-en-dequeue-def comp-def de*
by (*auto simp add*: *vmtf-enqueue-def split*: *option.splits*)
have ‹*lst-As* = *L*› **if** ‹*ys'* = []› **and** ‹*xs'* = []›
proof −

```
    have 1: ‹set-mset 𝒜 = {L}›
      using abs-l unfolding that vmtf-ℒall-def by (auto simp: atms-of-ℒall-𝒜in)
    then have ‹set (ys @ xs) = {L} ›
      using vmtf-ns-distinct[OF vmtf-ns] ys-xs-ℒall abs-inv
      unfolding atms-of-ℒall-𝒜in 1 fst-As vmtf-ℒall-def
      by auto
    then have ‹ys @ xs = [L]›
      using vmtf-ns-distinct[OF vmtf-ns] ys-xs-ℒall abs-inv vmtf-ℒall-def
      unfolding atms-of-ℒall-𝒜in 1 fst-As
      by (cases ‹ys @ xs› rule: rev-cases) (auto simp del: set-append distinct-append
          simp: set-append[symmetric], auto)
    then show ?thesis
      using vmtf-ns-distinct[OF vmtf-ns] ys-xs-ℒall abs-inv vmtf-ℒall-def
      unfolding atms-of-ℒall-𝒜in 1 lst-As
      by (auto simp del: set-append distinct-append simp: set-append[symmetric])
  qed
  then have [simp]: ‹lst-As' = L› if ‹ys' = []› and ‹xs' = []›
    using lst-As' fst-Asd unfolding that by auto
  have [simp]: ‹lst-As' = last (ys' @ xs')› if ‹ys' ≠ [] ∨ xs' ≠ []›
    using lst-As' fst-Asd that lst-Asd by auto


  have ‹get-prev (nsd ! i) ≠ Some L› (is ?prev) and
    ‹get-next (nsd ! i) ≠ Some L› (is ?next)
    if
      i-le-A: ‹i < length ns› and
      i-L: ‹i ≠ L› and
      i-ys': ‹i ∉ set ys'› and
      i-xs': ‹i ∉ set xs'›
    for i
  proof −
    have ‹i ∉ set xs› ‹i ∉ set ys› and L-xs-ys: ‹L ∈ set xs ∨ L ∈ set ys›
      using i-ys' i-xs' abs-l abs-inv i-L unfolding vmtf-ℒall-def
      by auto
    then have
      ‹get-next (ns ! i) = None›
      ‹get-prev (ns ! i) = None›
      using notin i-le-A unfolding nsd vmtf-ns-notin-def ns-vmtf-dequeue-def
      by (auto simp: Let-def split: option.splits)
    moreover have ‹get-prev (ns ! L) ≠ Some L› and ‹get-next (ns ! L) ≠ Some L›
      using vmtf-ns-get-prev-not-itself[OF vmtf-ns, of L] L-xs-ys atm-L-A
        vmtf-ns-get-next-not-itself[OF vmtf-ns, of L] by auto
    ultimately show ?next and ?prev
      using i-le-A L-xs-ys unfolding nsd ns-vmtf-dequeue-def vmtf-ns-notin-def
      by (auto simp: Let-def split: option.splits)
  qed
  then have vmtf-ns-notin': ‹vmtf-ns-notin (L # ys' @ xs') m' ns'›
    using not-in dequeue fst-Asd unfolding vmtf-en-dequeue-def comp-def de vmtf-ns-notin-def
      ns-vmtf-dequeue-def
    by (auto simp add: vmtf-enqueue-def hd-append split: option.splits if-splits)

consider
  (defined) ‹defined-lit M (Pos L)› |
  (undef) ‹undefined-lit M (Pos L)›
  by blast
then show ‹∃ xs' ys'.
```

```
    vmtf-ns (ys' @ xs') m' ns' ∧
    fst-As' = hd (ys' @ xs') ∧
    lst-As' = last (ys' @ xs') ∧
    next-search' = option-hd xs' ∧
    vmtf-ℒall 𝒜 M ((set xs', set ys')) ∧
    vmtf-ns-notin (ys' @ xs') m' ns' ∧
    (∀ L∈atms-of (ℒall 𝒜). L < length ns') ∧
    (∀ L∈set (ys' @ xs'). L ∈ atms-of (ℒall 𝒜))›
proof cases
  case defined
  have L-in-M: ‹L ∈ atm-of ‘ lits-of-l M›
    using defined by (auto simp: defined-lit-map lits-of-def)
  have next-search': ‹fst-Asd ≠ None ⟶ next-search' = next-searchd›
    ‹fst-Asd = None ⟶ next-search' = None›
    using dequeue defined unfolding vmtf-en-dequeue-def comp-def de
    by (auto simp add: vmtf-enqueue-def split: option.splits)
  have next-searchd:
    ‹next-searchd = (if next-search = Some L then get-next (ns ! L) else next-search)›
    using de by (auto simp: vmtf-dequeue-def)
  have abs': ‹vmtf-ℒall 𝒜 M  ((set xs', insert L (set ys')))›
    using abs-l L-in-M L-xs'-ys' unfolding vmtf-ℒall-def
    by (auto 5 5 dest: in-diffD)

  have vmtf-ns: ‹vmtf-ns (L # (ys' @ xs')) m' ns'›
  proof (cases ‹ys' @ xs'›)
    case Nil
    then have ‹fst-Asd = None›
      using fst-Asd by auto
    then show ?thesis
      using atm-L-A dequeue Nil unfolding Nil vmtf-en-dequeue-def comp-def de nsd
      by (auto simp: vmtf-ns-single-iff vmtf-enqueue-def split: option.splits)
  next
    case (Cons z zs)
    let ?m = ‹(stamp (nsd!z))›
    let ?Ad = ‹nsd[L := VMTF-Node m' None (Some z)]›
    have L-z-zs: ‹L ∉ set (z # zs)›
      using L-xs'-ys' atm-L-A unfolding Cons
      by simp
    have vmtf-ns-z: ‹vmtf-ns (z # zs) md nsd›
      using vmtf-ns' unfolding Cons .

    have vmtf-ns-A: ‹vmtf-ns (z # zs) md ?Ad›
      apply (rule vmtf-ns-eq-iffI[of - - nsd])
      subgoal using L-z-zs atm-L-A by auto
      subgoal using vmtf-ns-le-length[OF vmtf-ns-z] by auto
      subgoal using vmtf-ns-z .
      done
    have [simp]: ‹fst-Asd = Some z›
      using fst-Asd unfolding Cons by simp
    show ?thesis
      unfolding Cons
      apply (rule vmtf-ns.Cons[of - - md ?Ad - m'])
      subgoal using vmtf-ns-A .
      subgoal using atm-L-A by simp
      subgoal using atm-L-A by simp
      subgoal using L-z-zs by simp
```

```
    subgoal using L-z-zs by simp
    subgoal using m' by simp-all
    subgoal
      using atm-L-A dequeue L-z-zs unfolding Nil vmtf-en-dequeue-def comp-def de nsd
      apply (cases ‹ns-vmtf-dequeue z ns ! z›)
      by (auto simp: vmtf-ns-single-iff vmtf-enqueue-def split: option.splits)
    subgoal by linarith
    done
  qed
  have L-xs'-ys'-\mathcal{L}_all': ‹∀ L'∈set ((L # ys') @ xs'). L' ∈ atms-of (\mathcal{L}_all \mathcal{A})›
    using L L-xs'-ys'-\mathcal{L}_all by auto
  have next-search'-xs': ‹next-search' = option-hd xs'›
    using next-searchd-L-next next-search' next-searchd-hd lst-As' fst-Asd
    by (auto split: if-splits)
  show ?thesis
    apply (rule exI[of - ‹xs'›])
    apply (rule exI[of - ‹L # ys'›])
    using fst-As' next-search' abs' atm-A vmtf-ns-notin' vmtf-ns ys-xs-\mathcal{L}_all L-xs'-ys'-\mathcal{L}_all'
      next-searchd next-search'-xs'
    by simp
next
  case undef
  have next-search': ‹next-search' = Some L›
    using dequeue undef unfolding vmtf-en-dequeue-def comp-def de
    by (auto simp add: vmtf-enqueue-def split: option.splits)
  have next-searchd:
    ‹next-searchd = (if next-search = Some L then get-next (ns ! L) else next-search)›
    using de by (auto simp: vmtf-dequeue-def)
  have abs': ‹vmtf-\mathcal{L}_all \mathcal{A} M  ((insert L (set (ys' @ xs')), set []))›
    using abs-l L-xs'-ys' unfolding vmtf-\mathcal{L}_all-def
    by (auto 5 5 dest: in-diffD)

  have vmtf-ns: ‹vmtf-ns (L # (ys' @ xs')) m' ns'›
  proof (cases ‹ys' @ xs'›)
    case Nil
    then have ‹fst-Asd = None›
      using fst-Asd by auto
    then show ?thesis
      using atm-L-A dequeue Nil unfolding Nil vmtf-en-dequeue-def comp-def de nsd
      by (auto simp: vmtf-ns-single-iff vmtf-enqueue-def split: option.splits)
  next
    case (Cons z zs)
    let ?m = ‹(stamp (nsd!z))›
    let ?Ad = ‹nsd[L := VMTF-Node m' None (Some z)]›
    have L-z-zs: ‹L ∉ set (z # zs)›
      using L-xs'-ys' atm-L-A unfolding Cons
      by simp
    have vmtf-ns-z: ‹vmtf-ns (z # zs) md nsd›
      using vmtf-ns' unfolding Cons .

    have vmtf-ns-A: ‹vmtf-ns (z # zs) md ?Ad›
      apply (rule vmtf-ns-eq-iffI[of - - nsd])
      subgoal using L-z-zs atm-L-A by auto
      subgoal using vmtf-ns-le-length[OF vmtf-ns-z] by auto
      subgoal using vmtf-ns-z .
      done
```

```
      have [simp]: ‹fst-Asd = Some z›
        using fst-Asd unfolding Cons by simp
      show ?thesis
        unfolding Cons
        apply (rule vmtf-ns.Cons[of - - md ?Ad - m'])
        subgoal using vmtf-ns-A .
        subgoal using atm-L-A by simp
        subgoal using atm-L-A by simp
        subgoal using L-z-zs by simp
        subgoal using L-z-zs by simp
        subgoal using m' by simp-all
        subgoal
          using atm-L-A dequeue L-z-zs unfolding Nil vmtf-en-dequeue-def comp-def de nsd
          apply (cases ‹ns-vmtf-dequeue z ns ! z›)
          by (auto simp: vmtf-ns-single-iff vmtf-enqueue-def split: option.splits)
        subgoal by linarith
        done
    qed
    have L-xs'-ys'-\mathcal{L}_{all}': ‹∀ L'∈set ((L # ys') @ xs'). L' ∈ atms-of (\mathcal{L}_{all} \mathcal{A})›
      using L L-xs'-ys'-\mathcal{L}_{all} by auto
    show ?thesis
      apply (rule exI[of - ‹(L # ys') @ xs'›])
      apply (rule exI[of - ‹[]›])
      using fst-As' next-search' abs' atm-A vmtf-ns-notin' vmtf-ns ys-xs-\mathcal{L}_{all} L-xs'-ys'-\mathcal{L}_{all}'
        next-searchd
      by simp
  qed
qed


lemma abs-vmtf-ns-bump-vmtf-en-dequeue':
  fixes M
  assumes
    vmtf: ‹(vm) ∈ vmtf \mathcal{A} M› and
    L: ‹L ∈ atms-of (\mathcal{L}_{all} \mathcal{A})› and
    nempty: ‹isasat-input-nempty \mathcal{A}›
  shows ‹vmtf-en-dequeue M L vm ∈ vmtf \mathcal{A} M›
  using abs-vmtf-ns-bump-vmtf-en-dequeue assms by (cases vm) blast

definition (in −) vmtf-unset :: ‹nat ⇒ vmtf ⇒ vmtf› where
‹vmtf-unset = (λL (ns, m, fst-As, lst-As, next-search).
  (if next-search = None ∨ stamp (ns ! (the next-search)) < stamp (ns ! L)
  then ((ns, m, fst-As, lst-As, Some L))
  else ((ns, m, fst-As, lst-As, next-search))))›

lemma vmtf-atm-of-ys-iff:
  assumes
    vmtf-ns: ‹vmtf-ns (ys' @ xs') m ns› and
    next-search: ‹next-search = option-hd xs'› and
    abs-vmtf: ‹vmtf-\mathcal{L}_{all} \mathcal{A} M ((set xs', set ys'))› and
    L: ‹L ∈ atms-of (\mathcal{L}_{all} \mathcal{A})›
    shows ‹L ∈ set ys' ⟷ next-search = None ∨ stamp (ns ! (the next-search)) < stamp (ns ! L)›
proof −
  let ?xs' = ‹set xs'›
  let ?ys' = ‹set ys'›
  have L-xs-ys: ‹L ∈ ?xs' ∪ ?ys'›
```

```
    using abs-vmtf L unfolding vmtf-\mathcal{L}_{all}-def
    by (auto simp: in-\mathcal{L}_{all}-atm-of-in-atms-of-iff)
  have dist: ‹distinct (xs' @ ys')›
    using vmtf-ns-distinct[OF vmtf-ns] by auto

  have sorted: ‹sorted (map (λa. stamp (ns ! a)) (rev xs' @ rev ys'))› and
    distinct: ‹distinct (map (λa. stamp (ns ! a)) (xs' @ ys'))›
    using vmtf-ns-stamp-sorted[OF vmtf-ns] vmtf-ns-stamp-distinct[OF vmtf-ns]
    by (auto simp: rev-map[symmetric])
  have next-search-xs: ‹?xs' = {} ⟷ next-search = None›
    using next-search by auto

  have ‹stamp (ns ! (the next-search)) < stamp (ns ! L) ⟹ L ∉ ?xs'›
    if ‹xs' ≠ []›
    using that sorted distinct L-xs-ys unfolding next-search
    by (cases xs') (auto simp: sorted-append)
  moreover have ‹stamp (ns ! (the next-search)) < stamp (ns ! L)› (is ‹?n < ?L›)
    if xs': ‹xs' ≠ []› and ‹L ∈ ?ys'›
  proof -
    have ‹?n ≤ ?L›
      using vmtf-ns-stamp-sorted[OF vmtf-ns] that last-in-set[OF xs']
      by (cases xs')
        (auto simp: rev-map[symmetric] next-search sorted-append sorted2)
    moreover have ‹?n ≠ ?L›
      using vmtf-ns-stamp-distinct[OF vmtf-ns] that last-in-set[OF xs']
      by (cases xs') (auto simp: rev-map[symmetric] next-search)
    ultimately show ?thesis
      by arith
  qed
  ultimately show ?thesis
    using L-xs-ys next-search-xs dist by auto
qed


lemma abs-vmtf-ns-unset-vmtf-unset:
  assumes vmtf:‹((ns, m, fst-As, lst-As, next-search)) ∈ vmtf \mathcal{A} M› and
  L-N: ‹L ∈ atms-of (\mathcal{L}_{all} \mathcal{A})›
  shows ‹(vmtf-unset L ((ns, m, fst-As, lst-As, next-search))) ∈ vmtf \mathcal{A} M› (is ‹?S ∈ -›)
proof -
  obtain xs' ys' where
    vmtf-ns: ‹vmtf-ns (ys' @ xs') m ns› and
    fst-As: ‹fst-As = hd (ys' @ xs')› and
    lst-As: ‹lst-As = last (ys' @ xs')› and
    next-search: ‹next-search = option-hd xs'› and
    abs-vmtf: ‹vmtf-\mathcal{L}_{all} \mathcal{A} M ((set xs', set ys'))› and
    notin: ‹vmtf-ns-notin (ys' @ xs') m ns› and
    atm-A: ‹∀ L∈atms-of (\mathcal{L}_{all} \mathcal{A}). L < length ns› and
    L-ys'-xs'-\mathcal{L}_{all}: ‹∀ L∈set (ys' @ xs'). L ∈ atms-of (\mathcal{L}_{all} \mathcal{A})›
    using vmtf unfolding vmtf-def by fast
  obtain ns' m' fst-As' next-search' lst-As' where
    S: ‹?S = ((ns', m', fst-As', lst-As', next-search'))›
    by (cases ?S) auto
  have L-ys'-iff: ‹L ∈ set ys' ⟷ (next-search = None ∨ stamp (ns ! the next-search) < stamp (ns !
L))›
    using vmtf-atm-of-ys-iff[OF vmtf-ns next-search abs-vmtf L-N] .
  have ‹L ∈ set (xs' @ ys')›
```

```
  using abs-vmtf L-N unfolding vmtf-\mathcal{L}_{all}-def by auto
then have L-ys'-xs': ‹L ∈ set ys' ⟷ L ∉ set xs'›
  using vmtf-ns-distinct[OF vmtf-ns] by auto
have ‹∃ xs' ys'.
    vmtf-ns (ys' @ xs') m' ns' ∧
    fst-As' = hd (ys' @ xs') ∧
    lst-As' = last (ys' @ xs') ∧
    next-search' = option-hd xs' ∧
    vmtf-\mathcal{L}_{all} \mathcal{A} M ((set xs', set ys')) ∧
    vmtf-ns-notin (ys' @ xs') m' ns' ∧ (∀ L∈atms-of (\mathcal{L}_{all} \mathcal{A}). L < length ns') ∧
    (∀ L∈set (ys' @ xs'). L ∈ atms-of (\mathcal{L}_{all} \mathcal{A}))›
proof (cases ‹L ∈ set xs'›)
  case True
  then have C: ‹¬(next-search = None ∨ stamp (ns ! the next-search) < stamp (ns ! L))›
    by (subst L-ys'-iff[symmetric]) (use L-ys'-xs' in auto)
  show ?thesis
    using S True unfolding vmtf-unset-def L-ys'-xs'[symmetric]
    apply −
    apply (simp add: C)
    using vmtf-ns fst-As next-search abs-vmtf notin atm-A L-ys'-xs'-\mathcal{L}_{all} lst-As
    by auto
next
  case False
  then have C: ‹next-search = None ∨ stamp (ns ! the next-search) < stamp (ns ! L)›
    by (subst L-ys'-iff[symmetric]) (use L-ys'-xs' in auto)
  have L-ys: ‹L ∈ set ys'›
    by (use False L-ys'-xs' in auto)
  define y-ys where ‹y-ys ≡ takeWhile ((≠) L) ys'›
  define x-ys where ‹x-ys ≡ drop (length y-ys) ys'›
  let ?ys' = ‹y-ys›
  let ?xs' = ‹x-ys @ xs'›
  have x-ys-take-ys': ‹y-ys = take (length y-ys) ys'›
     unfolding y-ys-def
     by (subst take-length-takeWhile-eq-takeWhile[of ‹(≠) L› ‹ys'›, symmetric]) standard
  have ys'-y-x: ‹ys' = y-ys @ x-ys›
    by (subst x-ys-take-ys') (auto simp: x-ys-def)
  have y-ys-le-ys': ‹length y-ys < length ys'›
    using L-ys by (metis (full-types) append-eq-conv-conj append-self-conv le-antisym
      length-takeWhile-le not-less takeWhile-eq-all-conv x-ys-take-ys' y-ys-def)
  from nth-length-takeWhile[OF this[unfolded y-ys-def]] have [simp]: ‹x-ys ≠ []› ‹hd x-ys = L›
    using y-ys-le-ys' unfolding x-ys-def y-ys-def
    by (auto simp: x-ys-def y-ys-def hd-drop-conv-nth)
  have [simp]: ‹ns' = ns› ‹m' = m› ‹fst-As' = fst-As› ‹next-search' = Some L›
    ‹lst-As' = lst-As›
    using S unfolding vmtf-unset-def by (auto simp: C)

  have ‹vmtf-ns (?ys' @ ?xs') m ns›
    using vmtf-ns unfolding ys'-y-x by simp
  moreover have ‹fst-As' = hd (?ys' @ ?xs')›
    using fst-As unfolding ys'-y-x by simp
  moreover have ‹lst-As' = last (?ys' @ ?xs')›
    using lst-As unfolding ys'-y-x by simp
  moreover have ‹next-search' = option-hd ?xs'›
    by auto
  moreover
    have ‹vmtf-\mathcal{L}_{all} \mathcal{A} M ((set ?xs', set ?ys'))›
```

```
    using abs-vmtf vmtf-ns-distinct[OF vmtf-ns] unfolding vmtf-\mathcal{L}_{all}-def ys'-y-x
    by auto
  moreover have ‹vmtf-ns-notin (?ys' @ ?xs') m ns›
    using notin unfolding ys'-y-x by simp
  moreover have ‹\forall L\in set (?ys' @ ?xs'). L \in atms-of (\mathcal{L}_{all} \mathcal{A})›
    using L-ys'-xs'-\mathcal{L}_{all} unfolding ys'-y-x by auto
  ultimately show ?thesis
    using S False atm-A unfolding vmtf-unset-def L-ys'-xs'[symmetric]
    by (fastforce simp add: C)
 qed
 then show ?thesis
  unfolding vmtf-def S
  by fast
qed


definition (in -) vmtf-dequeue-pre where
 ‹vmtf-dequeue-pre = (\lambda(L,ns). L < length ns \wedge
       (get-next (ns!L) \neq None \longrightarrow the (get-next (ns!L)) < length ns) \wedge
       (get-prev (ns!L) \neq None \longrightarrow the (get-prev (ns!L)) < length ns))›

lemma (in -) vmtf-dequeue-pre-alt-def:
 ‹vmtf-dequeue-pre = (\lambda(L, ns). L < length ns \wedge
       (\forall a. Some a = get-next (ns!L) \longrightarrow a < length ns) \wedge
       (\forall a. Some a = get-prev (ns!L) \longrightarrow a < length ns))›
 apply (intro ext, rename-tac x)
 subgoal for x
  by (cases ‹get-next ((snd x)!(fst x))›; cases ‹get-prev ((snd x)!(fst x))›)
   (auto simp: vmtf-dequeue-pre-def intro!: ext)
 done

definition vmtf-en-dequeue-pre :: ‹nat multiset \Rightarrow ((nat, nat) ann-lits \times nat) \times vmtf \Rightarrow bool› where
 ‹vmtf-en-dequeue-pre \mathcal{A} = (\lambda((M, L),(ns,m,fst-As, lst-As, next-search)).
     L < length ns \wedge vmtf-dequeue-pre (L, ns) \wedge
     fst-As < length ns \wedge (get-next (ns ! fst-As) \neq None \longrightarrow get-prev (ns ! lst-As) \neq None) \wedge
     (get-next (ns ! fst-As) = None \longrightarrow fst-As = lst-As) \wedge
     m+1 \leq unat64-max \wedge
     Pos L \in# \mathcal{L}_{all} \mathcal{A})›

lemma (in -) id-reorder-list:
  ‹(RETURN o id, reorder-list vm) \in \langlenat-rel\rangle list-rel \rightarrow_f \langle\langlenat-rel\rangle list-rel\rangle nres-rel›
 unfolding reorder-list-def by (intro frefI nres-relI) auto

lemma vmtf-vmtf-en-dequeue-pre-to-remove:
 assumes vmtf: ‹((ns, m, fst-As, lst-As, next-search)) \in vmtf \mathcal{A} M› and
   m-le: ‹m + 1 \leq unat64-max› and
   nempty: ‹isasat-input-nempty \mathcal{A}› and
   A: ‹A \in atms-of (\mathcal{L}_{all} \mathcal{A})›
 shows ‹vmtf-en-dequeue-pre \mathcal{A} ((M, A), (ns, m, fst-As, lst-As, next-search))›
proof -
 obtain xs' ys' where
   vmtf-ns: ‹vmtf-ns (ys' @ xs') m ns› and
   fst-As: ‹fst-As = hd (ys' @ xs')› and
   lst-As: ‹lst-As = last (ys' @ xs')› and
   next-search: ‹next-search = option-hd xs'› and
   abs-vmtf: ‹vmtf-\mathcal{L}_{all} \mathcal{A} M ((set xs', set ys'))› and
```

notin: ‹*vmtf-ns-notin* (*ys′* @ *xs′*) *m ns*› **and**
atm-A: ‹∀ *L*∈*atms-of* (*$\mathcal{L}_{all}$* *$\mathcal{A}$*). *L* < *length ns*› **and**
L-ys′-xs′-$\mathcal{L}_{all}$: ‹∀ *L*∈*set* (*ys′* @ *xs′*). *L* ∈ *atms-of* (*$\mathcal{L}_{all}$* *$\mathcal{A}$*)›
using *vmtf* **unfolding** *vmtf-def* **by** *fast*
have [*dest*]: *False* **if** ‹*ys′* = []› **and** ‹*xs′* = []›
proof −
have *1*: ‹*set-mset* *$\mathcal{A}$* = {}›
using *abs-vmtf* **unfolding** *that vmtf-$\mathcal{L}_{all}$-def* **by** (*auto simp*: *atms-of-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$*)
then show *?thesis*
using *nempty* **by** *auto*
qed

have ‹*A* ∈ *atms-of* (*$\mathcal{L}_{all}$* *$\mathcal{A}$*)›
using *abs-vmtf A* **unfolding** *vmtf-$\mathcal{L}_{all}$-def* **by** *auto*
then have *remove-i-le-A*: ‹*A* < *length ns*› **and**
i-L: ‹*Pos A* ∈# *$\mathcal{L}_{all}$* *$\mathcal{A}$*›
using *atm-A* **by** (*auto simp*: *in-$\mathcal{L}_{all}$-atm-of-$\mathcal{A}_{in}$ atms-of-def*)
moreover have ‹*fst-As* < *length ns*›
using *fst-As atm-A L-ys′-xs′-$\mathcal{L}_{all}$* **by** (*cases ys′*; *cases xs′*) *auto*
moreover have ‹*get-prev* (*ns* ! *lst-As*) ≠ *None*› **if** ‹*get-next* (*ns* ! *fst-As*) ≠ *None*›
using *that vmtf-ns-hd-next*[*of* ‹*hd* (*ys′* @ *xs′*)› ‹*hd* (*tl* (*ys′* @ *xs′*))› ‹*tl* (*tl* (*ys′* @ *xs′*))›]
vmtf-ns vmtf-ns-last-prev[*of* ‹*butlast* (*ys′* @ *xs′*)› ‹*last* (*ys′* @ *xs′*)›]
vmtf-ns-last-next[*of* ‹*butlast* (*ys′* @ *xs′*)› ‹*last* (*ys′* @ *xs′*)›]
by (*cases* ‹*ys′* @ *xs′*›; *cases* ‹*tl* (*ys′* @ *xs′*)›)
(*auto simp*: *fst-As lst-As*)
moreover have ‹*vmtf-dequeue-pre* (*A*, *ns*)›
proof −
have ‹*A* < *length ns*›
using *A abs-vmtf atm-A* **unfolding** *vmtf-$\mathcal{L}_{all}$-def* **by** *auto*
moreover have ‹*y* < *length ns*› **if** *get-next*: ‹*get-next* (*ns* ! (*A*)) = *Some y*› **for** *y*
proof (*cases* ‹*A* ∈ *set* (*ys′* @ *xs′*)›)
case *False*
then show *?thesis*
using *notin get-next remove-i-le-A* **by** (*auto simp*: *vmtf-ns-notin-def*)
next
case *True*
then obtain *zs zs′* **where** *zs*: ‹*ys′* @ *xs′* = *zs′* @ [*A*] @ *zs*›
using *split-list* **by** *fastforce*
moreover have ‹*set* (*ys′* @ *xs′*) = *atms-of* (*$\mathcal{L}_{all}$* *$\mathcal{A}$*)›
using *abs-vmtf* **unfolding** *vmtf-$\mathcal{L}_{all}$-def* **by** *auto*
ultimately show *?thesis*
using *vmtf-ns-last-mid-get-next-option-hd*[*of zs′ A zs m ns*] *vmtf-ns atm-A get-next*
L-ys′-xs′-$\mathcal{L}_{all}$ **unfolding** *zs* **by** *force*
qed
moreover have ‹*y* < *length ns*› **if** *get-prev*: ‹*get-prev* (*ns* ! (*A*)) = *Some y*› **for** *y*
proof (*cases* ‹*A* ∈ *set* (*ys′* @ *xs′*)›)
case *False*
then show *?thesis*
using *notin get-prev remove-i-le-A* **by** (*auto simp*: *vmtf-ns-notin-def*)
next
case *True*
then obtain *zs zs′* **where** *zs*: ‹*ys′* @ *xs′* = *zs′* @ [*A*] @ *zs*›
using *split-list* **by** *fastforce*
moreover have ‹*set* (*ys′* @ *xs′*) = *atms-of* (*$\mathcal{L}_{all}$* *$\mathcal{A}$*)›
using *abs-vmtf* **unfolding** *vmtf-$\mathcal{L}_{all}$-def* **by** *auto*
ultimately show *?thesis*

```
      using vmtf-ns-last-mid-get-prev-option-last[of zs' A zs m ns] vmtf-ns atm-A get-prev
        L-ys'-xs'-\<L>_all unfolding zs by force
    qed
    ultimately show ?thesis
      unfolding vmtf-dequeue-pre-def by auto
  qed
  moreover have ‹get-next (ns ! fst-As) = None ⟶ fst-As = lst-As›
    using vmtf-ns-hd-next[of ‹hd (ys' @ xs')› ‹hd (tl (ys' @ xs'))› ‹tl (tl (ys' @ xs'))›]
      vmtf-ns vmtf-ns-last-prev[of ‹butlast (ys' @ xs')› ‹last (ys' @ xs')›]
      vmtf-ns-last-next[of ‹butlast (ys' @ xs')› ‹last (ys' @ xs')›]
    by (cases ‹ys' @ xs'›; cases ‹tl (ys' @ xs')›)
       (auto simp: fst-As lst-As)
  ultimately show ?thesis
    using m-le unfolding vmtf-en-dequeue-pre-def by auto
qed

lemma vmtf-vmtf-en-dequeue-pre-to-remove':
  assumes vmtf: ‹(vm) ∈ vmtf \<A> M› and
    i: ‹A ∈ atms-of (\<L>_all \<A>)› and ‹fst (snd vm) + 1 ≤ unat64-max› and
    A: ‹isasat-input-nempty \<A>›
  shows ‹vmtf-en-dequeue-pre \<A> ((M, A), vm)›
  using vmtf-vmtf-en-dequeue-pre-to-remove assms
  by (cases vm) auto

lemma wf-vmtf-get-next:
  assumes vmtf: ‹((ns, m, fst-As, lst-As, next-search)) ∈ vmtf \<A> M›
  shows ‹wf {(get-next (ns ! the a), a) |a. a ≠ None ∧ the a ∈ atms-of (\<L>_all \<A>)}› (is ‹wf ?R›)
proof (rule ccontr)
  assume ‹¬ ?thesis›
  then obtain f where
    f: ‹(f (Suc i), f i) ∈ ?R› for i
    unfolding wf-iff-no-infinite-down-chain by blast

  obtain xs' ys' where
    vmtf-ns: ‹vmtf-ns (ys' @ xs') m ns› and
    fst-As: ‹fst-As = hd (ys' @ xs')› and
    lst-As: ‹lst-As = last (ys' @ xs')› and
    next-search: ‹next-search = option-hd xs'› and
    abs-vmtf: ‹vmtf-\<L>_all \<A> M ((set xs', set ys'))› and
    notin: ‹vmtf-ns-notin (ys' @ xs') m ns› and
    atm-A: ‹∀ L∈atms-of (\<L>_all \<A>). L < length ns›
    using vmtf unfolding vmtf-def by fast
  let ?f0 = ‹the (f 0)›
  have f-None: ‹f i ≠ None› for i
    using f[of i] by fast
  have f-Suc : ‹f (Suc n) = get-next (ns ! the (f n))› for n
    using f[of n] by auto
  have f0-length: ‹?f0 < length ns›
    using f[of 0] atm-A
    by auto
  have ‹?f0 ∈ set (ys' @ xs')›
    apply (rule ccontr)
    using notin f-Suc[of 0] f0-length unfolding vmtf-ns-notin-def
    by (auto simp: f-None)
  then obtain i0 where
    i0: ‹(ys' @ xs') ! i0 = ?f0› ‹i0 < length (ys' @ xs')›
```

```
    by (meson in-set-conv-nth)
  define zs where ‹zs = ys' @ xs'›
  have H: ‹ys' @ xs' = take m (ys' @ xs') @ [(ys' @ xs') ! m, (ys' @ xs') ! (m+1)] @
     drop (m+2) (ys' @ xs')›
    if ‹m+1 < length (ys' @ xs')›
    for m
    using that
    unfolding zs-def[symmetric]
    apply −
    apply (subst id-take-nth-drop[of m])
    by (auto simp: Cons-nth-drop-Suc simp del: append-take-drop-id)

  have ‹the (f n) = (ys' @ xs') ! (i0 + n) ∧ i0 + n < length (ys' @ xs')› for n
  proof (induction n)
    case 0
    then show ?case using i0 by simp
  next
    case (Suc n')
    have i0-le: ‹i0 + n' + 1 < length (ys' @ xs')›
    proof (rule ccontr)
      assume ‹¬ ?thesis›
      then have ‹i0 + n' + 1 = length (ys' @ xs')›
        using Suc by auto
      then have ‹ys' @ xs' = butlast (ys' @ xs') @ [the (f n')]›
        using Suc by (metis add-diff-cancel-right' append-butlast-last-id length-0-conv
            length-butlast less-one not-add-less2 nth-append-length)
      then show False
        using vmtf-ns-last-next[of ‹butlast (ys' @ xs')› ‹the (f n')› m ns] vmtf-ns
         f-Suc[of n'] by (auto simp: f-None)
    qed
    have get-next: ‹get-next (ns ! ((ys' @ xs') ! (i0 + n'))) = Some ((ys' @ xs') ! (i0 + n' + 1))›
      apply(rule vmtf-ns-last-mid-get-next[of ‹take (i0 + n') (ys' @ xs')›
        ‹(ys' @ xs') ! (i0 + n')›
        ‹(ys' @ xs') ! ((i0 + n') + 1)›
        ‹drop ((i0 + n') + 2) (ys' @ xs')›
        m ns])
      apply (subst H[symmetric])
      subgoal using i0-le .
      subgoal using vmtf-ns by simp
      done
    then show ?case
      using f-Suc[of n'] Suc i0-le by auto
  qed
  then show False
    by blast
qed

lemma vmtf-next-search-take-next:
  assumes
    vmtf: ‹((ns, m, fst-As, lst-As, next-search)) ∈ vmtf 𝒜 M› and
    n: ‹next-search ≠ None› and
    def-n: ‹defined-lit M (Pos (the next-search))›
  shows ‹(ns, m, fst-As, lst-As, get-next (ns!the next-search)) ∈ vmtf 𝒜 M›
  unfolding vmtf-def
proof clarify
  obtain xs' ys' where
```

vmtf-ns: ‹*vmtf-ns* (*ys'* @ *xs'*) *m ns*› **and**
fst-As: ‹*fst-As* = *hd* (*ys'* @ *xs'*)› **and**
lst-As: ‹*lst-As* = *last* (*ys'* @ *xs'*)› **and**
next-search: ‹*next-search* = *option-hd xs'*› **and**
abs-vmtf: ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* (*set xs'*, *set ys'*)› **and**
notin: ‹*vmtf-ns-notin* (*ys'* @ *xs'*) *m ns*› **and**
atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*› **and**
ys'-xs'-$\mathcal{L}_{all}$: ‹∀ *L*∈*set* (*ys'* @ *xs'*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *vmtf* **unfolding** *vmtf-def* **by** *fast*
let *?xs'* = ‹*tl xs'*›
let *?ys'* = ‹*ys'* @ [*hd xs'*]›
have [*simp*]: ‹*xs'* ≠ []›
using *next-search n* **by** *auto*
have ‹*vmtf-ns* (*?ys'* @ *?xs'*) *m ns*›
using *vmtf-ns* **by** (*cases xs'*) *auto*
moreover have ‹*fst-As* = *hd* (*?ys'* @ *?xs'*)›
using *fst-As* **by** *auto*
moreover have ‹*lst-As* = *last* (*?ys'* @ *?xs'*)›
using *lst-As* **by** *auto*
moreover have ‹*get-next* (*ns* ! *the next-search*) = *option-hd ?xs'*›
using *next-search n vmtf-ns*
by (*cases xs'*) (*auto dest*: *vmtf-ns-last-mid-get-next-option-hd*)
moreover {
have [*dest*]: ‹*defined-lit M* (*Pos a*) ⟹ *a* ∈ *atm-of* ‘ *lits-of-l M*› **for** *a*
by (*auto simp*: *defined-lit-map lits-of-def*)
have ‹*vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M* (*set ?xs'*, *set ?ys'*)›
using *abs-vmtf def-n next-search n vmtf-ns-distinct*[*OF vmtf-ns*]
unfolding *vmtf-*$\mathcal{L}_{all}$*-def*
by (*cases xs'*) *auto* **}**
moreover have ‹*vmtf-ns-notin* (*?ys'* @ *?xs'*) *m ns*›
using *notin* **by** *auto*
moreover have ‹∀ *L*∈*set* (*?ys'* @ *?xs'*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *ys'-xs'-*$\mathcal{L}_{all}$ **by** *auto*
ultimately show ‹∃ *xs' ys'*. *vmtf-ns* (*ys'* @ *xs'*) *m ns* ∧
fst-As = *hd* (*ys'* @ *xs'*) ∧
lst-As = *last* (*ys'* @ *xs'*) ∧
get-next (*ns* ! *the next-search*) = *option-hd xs'* ∧
vmtf-$\mathcal{L}_{all}$ $\mathcal{A}$ *M* ((*set xs'*, *set ys'*)) ∧
vmtf-ns-notin (*ys'* @ *xs'*) *m ns* ∧
(∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*) ∧
(∀ *L*∈*set* (*ys'* @ *xs'*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
using *atm-A* **by** *blast*
qed

definition *vmtf-find-next-undef* :: ‹*nat multiset* ⇒ *vmtf* ⇒ (*nat*, *nat*) *ann-lits* ⇒ (*nat option*) *nres*›
where
‹*vmtf-find-next-undef* $\mathcal{A}$ = (λ(*ns*, *m*, *fst-As*, *lst-As*, *next-search*) *M*. *do* {
WHILE$_T$λ*next-search*. (*ns*, *m*, *fst-As*, *lst-As*, *next-search*) ∈ *vmtf* $\mathcal{A}$ *M* ∧ (*next-search* ≠ *None* ⟶ *Pos* (*the next-searc*
(λ*next-search*. *next-search* ≠ *None* ∧ *defined-lit M* (*Pos* (*the next-search*)))
(λ*next-search*. *do* {
ASSERT(*next-search* ≠ *None*);
let n = *the next-search*;
ASSERT(*Pos n* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$);
ASSERT (*n* < *length ns*);
RETURN (*get-next* (*ns*!*n*))

```
      }
    )
    next-search
})›

lemma vmtf-find-next-undef-ref:
  assumes
    vmtf: ‹(ns, m, fst-As, lst-As, next-search) ∈ vmtf A M›
  shows ‹vmtf-find-next-undef A (ns, m, fst-As, lst-As, next-search) M
    ≤ ⇓ Id (SPEC (λL. (ns, m, fst-As, lst-As, L) ∈ vmtf A M ∧
      (L = None ⟶ (∀ L∈#\mathcal{L}_{all} A. defined-lit M L)) ∧
      (L ≠ None ⟶ Pos (the L) ∈# \mathcal{L}_{all} A ∧ undefined-lit M (Pos (the L)))))›
proof −
  obtain xs' ys' where
    vmtf-ns: ‹vmtf-ns (ys' @ xs') m ns› and
    fst-As: ‹fst-As = hd (ys' @ xs')› and
    lst-As: ‹lst-As = last (ys' @ xs')› and
    next-search: ‹next-search = option-hd xs'› and
    abs-vmtf: ‹vmtf-\mathcal{L}_{all} A M (set xs', set ys')› and
    notin: ‹vmtf-ns-notin (ys' @ xs') m ns› and
    atm-A: ‹∀ L∈atms-of (\mathcal{L}_{all} A). L < length ns›
    using vmtf unfolding vmtf-def by fast
  have no-next-search-all-defined:
    ‹(ns', m', fst-As', lst-As', None) ∈ vmtf A M ⟹ x ∈# \mathcal{L}_{all} A ⟹ defined-lit M x›
    for x ns' m' fst-As' lst-As' remove
    by (auto simp: vmtf-def vmtf-\mathcal{L}_{all}-def in-\mathcal{L}_{all}-atm-of-in-atms-of-iff
      defined-lit-map lits-of-def)
  have next-search-\mathcal{L}_{all}:
    ‹(ns', m', fst-As', lst-As', Some y) ∈ vmtf A M ⟹ y ∈ atms-of (\mathcal{L}_{all} A)›
    for ns' m' fst-As' remove y lst-As'
    by (auto simp: vmtf-def vmtf-\mathcal{L}_{all}-def in-\mathcal{L}_{all}-atm-of-in-atms-of-iff
      defined-lit-map lits-of-def)
  have next-search-le-A':
    ‹(ns', m', fst-As', lst-As', Some y) ∈ vmtf A M ⟹ y < length ns'›
    for ns' m' fst-As' remove y lst-As'
    by (auto simp: vmtf-def vmtf-\mathcal{L}_{all}-def in-\mathcal{L}_{all}-atm-of-in-atms-of-iff
      defined-lit-map lits-of-def)
  show ?thesis
    unfolding vmtf-find-next-undef-def
    apply (refine-vcg
      WHILEIT-rule[where R=‹{(get-next (ns ! the a), a) |a. a ≠ None ∧ the a ∈ atms-of (\mathcal{L}_{all} A)}›])
    subgoal using vmtf by (rule wf-vmtf-get-next)
    subgoal using next-search vmtf by auto
   subgoal using vmtf by (auto dest!: next-search-\mathcal{L}_{all} simp: image-image in-\mathcal{L}_{all}-atm-of-in-atms-of-iff)
    subgoal using vmtf by auto
    subgoal using vmtf by auto
    subgoal using vmtf by (auto dest: next-search-le-A')
    subgoal by (auto simp: image-image in-\mathcal{L}_{all}-atm-of-in-atms-of-iff)
      (metis next-search-\mathcal{L}_{all} option.distinct(1) option.sel vmtf-next-search-take-next)
    subgoal by (auto simp: image-image in-\mathcal{L}_{all}-atm-of-in-atms-of-iff)
      (metis next-search-\mathcal{L}_{all} option.distinct(1) option.sel vmtf-next-search-take-next)
    subgoal by (auto dest: no-next-search-all-defined next-search-\mathcal{L}_{all})
    subgoal by (auto dest: next-search-le-A')
    subgoal for x1 ns' x2 m' x2a fst-As' next-search' x2c s
      by (auto dest: no-next-search-all-defined next-search-\mathcal{L}_{all})
    subgoal by (auto dest: vmtf-next-search-take-next)
```

subgoal by (*auto simp*: *image-image in-$\mathcal{L}_{all}$-atm-of-in-atms-of-iff*)
done
qed

lemma *vmtf-unset-vmtf-tl*:
fixes *M*
defines [*simp*]: ‹*L* ≡ *atm-of* (*lit-of* (*hd M*))›
assumes *vmtf*:‹(*ns*, *m*, *fst-As*, *lst-As*, *next-search*) ∈ *vmtf* $\mathcal{A}$ *M*› **and**
L-N: ‹*L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)› **and** [*simp*]: ‹*M* ≠ []›
shows ‹*vmtf-unset L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*) ∈ *vmtf* $\mathcal{A}$ (*tl M*)›
(**is** ‹*?S* ∈ -›)
proof −
obtain *xs′ ys′* **where**
vmtf-ns: ‹*vmtf-ns* (*ys′* @ *xs′*) *m ns*› **and**
fst-As: ‹*fst-As* = *hd* (*ys′* @ *xs′*)› **and**
lst-As: ‹*lst-As* = *last* (*ys′* @ *xs′*)› **and**
next-search: ‹*next-search* = *option-hd xs′*› **and**
abs-vmtf: ‹*vmtf-$\mathcal{L}_{all}$* $\mathcal{A}$ *M* (*set xs′*, *set ys′*)› **and**
notin: ‹*vmtf-ns-notin* (*ys′* @ *xs′*) *m ns*› **and**
atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns*› **and**
ys′-xs′-$\mathcal{L}_{all}$: ‹∀ *L*∈*set* (*ys′* @ *xs′*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
using *vmtf* **unfolding** *vmtf-def* **by** *fast*
obtain *ns′ m′ fst-As′ next-search′ lst-As′* **where**
S: ‹*?S* = (*ns′*, *m′*, *fst-As′*, *lst-As′*, *next-search′*)›
by (*cases ?S*) *auto*
have *L-ys′-iff*: ‹*L* ∈ *set ys′* ⟷ (*next-search* = *None* ∨ *stamp* (*ns* ! *the next-search*) < *stamp* (*ns* ! *L*))›
using *vmtf-atm-of-ys-iff*[*OF vmtf-ns next-search abs-vmtf L-N*] **.**
have *dist*: ‹*distinct* (*ys′* @ *xs′*)›
using *vmtf-ns-distinct*[*OF vmtf-ns*] **.**
have ‹*L* ∈ *set* (*xs′* @ *ys′*)›
using *abs-vmtf L-N* **unfolding** *vmtf-$\mathcal{L}_{all}$-def* **by** *auto*
then have *L-ys′-xs′*: ‹*L* ∈ *set ys′* ⟷ *L* ∉ *set xs′*›
using *dist* **by** *auto*
have ‹∃ *xs′ ys′*.
vmtf-ns (*ys′* @ *xs′*) *m′ ns′* ∧
fst-As′ = *hd* (*ys′* @ *xs′*) ∧
lst-As′ = *last* (*ys′* @ *xs′*) ∧
next-search′ = *option-hd xs′* ∧
vmtf-$\mathcal{L}_{all}$ $\mathcal{A}$ (*tl M*) (*set xs′*, *set ys′*) ∧
vmtf-ns-notin (*ys′* @ *xs′*) *m′ ns′* ∧ (∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length ns′*) ∧
(∀ *L*∈*set* (*ys′* @ *xs′*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
proof (*cases* ‹*L* ∈ *set xs′*›)
case *True*
then have *C*[*unfolded L-def*]: ‹¬(*next-search* = *None* ∨ *stamp* (*ns* ! *the next-search*) < *stamp* (*ns* ! *L*))›
by (*subst L-ys′-iff*[*symmetric*]) (*use L-ys′-xs′* **in** *auto*)
have *abs-vmtf*: ‹*vmtf-$\mathcal{L}_{all}$* $\mathcal{A}$ (*tl M*) (*set xs′*, *set ys′*)›
using *S abs-vmtf dist L-ys′-xs′ True* **unfolding** *vmtf-$\mathcal{L}_{all}$-def vmtf-unset-def*
by (*cases M*) (*auto simp*: *C*)
show *?thesis*
using *S True* **unfolding** *vmtf-unset-def L-ys′-xs′*[*symmetric*]
apply −
apply (*simp add*: *C*)

```
    using vmtf-ns fst-As next-search abs-vmtf notin atm-A ys'-xs'-\<L>_all lst-As
    by auto
  next
    case False
    then have C[unfolded L-def]: ‹next-search = None ∨ stamp (ns ! the next-search) < stamp (ns ! L)›
      by (subst L-ys'-iff[symmetric]) (use L-ys'-xs' in auto)
    have L-ys: ‹L ∈ set ys'›
      by (use False L-ys'-xs' in auto)
    define y-ys where ‹y-ys ≡ takeWhile ((≠) L) ys'›
    define x-ys where ‹x-ys ≡ drop (length y-ys) ys'›
    let ?ys' = ‹y-ys›
    let ?xs' = ‹x-ys @ xs'›
    have x-ys-take-ys': ‹y-ys = take (length y-ys) ys'›
      unfolding y-ys-def
      by (subst take-length-takeWhile-eq-takeWhile[of ‹(≠) L› ‹ys'›, symmetric]) standard
    have ys'-y-x: ‹ys' = y-ys @ x-ys›
      by (subst x-ys-take-ys') (auto simp: x-ys-def)
    have y-ys-le-ys': ‹length y-ys < length ys'›
      using L-ys by (metis (full-types) append-eq-conv-conj append-self-conv le-antisym
        length-takeWhile-le not-less takeWhile-eq-all-conv x-ys-take-ys' y-ys-def)
    from nth-length-takeWhile[OF this[unfolded y-ys-def]] have [simp]: ‹x-ys ≠ []› ‹hd x-ys = L›
      using y-ys-le-ys' unfolding x-ys-def y-ys-def
      by (auto simp: x-ys-def y-ys-def hd-drop-conv-nth)
    have [simp]: ‹ns' = ns› ‹m' = m› ‹fst-As' = fst-As› ‹next-search' = Some (atm-of (lit-of (hd M)))›
      ‹lst-As' = lst-As›
      using S unfolding vmtf-unset-def by (auto simp: C)
    have L-y-ys: ‹L ∉ set y-ys›
      unfolding y-ys-def by (metis (full-types) takeWhile-eq-all-conv takeWhile-idem)
    have ‹vmtf-ns (?ys' @ ?xs') m ns›
      using vmtf-ns unfolding ys'-y-x by simp
    moreover have ‹fst-As' = hd (?ys' @ ?xs')›
      using fst-As unfolding ys'-y-x by simp
    moreover have ‹lst-As' = last (?ys' @ ?xs')›
      using lst-As unfolding ys'-y-x by simp
    moreover have ‹next-search' = option-hd ?xs'›
      by auto
    moreover {
      have ‹vmtf-\<L>_all \<A> M (set ?xs', set ?ys')›
        using abs-vmtf dist unfolding vmtf-\<L>_all-def ys'-y-x
        by auto
      then have ‹vmtf-\<L>_all \<A> (tl M) (set ?xs', set ?ys')›
        using dist L-y-ys unfolding vmtf-\<L>_all-def ys'-y-x ys'-y-x
        by (cases M) auto
      }
    moreover have ‹vmtf-ns-notin (?ys' @ ?xs') m ns›
      using notin unfolding ys'-y-x by simp
    moreover have ‹∀ L∈set (?ys' @ ?xs'). L ∈ atms-of (\<L>_all \<A>)›
      using ys'-xs'-\<L>_all unfolding ys'-y-x by simp
    ultimately show ?thesis
      using S False atm-A unfolding vmtf-unset-def L-ys'-xs'[symmetric]
      by (fastforce simp add: C)
  qed
  then show ?thesis
    unfolding vmtf-def S
    by fast
qed
```

lemma *vmtf-ns-Cons*:
  assumes
    vmtf: ‹*vmtf-ns* (*b* # *l*) *m xs*› **and**
    a-xs: ‹*a* < *length xs*› **and**
    ab: ‹*a* ≠ *b*› **and**
    a-l: ‹*a* ∉ *set l*› **and**
    nm: ‹*n* > *m*› **and**
    xs': ‹*xs'* = *xs*[*a* := *VMTF-Node n None* (*Some b*),
        b := *VMTF-Node* (*stamp* (*xs*!*b*)) (*Some a*) (*get-next* (*xs*!*b*))]› **and**
    nn': ‹*n'* ≥ *n*›
  shows ‹*vmtf-ns* (*a* # *b* # *l*) *n' xs'*›
proof −
  have ‹*vmtf-ns* (*b* # *l*) *m* (*xs*[*a* := *VMTF-Node n None* (*Some b*)])›
    apply (*rule vmtf-ns-eq-iffI*[*OF* - - *vmtf*])
    subgoal using *ab a-l a-xs* **by** *auto*
    subgoal using *a-xs vmtf-ns-le-length*[*OF vmtf*] **by** *auto*
    done
  then show *?thesis*
    apply (*rule vmtf-ns.Cons*[*of* - - - - - *n*])
    subgoal using *a-xs* **by** *simp*
    subgoal using *a-xs* **by** *simp*
    subgoal using *ab* **.**
    subgoal using *a-l* **.**
    subgoal using *nm* **.**
    subgoal using *xs' ab a-xs* **by** (*cases* ‹*xs* ! *b*›) *auto*
    subgoal using *nn'* **.**
    done
qed

definition (**in** −) *vmtf-cons* **where**
‹*vmtf-cons ns L cnext st* =
  (*let*
    ns = *ns*[*L* := *VMTF-Node* (*Suc st*) *None cnext*];
    ns = (*case cnext of None* ⇒ *ns*
       | *Some cnext* ⇒ *ns*[*cnext* := *VMTF-Node* (*stamp* (*ns*!*cnext*)) (*Some L*) (*get-next* (*ns*!*cnext*))]) *in*
  ns)
›

lemma *vmtf-notin-vmtf-cons*:
  assumes
    vmtf-ns: ‹*vmtf-ns-notin xs m ns*› **and**
    cnext: ‹*cnext* = *option-hd xs*› **and**
    L-xs: ‹*L* ∉ *set xs*›
  shows
    ‹*vmtf-ns-notin* (*L* # *xs*) (*Suc m*) (*vmtf-cons ns L cnext m*)›
proof (*cases xs*)
  case *Nil*
  then show *?thesis*
    using *assms* **by** (*auto simp*: *vmtf-ns-notin-def vmtf-cons-def elim*: *vmtf-nsE*)
next

case (*Cons L′ xs′*) **note** *xs* = *this*
show *?thesis*
using *assms* **unfolding** *xs vmtf-ns-notin-def xs vmtf-cons-def* **by** *auto*
qed

lemma *vmtf-cons*:
assumes
vmtf-ns: ‹*vmtf-ns xs m ns*› **and**
cnext: ‹*cnext* = *option-hd xs*› **and**
L-A: ‹*L* < *length ns*› **and**
L-xs: ‹*L* ∉ *set xs*›
shows
‹*vmtf-ns* (*L* # *xs*) (*Suc m*) (*vmtf-cons ns L cnext m*)›
proof (*cases xs*)
case *Nil*
then show *?thesis*
using *assms* **by** (*auto simp*: *vmtf-ns-single-iff vmtf-cons-def elim*: *vmtf-nsE*)
next
case (*Cons L′ xs′*) **note** *xs* = *this*
show *?thesis*
unfolding *xs*
apply (*rule vmtf-ns-Cons*[*OF vmtf-ns*[*unfolded xs*], *of* - ‹*Suc m*›])
subgoal using *L-A* **.**
subgoal using *L-xs* **unfolding** *xs* **by** *simp*
subgoal using *L-xs* **unfolding** *xs* **by** *simp*
subgoal by *simp*
subgoal using *cnext L-xs*
by (*auto simp*: *vmtf-cons-def Let-def xs*)
subgoal by *linarith*
done
qed

lemma *length-vmtf-cons*[*simp*]: ‹*length* (*vmtf-cons ns L n m*) = *length ns*›
by (*auto simp*: *vmtf-cons-def Let-def split*: *option.splits*)

lemma *wf-vmtf-get-prev*:
assumes *vmtf*: ‹(*ns*, *m*, *fst-As*, *lst-As*, *next-search*) ∈ *vmtf* 𝒜 *M*›
shows ‹*wf* {(*get-prev* (*ns* ! *the a*), *a*) |*a*. *a* ≠ *None* ∧ *the a* ∈ *atms-of* (ℒ*all* 𝒜)}› (**is** ‹*wf* *?R*›)
proof (*rule ccontr*)
assume ‹¬ *?thesis*›
then obtain *f* **where**
f: ‹(*f* (*Suc i*), *f i*) ∈ *?R*› **for** *i*
unfolding *wf-iff-no-infinite-down-chain* **by** *blast*

obtain *xs′ ys′* **where**
vmtf-ns: ‹*vmtf-ns* (*ys′* @ *xs′*) *m ns*› **and**
fst-As: ‹*fst-As* = *hd* (*ys′* @ *xs′*)› **and**
lst-As: ‹*lst-As* = *last* (*ys′* @ *xs′*)› **and**
next-search: ‹*next-search* = *option-hd xs′*› **and**
abs-vmtf: ‹*vmtf*-ℒ*all* 𝒜 *M* (*set xs′*, *set ys′*)› **and**
notin: ‹*vmtf-ns-notin* (*ys′* @ *xs′*) *m ns*› **and**
atm-A: ‹∀ *L*∈*atms-of* (ℒ*all* 𝒜). *L* < *length ns*›
using *vmtf* **unfolding** *vmtf-def* **by** *fast*
let *?f0* = ‹*the* (*f* 0)›
have *f-None*: ‹*f i* ≠ *None*› **for** *i*
using *f*[*of i*] **by** *fast*

```
  have f-Suc : ‹f (Suc n) = get-prev (ns ! the (f n))› for n
    using f[of n] by auto
  have f0-length: ‹?f0 < length ns›
    using f[of 0] atm-A
    by auto
  have f0-in:  ‹?f0 ∈ set (ys' @ xs')›
    apply (rule ccontr)
    using notin f-Suc[of 0] f0-length unfolding vmtf-ns-notin-def
    by (auto simp: f-None)
  then obtain i0 where
    i0: ‹(ys' @ xs') ! i0 = ?f0› ‹i0 < length (ys' @ xs')›
    by (meson in-set-conv-nth)
  define zs where ‹zs = ys' @ xs'›
  have H: ‹ys' @ xs' = take m (ys' @ xs') @ [(ys' @ xs') ! m, (ys' @ xs') ! (m+1)] @
     drop (m+2) (ys' @ xs')›
    if ‹m + 1 < length (ys' @ xs')›
    for m
    using that
    unfolding zs-def[symmetric]
    apply −
    apply (subst id-take-nth-drop[of m])
    by (auto simp: take-Suc-conv-app-nth Cons-nth-drop-Suc  simp del: append-take-drop-id)

  have ‹the (f n) = (ys' @ xs') ! (i0 − n) ∧ i0 − n ≥ 0 ∧ n ≤ i0› for n
  proof (induction n)
    case 0
    then show ?case using i0 by simp
  next
    case (Suc n')
    have i0-le: ‹n' < i0›
    proof (rule ccontr)
      assume ‹¬ ?thesis›
      then have ‹i0 = n'›
        using Suc by auto
      then have ‹ys' @ xs' = [the (f n')] @ tl (ys' @ xs')›
        using Suc f0-in
        by (cases ‹ys' @ xs'›) auto
      then show False
        using vmtf-ns-hd-prev[of ‹the (f n')› ‹tl (ys' @ xs')› m ns] vmtf-ns
         f-Suc[of n'] by (auto simp: f-None)
    qed
    have get-prev: ‹get-prev (ns ! ((ys' @ xs') ! (i0 − (n' +1) + 1))) =
        Some ((ys' @ xs') ! ((i0 − (n' + 1))))›
      apply (rule vmtf-ns-last-mid-get-prev[of ‹take (i0 − (n' +1)) (ys' @ xs')› - -
        ‹drop ((i0 − (n' + 1)) + 2) (ys' @ xs')› m])
      apply (subst H[symmetric])
      subgoal using i0-le i0 by auto
      subgoal using vmtf-ns by simp
      done
    then show ?case
      using f-Suc[of n'] Suc i0-le by auto
  qed
  from this[of ‹Suc i0›] show False
    by auto
qed
```

fun *update-stamp* **where**
  ‹*update-stamp xs n a* = *xs*[*a* := *VMTF-Node n* (*get-prev* (*xs*!*a*)) (*get-next* (*xs*!*a*))]›

definition *vmtf-rescale* :: ‹*vmtf* ⇒ *vmtf nres*› **where**
‹*vmtf-rescale* = (λ(*ns*, *m*, *fst-As*, *lst-As* :: *nat*, *next-search*). *do* {
  (*ns*, *m*, -) ← *WHILE*$_T$$^{\lambda\text{-}.\ True}$
    (λ(*ns*, *n*, *lst-As*). *lst-As* ≠*None*)
    (λ(*ns*, *n*, *a*). *do* {
      ASSERT(*a* ≠ *None*);
      ASSERT(*n*+*1* ≤ *unat32-max*);
      ASSERT(*the a* < *length ns*);
      RETURN (*update-stamp ns n* (*the a*), *n*+*1*, *get-prev* (*ns* ! *the a*))
    })
    (*ns*, *0*, *Some lst-As*);
  RETURN ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*))
  })
›

lemma *vmtf-rescale-vmtf*:
  assumes *vmtf*: ‹*vm* ∈ *vmtf* *A* *M*› **and**
    nempty: ‹*isasat-input-nempty* *A*› **and**
    bounded: ‹*isasat-input-bounded* *A*›
  shows
    ‹*vmtf-rescale vm* ≤ *SPEC* (λ*vm*. *vm* ∈ *vmtf* *A* *M* ∧ *fst* (*snd vm*) ≤ *unat32-max*)›
    (**is** ‹*?A* ≤ *?R*›)
proof −
  obtain *ns m fst-As lst-As next-search* **where**
    vm: ‹*vm* = ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*))›
    by (*cases vm*) *auto*

  obtain *xs′ ys′* **where**
    vmtf-ns: ‹*vmtf-ns* (*ys′* @ *xs′*) *m ns*› **and**
    fst-As: ‹*fst-As* = *hd* (*ys′* @ *xs′*)› **and**
    lst-As: ‹*lst-As* = *last* (*ys′* @ *xs′*)› **and**
    next-search: ‹*next-search* = *option-hd xs′*› **and**
    abs-vmtf: ‹*vmtf*-$\mathcal{L}_{all}$ *A* *M* (*set xs′*, *set ys′*)› **and**
    notin: ‹*vmtf-ns-notin* (*ys′* @ *xs′*) *m ns*› **and**
    atm-A: ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ *A*). *L* < *length ns*› **and**
    in-lall: ‹∀ *L*∈*set* (*ys′* @ *xs′*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ *A*)›
    using *vmtf* **unfolding** *vmtf-def vm* **by** *fast*
  have [*dest*]: ‹*ys′* = [] ⟹ *xs′* = [] ⟹ *False*› **and**
    [*simp*]: ‹*ys′* = [] ⟶ *xs′* ≠ []›
    using *abs-vmtf nempty* **unfolding** *vmtf*-$\mathcal{L}_{all}$*-def*
    by (*auto simp*: *atms-of*-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$)
  have *1*: ‹*RES* (*vmtf* *A* *M*) = *do* {
    a ← *RETURN* ();
    RES (*vmtf* *A* *M*)
    }›
    by *auto*
  define *zs* **where** ‹*zs* ≡ *ys′* @ *xs′*›

  define *I′* **where**
    ‹*I′* ≡ λ(*ns′*, *n*::*nat*, *lst*::*nat option*).
        map get-prev ns = *map get-prev ns′* ∧
        map get-next ns = *map get-next ns′* ∧

```
    (∀ i<n. stamp (ns' ! (rev zs ! i)) = i) ∧
    (lst ≠ None ⟶ n < length (zs) ∧ the lst = zs ! (length zs − Suc n)) ∧
    (lst = None ⟶ n = length zs) ∧
      n ≤ length zs›
have [simp]: ‹zs ≠ []›
  unfolding zs-def by auto
have I'0: ‹I' (ns, 0, Some lst-As)›
  using vmtf lst-As unfolding I'-def vm zs-def[symmetric] by (auto simp: last-conv-nth)


have lits: ‹literals-are-in-ℒ_in 𝒜 (Pos `# mset zs)› and
  dist: ‹distinct zs›
  using abs-vmtf vmtf-ns-distinct[OF vmtf-ns] unfolding vmtf-def zs-def
    vmtf-ℒ_all-def
  by (auto simp: literals-are-in-ℒ_in-alt-def inj-on-def)
have dist: ‹distinct-mset (Pos `# mset zs)›
  by (subst distinct-image-mset-inj)
    (use dist in ‹auto simp: inj-on-def›)
have tauto: ‹¬ tautology (poss (mset zs))›
  by (auto simp: tautology-decomp)

have length-zs-le: ‹length zs < unat32-max› using vmtf-ns-distinct[OF vmtf-ns]
    using simple-clss-size-upper-div2[OF bounded lits dist tauto]
    by (auto simp: unat32-max-def)

have ‹wf {(a, b). (a, b) ∈ {(get-prev (ns ! the a), a) |a. a ≠ None ∧ the a ∈ atms-of (ℒ_all 𝒜)}}›
  by (rule wf-subset[OF wf-vmtf-get-prev[OF vmtf[unfolded vm]]]) auto
from wf-snd-wf-pair[OF wf-snd-wf-pair[OF this]]
have wf: ‹wf {((-, -, a), (-, -, b)). (a, b) ∈ {(get-prev (ns ! the a), a) |a. a ≠ None ∧
    the a ∈ atms-of (ℒ_all 𝒜)}}›
  by (rule wf-subset) auto
have zs-lall: ‹zs ! (length zs − Suc n) ∈ atms-of (ℒ_all 𝒜)› for n
  using abs-vmtf nth-mem[of ‹length zs − Suc n› zs] unfolding zs-def vmtf-ℒ_all-def
  by auto
then have zs-le-ns[simp]: ‹zs ! (length zs − Suc n) < length ns› for n
  using atm-A by auto
have loop-body: ‹(case s' of
    (ns, n, a) ⇒ do {
      ASSERT (a ≠ None);
      ASSERT (n + 1 ≤ unat32-max);
      ASSERT(the a < length ns);
      RETURN (update-stamp ns n (the a), n + 1, get-prev (ns ! the a))
    })
    ≤ SPEC
     (λs'a. True ∧
          I' s'a ∧
          (s'a, s')
          ∈ {((-, -, a), -, -, b).
            (a, b)
            ∈ {(get-prev (ns ! the a), a) |a.
                a ≠ None ∧ the a ∈ atms-of (ℒ_all 𝒜)}})›
  if
    I': ‹I' s'› and
    cond: ‹case s' of (ns, n, lst-As) ⇒ lst-As ≠ None›
  for s'
proof −
```

obtain *ns′ n′ a′* **where** *s′*: ‹*s′ = (ns′, n′ , a′)*›
by (*cases s′*)
have
a[*simp*]: ‹*a′ = Some* (*zs* ! (*length zs − Suc n′*))› **and**
eq-prev: ‹*map get-prev ns = map get-prev ns′*› **and**
eq-next: ‹*map get-next ns = map get-next ns′*› **and**
eq-stamps: ‹⋀*i*. *i*<*n′* ⟹ *stamp* (*ns′* ! (*rev zs* ! *i*)) = *i*› **and**
n′-le: ‹*n′ < length zs*›
using *I′ cond* **unfolding** *I′-def prod.simps s′*
by *auto*
have [*simp*]: ‹*length ns′ = length ns*›
using *arg-cong*[*OF eq-prev*, *of length*] **by** *auto*
have *vmtf-as*: ‹*vmtf-ns*
(*take* (*length zs − (n′ + 1)*) *zs* @
zs ! (*length zs − (n′ + 1)*) #
drop (*Suc* (*length zs − (n′ + 1)*)) *zs*)
m ns›
apply (*subst Cons-nth-drop-Suc*)
subgoal by *auto*
apply (*subst append-take-drop-id*)
using *vmtf-ns* **unfolding** *zs-def*[*symmetric*] **.**

have ‹*get-prev* (*ns′* ! *the a′*) ≠ *None* ⟶
n′ + 1 < length zs ∧
the (*get-prev* (*ns′* ! *the a′*)) = *zs* ! (*length zs − Suc* (*n′ + 1*))›
using *n′-le vmtf-ns arg-cong*[*OF eq-prev*, *of* ‹λ*xs*. *xs* ! (*zs* ! (*length zs − Suc n′*))›]
vmtf-ns-last-mid-get-prev-option-last[*OF vmtf-as*]
by (*auto simp*: *last-conv-nth*)
moreover have ‹*map get-prev ns = map get-prev* (*update-stamp ns′ n′* (*the a′*))›
unfolding *update-stamp.simps*
apply (*subst map-update*)
apply (*subst list-update-id′*)
subgoal by *auto*
subgoal using *eq-prev* **.**
done
moreover have ‹*map get-next ns = map get-next* (*update-stamp ns′ n′* (*the a′*))›
unfolding *update-stamp.simps*
apply (*subst map-update*)
apply (*subst list-update-id′*)
subgoal by *auto*
subgoal using *eq-next* **.**
done
moreover have ‹*i*<*n′ + 1* ⟹ *stamp* (*update-stamp ns′ n′* (*the a′*) ! (*rev zs* ! *i*)) = *i*› **for** *i*
using *eq-stamps*[*of i*] *vmtf-ns-distinct*[*OF vmtf-ns*] *n′-le*
unfolding *zs-def*[*symmetric*]
by (*cases* ‹*i < n′*›)
(*auto simp*: *rev-nth nth-eq-iff-index-eq*)
moreover have ‹*n′ + 1 ≤ length zs*›
using *n′-le* **by** (*auto simp*: *Suc-le-eq*)
moreover have ‹*get-prev* (*ns′* ! *the a′*) = *None* ⟹ *n′ + 1 = length zs*›
using *n′-le vmtf-ns arg-cong*[*OF eq-prev*, *of* ‹λ*xs*. *xs* ! (*zs* ! (*length zs − Suc n′*))›]
vmtf-ns-last-mid-get-prev-option-last[*OF vmtf-as*]
by *auto*
ultimately have *I′-f*: ‹*I′* (*update-stamp ns′ n′* (*the a′*), *n′ + 1*, *get-prev* (*ns′* ! *the a′*))›
using *cond n′-le* **unfolding** *I′-def prod.simps s′*
by *simp*

```
  show ?thesis
    unfolding s' prod.case
    apply refine-vcg
    subgoal using cond by auto
    subgoal using length-zs-le n'-le by auto
    subgoal by auto
    subgoal by fast
    subgoal by (rule I'-f)
    subgoal
      using arg-cong[OF eq-prev, of ‹λxs. xs ! (zs ! (length zs − Suc n'))›] zs-lall
      by auto
    done
qed
have loop-final: ‹s ∈ {x. (case x of
            (ns, m, uua-) ⇒
              RETURN ((ns, m, fst-As, lst-As, next-search)))
            ≤ ?R}›
  if
    ‹True› and
    ‹I' s› and
    ‹¬ (case s of (ns, n, lst-As) ⇒ lst-As ≠ None)›
  for s
proof −
  obtain ns' n' a' where s: ‹s = (ns', n' , a')›
    by (cases s)
  have
    [simp]:‹a' = None› and
    eq-prev: ‹map get-prev ns = map get-prev ns'› and
    eq-next: ‹map get-next ns = map get-next ns'› and
    stamp: ‹∀ i<n'. stamp (ns' ! (rev zs ! i)) = i› and
    [simp]: ‹n' = length zs›
    using that unfolding I'-def s prod.case by auto
  have [simp]: ‹length ns' = length ns›
    using arg-cong[OF eq-prev, of length] by auto
  have [simp]: ‹map ((!) (map stamp ns')) (rev zs) = [0..<length zs]›
    apply (subst list-eq-iff-nth-eq, intro conjI)
    subgoal by auto
    subgoal using stamp by (auto simp: rev-nth)
    done
  then have stamps-zs[simp]: ‹map ((!) (map stamp ns')) zs = rev [0..<length zs]›
      unfolding rev-map[symmetric]
      using rev-swap by blast

  have ‹sorted (map ((!) (map stamp ns')) (rev zs))›
    by simp
  moreover have ‹distinct (map ((!) (map stamp ns')) zs)›
    by simp
  moreover have ‹∀ a∈set zs. get-prev (ns' ! a) = get-prev (ns ! a)›
    using eq-prev map-eq-nth-eq by fastforce
  moreover have ‹∀ a∈set zs. get-next (ns' ! a) = get-next (ns ! a)›
    using eq-next map-eq-nth-eq by fastforce
  moreover have ‹∀ a∈set zs. stamp (ns' ! a) = map stamp ns' ! a›
    using vmtf-ns vmtf-ns-le-length zs-def by auto
  moreover have ‹length ns ≤ length ns'›
   by simp
```

```
  moreover have ‹∀ a∈set zs. a < length (map stamp ns')›
    using vmtf-ns vmtf-ns-le-length zs-def by auto
  moreover have ‹∀ a∈set zs. map stamp ns' ! a < n'›
  proof
    fix a
    assume ‹a ∈ set zs›
    then have ‹map stamp ns' ! a ∈ set (map ((!) (map stamp ns')) zs)›
      by (metis in-set-conv-nth length-map nth-map)
    then show ‹map stamp ns' ! a < n'›
      unfolding stamps-zs by simp
  qed
  ultimately have ‹vmtf-ns zs n' ns'›
    using vmtf-ns-rescale[OF vmtf-ns, of ‹map stamp ns'› ns', unfolded zs-def[symmetric]]
    by fast
  moreover have ‹vmtf-ns-notin zs (length zs) ns'›
    using notin map-eq-nth-eq[OF eq-prev] map-eq-nth-eq[OF eq-next]
    unfolding zs-def[symmetric]
    by (auto simp: vmtf-ns-notin-def)
  ultimately have ‹(ns', n', fst-As, lst-As, next-search) ∈ vmtf 𝒜 M›
    using fst-As lst-As next-search abs-vmtf atm-A notin in-lall
    unfolding vmtf-def in-pair-collect-simp prod.case apply −
    apply (rule exI[of - xs'])
    apply (rule exI[of - ys'])
    unfolding zs-def[symmetric]
    by auto
  then show ?thesis
    using length-zs-le
    by (auto simp: s)
qed

have H: ‹WHILE_T^λ-. True (λ(ns, n, lst-As). lst-As ≠ None)
   (λ(ns, n, a). do {
        - ← ASSERT (a ≠ None);
        - ← ASSERT (n + 1 ≤ unat32-max);
        ASSERT(the a < length ns);
        RETURN (update-stamp ns n (the a), n + 1, get-prev (ns ! the a))
      })
   (ns, 0, Some lst-As)
  ≤ SPEC
     (λx. (case x of
           (ns, m, uua-) ⇒
             RETURN ((ns, m, fst-As, lst-As, next-search)))
          ≤ ?R)
›
apply (rule WHILEIT-rule-stronger-inv-RES[where I' = I' and
    R = ‹{((-, -, a), (-, -, b)). (a, b) ∈
      {(get-prev (ns ! the a), a) |a. a ≠ None ∧ the a ∈ atms-of (𝓛_all 𝒜)}}›])
subgoal
 by (rule wf)
subgoal by fast
subgoal by (rule I'0)
subgoal for s'
  by (rule loop-body)
subgoal for s
  by (rule loop-final)
done
```

show *?thesis*
unfolding *vmtf-rescale-def vm prod.case*
apply (*subst bind-rule-complete-RES*)
apply (*rule H*)
done
qed

definition *vmtf-flush*
:: ‹*nat multiset* ⇒ (*nat*,*nat*) *ann-lits* ⇒ *vmtf* ⇒ *nat set* ⇒ (*vmtf* × *nat set*) *nres*›
where
‹*vmtf-flush* $\mathcal{A}_{in}$ = (λ*M vm remove-int*. *SPEC* (λ*x*. (*fst x*) ∈ *vmtf* $\mathcal{A}_{in}$ *M* ∧ *snd x* = {}))›

definition *atoms-hash-rel* :: ‹*nat multiset* ⇒ (*bool list* × *nat set*) *set*› **where**
‹*atoms-hash-rel* $\mathcal{A}$ = {(*C*, *D*). (∀ *L* ∈ *D*. *L* < *length C*) ∧ (∀ *L* < *length C*. *C* ! *L* ⟷ *L* ∈ *D*) ∧
(∀ *L* ∈# $\mathcal{A}$. *L* < *length C*) ∧ *D* ⊆ *set-mset* $\mathcal{A}$}›

definition *distinct-hash-atoms-rel*
:: ‹'*v multiset* ⇒ (('*v list* × '*v set*) × '*v set*) *set*›
where
‹*distinct-hash-atoms-rel* $\mathcal{A}$ = {((*C*, *h*), *D*). *set C* = *D* ∧ *h* = *D* ∧ *distinct C* ∧ *D* ⊆ *set-mset* $\mathcal{A}$}›

definition *distinct-atoms-rel*
:: ‹*nat multiset* ⇒ ((*nat list* × *bool list*) × *nat set*) *set*›
where
‹*distinct-atoms-rel* $\mathcal{A}$ = (*Id* ×$_r$ *atoms-hash-rel* $\mathcal{A}$) *O distinct-hash-atoms-rel* $\mathcal{A}$›

lemma *distinct-atoms-rel-alt-def*:
‹*distinct-atoms-rel* $\mathcal{A}$ = {((*D'*, *C*), *D*). (∀ *L* ∈ *D*. *L* < *length C*) ∧ (∀ *L* < *length C*. *C* ! *L* ⟷ *L* ∈ *D*) ∧
(∀ *L* ∈# $\mathcal{A}$. *L* < *length C*) ∧ *set D'* = *D* ∧ *distinct D'* ∧ *set D'* ⊆ *set-mset* $\mathcal{A}$}›
unfolding *distinct-atoms-rel-def atoms-hash-rel-def distinct-hash-atoms-rel-def prod-rel-def*
apply *rule*
subgoal
by (*auto simp*: *mset-set-set*)
subgoal
by (*auto simp*: *mset-set-set*)
done

lemma *distinct-atoms-rel-empty-hash-iff*:
‹(([], *h*), {}) ∈ *distinct-atoms-rel* $\mathcal{A}$ ⟷ (∀ *L* ∈# $\mathcal{A}$. *L* < *length h*) ∧ (∀ *i*∈*set h*. *i* = *False*)›
unfolding *distinct-atoms-rel-alt-def all-set-conv-nth*
by *auto*

definition *atoms-hash-del-pre* **where**
‹*atoms-hash-del-pre i xs* = (*i* < *length xs*)›

definition *atoms-hash-del* **where**
‹*atoms-hash-del i xs* = *xs*[*i* := *False*]›

definition *vmtf-flush-int* :: ‹*nat multiset* ⇒ (*nat*,*nat*) *ann-lits* ⇒ - ⇒ - ⇒ - *nres*› **where**
‹*vmtf-flush-int* $\mathcal{A}_{in}$ = (λ*M vm* (*to-remove*, *h*). *do* {
ASSERT(∀ *x*∈*set to-remove*. *x* < *length* (*fst vm*));

ASSERT(*length to-remove* ≤ *unat32-max*);
to-remove′ ← *reorder-list vm to-remove*;
ASSERT(*length to-remove′* ≤ *unat32-max*);
vm ← (*if length to-remove′* + *fst* (*snd vm*) ≥ *unat64-max*
then vmtf-rescale vm else RETURN vm);
ASSERT(*length to-remove′* + *fst* (*snd vm*) ≤ *unat64-max*);
(-, *vm*, *h*) ← $WHILE_T$λ(*i*, *vm′*, *h*). *i* ≤ *length to-remove′* ∧ *fst* (*snd vm′*) = *i* + *fst* (*snd vm*) ∧ (*i* < *length to-remove*
(λ(*i*, *vm*, *h*). *i* < *length to-remove′*)
(λ(*i*, *vm*, *h*). *do* {
ASSERT(*i* < *length to-remove′*);
ASSERT(*to-remove′*!*i* ∈# $\mathcal{A}_{in}$);
ASSERT(*atoms-hash-del-pre* (*to-remove′*!*i*) *h*);
RETURN (*i*+1, *vmtf-en-dequeue M* (*to-remove′*!*i*) *vm*, *atoms-hash-del* (*to-remove′*!*i*) *h*)})
(*0*, *vm*, *h*);
RETURN (*vm*, (*emptied-list to-remove′*, *h*))
})›

lemma *vmtf-change-to-remove-order*:
assumes
vmtf: ‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*)) ∈ *vmtf* $\mathcal{A}_{in}$ *M*› **and**
CD-rem: ‹((*C*, *D*), *to-remove*) ∈ *distinct-atoms-rel* $\mathcal{A}_{in}$› **and**
nempty: ‹*isasat-input-nempty* $\mathcal{A}_{in}$› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}_{in}$› **and**
t: ‹*to-remove* ⊆ *set-mset* $\mathcal{A}_{in}$›
shows ‹*vmtf-flush-int* $\mathcal{A}_{in}$ *M* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*) (*C*, *D*)
≤ ⇓(*Id* ×$_r$ *distinct-atoms-rel* $\mathcal{A}_{in}$)
(*vmtf-flush* $\mathcal{A}_{in}$ *M* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*) *to-remove*)›
proof −
let *?vm* = ‹(*ns*, *m*, *fst-As*, *lst-As*, *next-search*)›
have *vmtf-flush-alt-def*: ‹*vmtf-flush* $\mathcal{A}_{in}$ *M ?vm to-remove* = *do* {
- ← *RETURN* ();
- ← *RETURN* ();
vm ← *SPEC* (λ*x*. (*fst x*) ∈ *vmtf* $\mathcal{A}_{in}$ *M* ∧ *snd x* = {});
RETURN (*vm*)
}›
unfolding *vmtf-flush-def* **by** (*auto simp*: *RES-RES-RETURN-RES RES-RETURN-RES vmtf*)

have *pre-sort*: ‹∀ *x*∈*set x1*. *x* < *length* ((*fst* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*)))›
if
‹(*C*, *D*) = (*x1*, *x2*)›
for *x1 x2 x1a x2a*
proof −
show *?thesis*
using *vmtf CD-rem that t* **apply** (*auto simp*: *vmtf-def vmtf-*$\mathcal{L}_{all}$*-def*
distinct-atoms-rel-alt-def)
by (*metis atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *in-mono*)
qed

have *length-le*: ‹*length x1a* ≤ *unat32-max*›
if
‹(*C*, *D*) = (*x1a*, *x2a*)› **and**
‹∀ *x*∈*set x1a*. *x* < *length* (*fst x1*)›
for *x1 x2 x1a x2a*
proof −

```
  have lits: ‹literals-are-in-ℒ_in 𝒜_in (Pos `# mset x1a)› and
    dist: ‹distinct x1a›
    using that vmtf CD-rem t unfolding vmtf-def
      vmtf-ℒ_all-def
    apply (auto simp: literals-are-in-ℒ_in-alt-def distinct-atoms-rel-alt-def inj-on-def)
    by (metis atms-of-ℒ_all-𝒜_in in-mono)
  have dist: ‹distinct-mset (Pos `# mset x1a)›
    by (subst distinct-image-mset-inj)
      (use dist in ‹auto simp: inj-on-def›)
  have tauto: ‹¬ tautology (poss (mset x1a))›
    by (auto simp: tautology-decomp)

  show ?thesis
    using simple-clss-size-upper-div2[OF bounded lits dist tauto]
    by (auto simp: unat32-max-def)
qed


have [refine0]:
   ‹reorder-list x1 x1a ≤ SPEC (λc. (c, ())) ∈
      {(c, c'). ((c, D), to-remove) ∈ distinct-atoms-rel 𝒜_in ∧ to-remove = set c ∧
        length C = length c})›
  (is ‹- ≤ SPEC(λ-. - ∈ ?reorder-list)›)
  if
    ‹x2 = (x1a, x2a)› and
    ‹((ns, m, fst-As, lst-As, next-search), C, D) = (x1, x2)›
  for x1 x2 x1a x2a
proof −
  show ?thesis
    using that assms by (force simp: reorder-list-def distinct-atoms-rel-alt-def
      dest: mset-eq-setD same-mset-distinct-iff mset-eq-length)
qed


have [refine0]: ‹(if unat64-max ≤ length to-remove' + fst (snd x1) then vmtf-rescale x1
    else RETURN x1)
    ≤ SPEC (λc. (c, ())) ∈
      {(vm ,vm'). unat64-max ≥ length to-remove' + fst (snd vm) ∧
        (vm) ∈ vmtf 𝒜_in M}) ›
  (is ‹- ≤ SPEC(λc. (c, ()) ∈ ?rescale)› is ‹- ≤ ?H›)
if
  ‹x2 = (x1a, x2a)› and
  ‹((ns, m, fst-As, lst-As, next-search), C, D) = (x1, x2)› and
  ‹∀ x∈set x1a. x < length (fst x1)› and
  ‹length x1a ≤ unat32-max› and
  ‹(to-remove', uu) ∈ ?reorder-list› and
  ‹length to-remove' ≤ unat32-max›
for x1 x2 x1a x2a to-remove' uu
proof −
  have ‹vmtf-rescale x1 ≤ ?H›
    apply (rule order-trans)
    apply (rule vmtf-rescale-vmtf[of - 𝒜_in M])
    subgoal using vmtf that by auto
    subgoal using nempty by fast
    subgoal using bounded by fast
    subgoal using that by (auto intro!: RES-refine simp: unat64-max-def unat32-max-def)
```

```
    done
  then show ?thesis
    using that vmtf
    by (auto intro!: RETURN-RES-refine)
qed


have loop-ref: ‹WHILE_T^λ(i, vm', h).                    i ≤ length to-remove' ∧ fst (snd vm') = i + fst (snd x1) ∧
      (λ(i, vm, h). i < length to-remove')
      (λ(i, vm, h). do {
            ASSERT (i < length to-remove');
            ASSERT(to-remove'!i ∈# A_in);
            ASSERT(atoms-hash-del-pre (to-remove'!i) h);
            RETURN
              (i + 1, vmtf-en-dequeue M (to-remove' ! i) vm,
              atoms-hash-del (to-remove'!i) h)
          })
      (0, x1, x2a)
      ≤ ⇓ {((i, vm::vmtf, h:: -), vm'). (vm, {}) = vm' ∧ (∀ i∈set h. i = False) ∧ i = length to-remove'
∧
            ((drop i to-remove', h), set(drop i to-remove')) ∈ distinct-atoms-rel A_in}
    (SPEC (λx. (fst x) ∈ vmtf A_in M ∧ snd x = {}))›
  if
    CD: ‹(C, D) = (x1a, x2a)› and
    x1: ‹(x1, u') ∈ ?rescale to-remove'›
    ‹(to-remove', u) ∈ ?reorder-list›
  for x1 x2 x1a x2a to-remove' u u' x1'
proof −
  define I where ‹I ≡ λ(i, vm'::vmtf, h::bool list).
              i ≤ length to-remove' ∧ fst (snd vm') = i + fst (snd x1) ∧
              (i < length to-remove' ⟶
                vmtf-en-dequeue-pre A_in ((M, to-remove' ! i), vm'))›
  define I' where ‹I' ≡ λ(i, vm::vmtf, h:: bool list).
    ((drop i to-remove', h), set(drop i to-remove')) ∈ distinct-atoms-rel A_in ∧
          vm ∈ vmtf A_in M›
  have [simp]:
    ‹x1a = C›
    ‹x2a = D› and
    rel: ‹((to-remove', D), to-remove) ∈ distinct-atoms-rel A_in› and
    to-rem: ‹to-remove = set to-remove'›
    using that by (auto simp: )
  have D: ‹set to-remove' = to-remove› and dist: ‹distinct to-remove'›
    using rel unfolding distinct-atoms-rel-alt-def by auto
  have in-lall: ‹to-remove' ! x1 ∈ atms-of (L_all A_in)› if le': ‹x1 < length to-remove'› for x1
    using vmtf to-rem nth-mem[OF le'] t by (auto simp: vmtf-def vmtf-L_all-def
      atms-of-L_all-A_in subsetD)
  have bound: ‹fst (snd x1) + 1 ≤ unat64-max› if ‹0 < length to-remove'›
     using rel vmtf to-rem that x1 by (cases to-remove') auto
  have I-init: ‹I (0, x1, x2a)› (is ?A)
    for x1a x2 x1aa x2aa
  proof −
    have ‹vmtf-en-dequeue-pre A_in ((M, to-remove' ! 0), x1)› if ‹0 < length to-remove'›
      apply (rule vmtf-vmtf-en-dequeue-pre-to-remove'[of -])
      using rel vmtf to-rem that x1 bound nempty in-lall[of 0] by (auto simp: )
    then show ?A
      unfolding I-def by auto
```

qed
have *I'-init*: ‹*I'* (*0*, *x1*, *x2a*)› (**is** *?B*)
for *x1a x2 x1aa x2aa*
proof −
show *?B*
using *rel to-rem CD-rem that*[*symmetric*] *vmtf t* **unfolding** *I'-def* **by** *auto*
qed
have *post-loop*: ‹*do* {
ASSERT (*x2* < *length to-remove'*);
ASSERT(*to-remove'* ! *x2* ∈# $\mathcal{A}_{in}$);
ASSERT(*atoms-hash-del-pre* (*to-remove'* ! *x2*) *x2a'*);
RETURN
(*x2* + *1*, *vmtf-en-dequeue M* (*to-remove'* ! *x2*) *x2aa*,
atoms-hash-del (*to-remove'*!*x2*) *x2a'*)
} ≤ *SPEC*
(λ*s'*. *I s'* ∧ *I' s'* ∧ (*s'*, *x1a*) ∈ *measure* (λ(*i*, *vm*, *h*). *Suc* (*length to-remove'*) − *i*))›
if
I: ‹*I x1a*› **and**
I': ‹*I' x1a*› **and**
‹*case x1a of* (*i*, *vm*, *h*) ⇒ *i* < *length to-remove'*› **and**
x1aa: ‹*x1aa* = (*x2aa*, *x2a'*)› ‹*x1a* = (*x2*, *x1aa*)›
for *s x2 x1a x2a x1a' x2a' x1aa x2aa*
proof −
let *?x2a'* = ‹*set* (*drop x2 to-remove'*)›
have *le*: ‹*x2* < *length to-remove'*› **and** *vm*: ‹(*x2aa*) ∈ *vmtf* $\mathcal{A}_{in}$ *M*› **and**
x2a': ‹*fst* (*snd x2aa*) = *x2* + *fst* (*snd x1*)›
using *that* **unfolding** *I-def I'-def* **by** (*auto simp*: *distinct-atoms-rel-alt-def*)
have *1*: ‹(*vmtf-en-dequeue M* (*to-remove'* ! *x2*) *x2aa*) ∈ *vmtf* $\mathcal{A}_{in}$ *M*›
by (*rule abs-vmtf-ns-bump-vmtf-en-dequeue'*[*OF vm in-lall*[*OF le*]])
(*use nempty* **in** *auto*)
have *3*: ‹*fst* (*snd x2aa*) = *fst* (*snd x1*) + *x2*›
using *I I' le dist that CD x2a'* **unfolding** *I-def I'-def x2a' x1aa*
by (*auto simp*: *distinct-atoms-rel-def in-set-drop-conv-nth I'-def*
nth-eq-iff-index-eq intro: *bex-geI*[*of* - ‹*Suc x2*›])
then have *4*: ‹*fst* (*snd* (*vmtf-en-dequeue M* (*to-remove'* ! *x2*) *x2aa*)) + *1* ≤ *unat64-max*›
if ‹*Suc x2* < *length to-remove'*›
using *x1 le that*
by (*cases x2aa*)
(*auto simp*: *vmtf-en-dequeue-def vmtf-enqueue-def vmtf-dequeue-def*
split: *option.splits*)
have *1*: ‹*vmtf-en-dequeue-pre* $\mathcal{A}_{in}$
((*M*, *to-remove'* ! *Suc x2*), *vmtf-en-dequeue M* (*to-remove'* ! *x2*) *x2aa*)›
if ‹*Suc x2* < *length to-remove'*›
by (*rule vmtf-vmtf-en-dequeue-pre-to-remove'*)
(*rule 1*, *rule in-lall*, *rule that*, *rule 4*[*OF that*], *rule nempty*)
have *3*: ‹(*vmtf-en-dequeue M* (*to-remove'* ! *x2*) *x2aa*) ∈ *vmtf* $\mathcal{A}_{in}$ *M*›
by (*rule abs-vmtf-ns-bump-vmtf-en-dequeue'*[*OF vm in-lall*[*OF le*]]) (*use nempty* **in** *auto*)
have *4*: ‹((*drop* (*Suc x2*) *to-remove'*, *atoms-hash-del* (*to-remove'* ! *x2*) *x2a'*),
set (*drop* (*Suc x2*) *to-remove'*))
∈ *distinct-atoms-rel* $\mathcal{A}_{in}$› **and**
3: ‹(*vmtf-en-dequeue M* (*to-remove'* ! *x2*) *x2aa*)
∈ *vmtf* $\mathcal{A}_{in}$ *M*›
using *3 I' le to-rem that* **unfolding** *I'-def distinct-atoms-rel-alt-def atoms-hash-del-def*
by (*auto simp*: *Cons-nth-drop-Suc*[*symmetric*] *intro*: *mset-le-add-mset-decr-left1*)

have *A*: ‹*to-remove'* ! *x2* ∈ *?x2a'*›

```
      using I I' le dist that x1aa unfolding I-def I'-def
      by (auto simp: distinct-atoms-rel-def in-set-drop-conv-nth I'-def
          nth-eq-iff-index-eq x2a' intro: bex-geI[of - ‹x2›])
    moreover have ‹I (Suc x2, vmtf-en-dequeue M (to-remove' ! x2) x2aa,
      atoms-hash-del (to-remove' ! x2) x2a')›
      using that 1 unfolding I-def
      by (cases x2aa)
        (auto simp: vmtf-en-dequeue-def vmtf-enqueue-def vmtf-dequeue-def
        split: option.splits)
    moreover have ‹length to-remove' − x2 < Suc (length to-remove') − x2›
      using le by auto
    moreover have ‹I' (Suc x2, vmtf-en-dequeue M (to-remove' ! x2) x2aa,
      atoms-hash-del (to-remove' ! x2) x2a')›
      using that 3 4 I' unfolding I'-def
      by auto
    moreover have ‹atoms-hash-del-pre (to-remove' ! x2) x2a'›
      unfolding atoms-hash-del-pre-def
      using that le A unfolding I-def I'-def by (auto simp: distinct-atoms-rel-alt-def)
    ultimately show ?thesis
      using that in-lall[OF le]
      by (auto simp: atms-of-ℒ_all-𝒜_in)
  qed
  have [simp]: ‹∀ L<length ba. ¬ ba ! L ⟹ True ∉ set ba› for ba
    by (simp add: in-set-conv-nth)
  have post-rel: ‹RETURN s
     ≤ ⇓ {((i, vm, h), vm').
          (vm, {}) = vm' ∧
          (∀ i∈set h. i = False) ∧
          i = length to-remove' ∧
          ((drop i to-remove', h), set (drop i to-remove'))
    ∈ distinct-atoms-rel 𝒜_in}
    (SPEC (λx. (fst x) ∈ vmtf 𝒜_in M ∧ snd x = {}))›
      if
       ‹¬ (case s of (i, vm, h) ⇒ i < length to-remove')› and
       ‹I s› and
       ‹I' s›
     for s
  proof −
    obtain i vm h where s: ‹s = (i, vm, h)› by (cases s)
    have [simp]: ‹i = length (to-remove')› and [iff]: ‹True ∉ set h› and
      [simp]: ‹(([], h), {}) ∈ distinct-atoms-rel 𝒜_in›
        ‹(vm) ∈ vmtf 𝒜_in M›
      using that unfolding s I-def I'-def by (auto simp: distinct-atoms-rel-empty-hash-iff)
    show ?thesis
      unfolding s
      by (rule RETURN-RES-refine) auto
  qed

  show ?thesis
    unfolding I-def[symmetric]
    apply (refine-rcg
     WHILEIT-rule-stronger-inv-RES'[where R=‹measure (λ(i, vm::vmtf, h). Suc (length to-remove')
−i)› and
          I'=‹I'›])
    subgoal by auto
    subgoal by (rule I-init)
```

subgoal by (*rule I'-init*)
subgoal for *x1'' x2'' x1a'' x2a''* **by** (*rule post-loop*)
subgoal by (*rule post-rel*)
done
qed

show *?thesis*
unfolding *vmtf-flush-int-def vmtf-flush-alt-def*
apply (*refine-rcg*)
subgoal by (*rule pre-sort*)
subgoal by (*rule length-le*)
apply (*assumption+*)[*2*]
subgoal by *auto*
subgoal by *auto*
apply (*assumption+*)[*5*]
subgoal by *auto*
apply (*assumption+*)[*3*]
subgoal by *auto*
apply (*rule loop-ref*; *assumption*)
subgoal by (*auto simp*: *emptied-list-def*)
done
qed

lemma *vmtf-change-to-remove-order'*:
‹(*uncurry2* (*vmtf-flush-int* $\mathcal{A}_{in}$), *uncurry2* (*vmtf-flush* $\mathcal{A}_{in}$)) ∈
[λ((*M*, *vm*), *to-r*). *vm* ∈ *vmtf* $\mathcal{A}_{in}$ *M* ∧ *isasat-input-bounded* $\mathcal{A}_{in}$ ∧ *isasat-input-nempty* $\mathcal{A}_{in}$ ∧ *to-r* ⊆ *set-mset* $\mathcal{A}_{in}$]$_f$
Id ×$_f$ *Id* ×$_f$ *distinct-atoms-rel* $\mathcal{A}_{in}$ → ⟨(*Id* ×$_r$ *distinct-atoms-rel* $\mathcal{A}_{in}$)⟩ *nres-rel*›
by (*intro frefI nres-relI*)
(*use* **in** ‹*auto intro*!: *vmtf-change-to-remove-order*›)

definition (**in** −) *isa-vmtf-unset* :: ‹*nat* ⇒ *vmtf* ⇒ *vmtf*› **where**
‹*isa-vmtf-unset* = (λ*L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*).
(*if next-search* = *None* ∨ *stamp* (*ns* ! (*the next-search*)) < *stamp* (*ns* ! *L*)
then ((*ns*, *m*, *fst-As*, *lst-As*, *Some L*))
else ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*))))›

definition *vmtf-unset-pre* **where**
‹*vmtf-unset-pre* = (λ*L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*).
L < *length ns* ∧ (*next-search* ≠ *None* ⟶ *the next-search* < *length ns*))›

lemma *vmtf-unset-pre-vmtf*:
assumes
‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*)) ∈ *vmtf* $\mathcal{A}$ *M*› **and**
‹*L* ∈# $\mathcal{A}$›
shows ‹*vmtf-unset-pre L* ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*))›
using *assms*
by (*auto simp*: *vmtf-def vmtf-unset-pre-def atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$)

definition *vmtf-heur-fst* **where**
‹*vmtf-heur-fst* = (λ(-, -, *a*, -). *a*)›

5.4.4 Hash for lists

definition *atms-hash-insert-pre* :: ‹*nat* ⇒ *nat list* × *bool list* ⇒ *bool*› **where**
‹*atms-hash-insert-pre i* = (λ(*n*, *xs*). *i* < *length xs* ∧ (¬*xs*!*i* ⟶ *length n* < *2* + *unat32-max div 2*))›

definition *atoms-hash-insert* :: ‹*nat* ⇒ *nat list* × *bool list* ⇒ (*nat list* × *bool list*)› **where**
‹*atoms-hash-insert i* = (λ(*n*, *xs*). *if xs* ! *i then* (*n*, *xs*) *else* (*n* @ [*i*], *xs*[*i* := *True*]))›

end
theory *LBD*
imports *IsaSAT-Literals*
begin

Chapter 6

LBD

LBD (literal block distance) or glue is a measure of usefulness of clauses: It is the number of different levels involved in a clause. This measure has been introduced by Glucose in 2009 (Audemart and Simon).

LBD has also another advantage, explaining why we implemented it even before working on restarts: It can speed the conflict minimisation. Indeed a literal might be redundant only if there is a literal of the same level in the conflict.

The LBD data structure is well-suited to do so: We mark every level that appears in the conflict in a hash-table like data structure.

Remark that we combine the LBD with a MTF scheme.

6.1 Types and relations

type-synonym *lbd* = ‹*bool list*›
type-synonym *lbd-ref* = ‹*nat list* × *nat* × *nat*›

Beside the actual "lookup" table, we also keep the highest level marked so far to unmark all levels faster (but we currently don't save the LBD and have to iterate over the data structure). We also handle growing of the structure by hand instead of using a proper hash-table.

definition *lbd-ref* :: ‹(*lbd-ref* × *lbd*) *set*› **where**
‹*lbd-ref* = {((*lbd*, *stamp*, *m*), *lbd'*).
length lbd' ≤ *Suc* (*Suc* (*unat32-max div 2*)) ∧
m = *length* (*filter id lbd'*) ∧
stamp > *0* ∧
length lbd = *length lbd'* ∧
(∀ *v* ∈ *set lbd*. *v* ≤ *stamp*) ∧
(∀ *i* < *length lbd'*. *lbd'* ! *i* ⟷ *lbd* ! *i* = *stamp*)
}›

6.2 Testing if a level is marked

definition *level-in-lbd* :: ‹*nat* ⇒ *lbd* ⇒ *bool*› **where**
‹*level-in-lbd i* = (λ*lbd*. *i* < *length lbd* ∧ *lbd*!*i*)›

definition *level-in-lbd-ref* :: ‹*nat* ⇒ *lbd-ref* ⇒ *bool*› **where**
‹*level-in-lbd-ref* = (λ*i* (*lbd*, *stamp*, -). *i* < *length-uint32-nat lbd* ∧ *lbd*!*i* = *stamp*)›

lemma *level-in-lbd-ref-level-in-lbd*:

‹(uncurry (RETURN oo level-in-lbd-ref), uncurry (RETURN oo level-in-lbd)) ∈
nat-rel ×$_r$ lbd-ref →$_f$ ⟨bool-rel⟩nres-rel›
by (intro frefI nres-relI) (auto simp: level-in-lbd-ref-def level-in-lbd-def lbd-ref-def)

6.3 Marking more levels

definition list-grow **where**
‹list-grow xs n x = xs @ replicate (n − length xs) x›

definition lbd-write :: ‹lbd ⇒ nat ⇒ lbd› **where**
‹lbd-write = (λlbd i.
(if i < length lbd then (lbd[i := True])
else ((list-grow lbd (i + 1) False)[i := True])))›

definition lbd-ref-write :: ‹lbd-ref ⇒ nat ⇒ lbd-ref nres› **where**
‹lbd-ref-write = (λ(lbd, stamp, n) i. do {
ASSERT(length lbd ≤ unat32-max ∧ n + 1 ≤ unat32-max);
(if i < length-uint32-nat lbd then
let n = if lbd ! i = stamp then n else n+1 in
RETURN (lbd[i := stamp], stamp, n)
else do {
ASSERT(i + 1 ≤ unat32-max);
RETURN ((list-grow lbd (i + 1) 0)[i := stamp], stamp, n + 1)
})
})›

lemma length-list-grow[simp]:
‹length (list-grow xs n a) = max (length xs) n›
by (auto simp: list-grow-def)

lemma list-update-append2: ‹i ≥ length xs ⟹ (xs @ ys)[i := x] = xs @ ys[i − length xs := x]›
by (induction xs arbitrary: i) (auto split: nat.splits)

lemma lbd-ref-write-lbd-write:
‹(uncurry (lbd-ref-write), uncurry (RETURN oo lbd-write)) ∈
[λ(lbd, i). i ≤ Suc (unat32-max div 2)]$_f$
lbd-ref ×$_f$ nat-rel → ⟨lbd-ref⟩nres-rel›
unfolding lbd-ref-write-def lbd-write-def
by (intro frefI nres-relI)
(auto simp: level-in-lbd-ref-def level-in-lbd-def lbd-ref-def list-grow-def
nth-append unat32-max-def length-filter-update-true list-update-append2
length-filter-update-false
intro!: ASSERT-leI le-trans[OF length-filter-le]
elim!: in-set-upd-cases)

6.4 Cleaning the marked levels

definition lbd-emtpy-inv :: ‹nat list ⇒ nat list × nat ⇒ bool› **where**
‹lbd-emtpy-inv ys = (λ(xs, i). (∀ j < i. xs ! j = 0) ∧ i ≤ length xs ∧ length ys = length xs)›

definition lbd-empty-loop-ref **where**
‹lbd-empty-loop-ref = (λ(xs, -, -). do {
(xs, i) ←
WHILE$_T$$^{lbd\text{-}emtpy\text{-}inv\ xs}$

```
      (λ(xs, i). i < length xs)
      (λ(xs, i). do {
        ASSERT(i < length xs);
        ASSERT(i + 1 < unat32-max);
        RETURN (xs[i := 0], i + 1)})
      (xs, 0);
    RETURN (xs, 1, 0)
  })›

definition lbd-empty where
  ‹lbd-empty xs = RETURN (replicate (length xs) False)›

lemma lbd-empty-loop-ref:
  assumes ‹((xs, m, n), ys) ∈ lbd-ref›
  shows
    ‹lbd-empty-loop-ref (xs, m, n) ≤ ⇓ lbd-ref (RETURN (replicate (length ys) False))›
proof −
  have le-xs: ‹length xs ≤ unat32-max div 2 + 2›
    ‹length ys = length xs›
    using assms by (auto simp: lbd-ref-def)
  have [iff]: ‹(∀ j. ¬ j < (b :: nat)) ⟷ b = 0› for b
    by auto
  have init: ‹lbd-emtpy-inv xs (xs, 0)›
    unfolding lbd-emtpy-inv-def
    by (auto simp: lbd-ref-def)
  have lbd-remove: ‹lbd-emtpy-inv xs (a[b := 0], b + 1)›
    if
      inv: ‹lbd-emtpy-inv xs s› and
      ‹case s of (ys, i) ⇒ length ys = length xs› and
      cond: ‹case s of (xs, i) ⇒ i < length xs› and
      s: ‹s = (a, b)› and
      b-le: ‹b < length a›
    for s a b
  proof −
    have 1: ‹a[b := 0] ! j = 0› if ‹j<b› for j
      using inv that unfolding lbd-emtpy-inv-def s
      by auto
    have ‹a[b := 0] ! j = 0› if ‹j<b + 1› for j
      using 1[of j] that cond b-le by (cases ‹j = b›) auto
    then show ?thesis
      using cond inv unfolding lbd-emtpy-inv-def s by auto
  qed
  have lbd-final: ‹((a, 1, 0), replicate (length ys) False) ∈ lbd-ref›
    if
      lbd: ‹lbd-emtpy-inv xs s› and
      I': ‹case s of (ys, i) ⇒ length ys = length xs› and
      cond: ‹¬ (case s of (xs, i) ⇒ i < length xs)› and
      s: ‹s = (a, b)›
      for s a b
  proof −
    have 1: ‹a[b := 0] ! j = 0› if ‹j<b› for j
      using lbd that unfolding lbd-emtpy-inv-def s
      by auto
    have [simp]: ‹length a = length xs›
      using I' unfolding s by auto
    have [dest]: ‹i < length xs ⟹ a ! i = 0› for i
```

```
    using 1[of i] lbd cond unfolding s lbd-emtpy-inv-def by (cases ‹i < Suc m›) auto

  have [simp]: ‹a = replicate (length xs) 0›
    unfolding list-eq-iff-nth-eq
    apply (intro conjI)
    subgoal by simp
    subgoal by auto
    done
  show ?thesis
    using le-xs by (auto simp: lbd-ref-def)
 qed
 show ?thesis
  unfolding lbd-empty-loop-ref-def conc-fun-RETURN
  apply clarify
  apply (refine-vcg WHILEIT-rule-stronger-inv[where R = ‹measure (λ(xs, i). length xs − i)› and
    I' = ‹λ(ys, i). length ys = length xs›])
  subgoal by auto
  subgoal by (rule init)
  subgoal by auto
  subgoal by auto
  subgoal using assms by (auto simp: lbd-ref-def lbd-emtpy-inv-def unat32-max-def)
  subgoal by (rule lbd-remove)
  subgoal by auto
  subgoal by (auto simp: lbd-emtpy-inv-def)
  subgoal by (rule lbd-final)
  done
qed

definition lbd-empty-cheap-ref where
 ‹lbd-empty-cheap-ref = (λ(xs, stamp, n). RETURN (xs, stamp + 1, 0))›

lemma lbd-empty-cheap-ref:
 assumes ‹((xs, m, n), ys) ∈ lbd-ref›
 shows
  ‹lbd-empty-cheap-ref (xs, m, n) ≤ ⇓ lbd-ref (RETURN (replicate (length ys) False))›
 using assms unfolding lbd-empty-cheap-ref-def lbd-ref-def
 by (auto simp: filter-empty-conv all-set-conv-nth in-set-conv-nth)

definition lbd-empty-ref :: ‹lbd-ref ⇒ lbd-ref nres› where
 ‹lbd-empty-ref = (λ(xs, m, n). if m = unat32-max then lbd-empty-loop-ref (xs,m,n)
   else lbd-empty-cheap-ref (xs, m, n)) ›

lemma lbd-empty-ref:
 assumes ‹((xs, m, n), ys) ∈ lbd-ref›
 shows
  ‹lbd-empty-ref (xs, m, n) ≤ ⇓ lbd-ref (RETURN (replicate (length ys) False))›
 using lbd-empty-cheap-ref[OF assms] lbd-empty-loop-ref[OF assms]
 by (auto simp: lbd-empty-ref-def)

lemma lbd-empty-ref-lbd-empty:
 ‹(lbd-empty-ref, lbd-empty) ∈ lbd-ref →f ⟨lbd-ref⟩nres-rel›
 apply (intro frefI nres-relI)
 apply clarify
 subgoal for lbd m lbd'
  using lbd-empty-ref[of lbd m]
  by (auto simp: lbd-empty-def)
```

done

definition (**in** −)*empty-lbd* :: ‹*lbd*› **where**
‹*empty-lbd* = (*replicate 32 False*)›

definition *empty-lbd-ref* :: ‹*lbd-ref*› **where**
‹*empty-lbd-ref* = (*replicate 32 0*, *1*, *0*)›

lemma *empty-lbd-ref-empty-lbd*:
‹(λ-. (*RETURN empty-lbd-ref*), λ-. (*RETURN empty-lbd*)) ∈ *unit-rel* →$_f$ ⟨*lbd-ref*⟩*nres-rel*›
by (*intro frefI nres-relI*) (*auto simp*: *empty-lbd-def lbd-ref-def empty-lbd-ref-def*
unat32-max-def nth-Cons split: *nat.splits*)

6.5 Extracting the LBD

We do not prove correctness of our algorithm, as we don't care about the actual returned value (for correctness).

definition *get-LBD* :: ‹*lbd* ⇒ *nat nres*› **where**
‹*get-LBD lbd* = *SPEC*(λ-. *True*)›

definition *get-LBD-ref* :: ‹*lbd-ref* ⇒ *nat nres*› **where**
‹*get-LBD-ref* = (λ(*xs*, *m*, *n*). *RETURN n*)›

lemma *get-LBD-ref*:
‹((*lbd*, *m*), *lbd'*) ∈ *lbd-ref* ⟹ *get-LBD-ref* (*lbd*, *m*) ≤ ⇓ *nat-rel* (*get-LBD lbd'*)›
unfolding *get-LBD-ref-def get-LBD-def*
by (*auto split*:*prod.splits*)

lemma *get-LBD-ref-get-LBD*:
‹(*get-LBD-ref*, *get-LBD*) ∈ *lbd-ref* →$_f$ ⟨*nat-rel*⟩*nres-rel*›
apply (*intro frefI nres-relI*)
apply *clarify*
subgoal for *lbd m n lbd'*
using *get-LBD-ref*[*of lbd*]
by (*auto simp*: *lbd-empty-def lbd-ref-def*)
done

end
theory *LBD-LLVM*
imports *LBD IsaSAT-Literals-LLVM*
begin

type-synonym *'a larray64* = ‹(*'a*,*64*) *larray*›
type-synonym *lbd-assn* = ‹(*32 word*) *larray64* × *32 word* × *32 word*›

abbreviation *lbd-int-assn* :: ‹*lbd-ref* ⇒ *lbd-assn* ⇒ *assn*› **where**
‹*lbd-int-assn* ≡ *larray64-assn uint32-nat-assn* ×$_a$ *uint32-nat-assn* ×$_a$ *uint32-nat-assn*›

definition *lbd-assn* :: ‹*lbd* ⇒ *lbd-assn* ⇒ *assn*› **where**
‹*lbd-assn* ≡ *hr-comp lbd-int-assn lbd-ref*›

Testing if a level is marked **sepref-def** *level-in-lbd-code*
is [] ‹*uncurry* (*RETURN oo level-in-lbd-ref*)›

:: ‹*uint32-nat-assn*^{k} $*_a$ *lbd-int-assn*^{k} $\rightarrow_a$ *bool1-assn*›
supply [[*goals-limit=1*]]
unfolding *level-in-lbd-ref-def short-circuit-conv length-uint32-nat-def*
apply (*rewrite* **in** ‹¤ < -› *annot-unat-snat-upcast*[**where** *'l*=‹*64*›])
apply (*rewrite* **in** ‹- ! ¤› *annot-unat-snat-upcast*[**where** *'l*=‹*64*›])
by *sepref*

lemma *level-in-lbd-hnr*[*sepref-fr-rules*]:
‹(*uncurry level-in-lbd-code*, *uncurry* (*RETURN* ∘∘ *level-in-lbd*)) ∈ *uint32-nat-assn*^{k} $*_a$ *lbd-assn*^{k} $\rightarrow_a$ *bool1-assn*›
supply *lbd-ref-def*[*simp*] *unat32-max-def*[*simp*]
using *level-in-lbd-code.refine*[*FCOMP level-in-lbd-ref-level-in-lbd*]
unfolding *lbd-assn-def*[*symmetric*]
by *simp*

sepref-def *lbd-empty-loop-code*
is ‹*lbd-empty-loop-ref*›
:: ‹*lbd-int-assn*^{d} $\rightarrow_a$ *lbd-int-assn*›
unfolding *lbd-empty-loop-ref-def*
supply [[*goals-limit=1*]]
apply (*rewrite at* ‹- + ¤› *snat-const-fold*[**where** *'a=64*])+
apply (*rewrite at* ‹(-, ¤)› *snat-const-fold*[**where** *'a=64*])
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

sepref-def *lbd-empty-cheap-code*
is ‹*lbd-empty-cheap-ref*›
:: ‹[λ(-, *stamp*, -). *stamp* < *unat32-max*]$_a$ *lbd-int-assn*^{d} → *lbd-int-assn*›
unfolding *lbd-empty-cheap-ref-def*
supply [[*goals-limit=1*]]
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

lemma *unat32-max-alt-def*: *unat32-max* = *4294967295*
by (*auto simp*: *unat32-max-def*)
sepref-register *lbd-empty-cheap-ref lbd-empty-loop-ref*

sepref-def *lbd-empty-code*
is ‹*lbd-empty-ref*›
:: ‹*lbd-int-assn*^{d} $\rightarrow_a$ *lbd-int-assn*›
unfolding *lbd-empty-ref-def unat32-max-alt-def*
supply [[*goals-limit=1*]]
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

lemma *lbd-empty-hnr*[*sepref-fr-rules*]:
‹(*lbd-empty-code*, *lbd-empty*) ∈ *lbd-assn*^{d} $\rightarrow_a$ *lbd-assn*›
using *lbd-empty-code.refine*[*FCOMP lbd-empty-ref-lbd-empty*]
unfolding *lbd-assn-def* **.**

sepref-def *empty-lbd-code*
is [] ‹*uncurry0* (*RETURN empty-lbd-ref*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *lbd-int-assn*›
supply [[*goals-limit=1*]]
unfolding *empty-lbd-ref-def larray-fold-custom-replicate*

apply (*rewrite at ‹op-larray-custom-replicate ⌑ -› snat-const-fold*[**where** *'a=64*])
apply (*annot-unat-const ‹TYPE(32)›*)
by *sepref*

lemma *empty-lbd-ref-empty-lbd*:
‹(*uncurry0* (*RETURN empty-lbd-ref*), *uncurry0* (*RETURN empty-lbd*)) ∈ *unit-rel* →$_f$ ⟨*lbd-ref*⟩*nres-rel*›
using *empty-lbd-ref-empty-lbd* **unfolding** *uncurry0-def* **.**

lemma *empty-lbd-hnr*[*sepref-fr-rules*]:
‹(*Sepref-Misc.uncurry0 empty-lbd-code*, *Sepref-Misc.uncurry0* (*RETURN empty-lbd*)) ∈ *unit-assn*^{k} →$_a$ *lbd-assn*›
using *empty-lbd-code.refine*[*FCOMP empty-lbd-ref-empty-lbd*]
unfolding *lbd-assn-def* **.**

sepref-def *get-LBD-code*
is [] ‹*get-LBD-ref*›
:: ‹*lbd-int-assn*^{k} →$_a$ *uint32-nat-assn*›
unfolding *get-LBD-ref-def*
by *sepref*

lemma *get-LBD-hnr*[*sepref-fr-rules*]:
‹(*get-LBD-code*, *get-LBD*) ∈ *lbd-assn*^{k} →$_a$ *uint32-nat-assn*›
using *get-LBD-code.refine*[*FCOMP get-LBD-ref-get-LBD*,
unfolded lbd-assn-def[*symmetric*]] **.**

Marking more levels **lemmas** *list-grow-alt* = *list-grow-def*[*unfolded op-list-grow-init'-def*[*symmetric*]]

sepref-def *lbd-write-code*
is [] ‹*uncurry lbd-ref-write*›
:: ‹ [λ(*lbd*, *i*). *i* ≤ *Suc* (*unat32-max div 2*)]$_a$
lbd-int-assn^d *$_a$ *uint32-nat-assn*^{k} → *lbd-int-assn*›
supply [[*goals-limit=1*]]
unfolding *lbd-ref-write-def length-uint32-nat-def list-grow-alt max-def*
op-list-grow-init'-alt
apply (*rewrite at ‹- + ⌑›* *unat-const-fold*[**where** *'a=32*])
apply (*rewrite at ‹- + ⌑›* *unat-const-fold*[**where** *'a=32*])
apply (*rewrite* **in** ‹*If* (⌑ < -)› *annot-unat-snat-upcast*[**where** *'l=64*])
apply (*rewrite* **in** ‹*If* (- ! ⌑ = -)› *annot-unat-snat-upcast*[**where** *'l=64*])
apply (*rewrite* **in** ‹-[⌑ := -]› *annot-unat-snat-upcast*[**where** *'l=64*])
apply (*rewrite* **in** ‹*op-list-grow-init* - ⌑ -› *annot-unat-snat-upcast*[**where** *'l=64*])
apply (*rewrite at ‹(-[⌑ := -], -, - + -)›* *annot-unat-snat-upcast*[**where** *'l=64*])
apply (*annot-unat-const ‹TYPE(32)›*)
by *sepref*

lemma *lbd-write-hnr-*[*sepref-fr-rules*]:
‹(*uncurry lbd-write-code*, *uncurry* (*RETURN* ∘∘ *lbd-write*))
∈ [λ(*lbd*, *i*). *i* ≤ *Suc* (*unat32-max div 2*)]$_a$
lbd-assn^d *$_a$ *uint32-nat-assn*^{k} → *lbd-assn*›
using *lbd-write-code.refine*[*FCOMP lbd-ref-write-lbd-write*]
unfolding *lbd-assn-def* **.**

schematic-goal *mk-free-lbd-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE lbd-assn ?fr*›
unfolding *lbd-assn-def*
by *synthesize-free*

experiment begin

export-llvm
level-in-lbd-code
lbd-empty-code
empty-lbd-code
get-LBD-code
lbd-write-code

end

end
theory *Version*
imports *Main*
begin

This code was taken from IsaFoR and adapted to git.

local-setup ‹
let
val version =
trim-line (#1 (Isabelle-System.bash-output (cd $ISAFOL/ && git rev−parse −−short HEAD || echo unknown)))
in
Local-Theory.define
((***binding*** ‹*version*›, *NoSyn*),
((***binding*** ‹*version-def*›, []), *HOLogic.mk-literal version*)) #> *#2*
end
›

declare *version-def* [*code*]

end
theory *IsaSAT-Watch-List*
imports *IsaSAT-Literals IsaSAT-Clauses Watched-Literals.Watched-Literals-Watch-List-Initialisation*
Pairing-Heap-LLVM.Map-Fun-Rel
begin

Chapter 7

Refinement of the Watched Function

There is not much to say about watch lists since they are arrays of resizeable arrays, which are defined in a separate theory.

However, when replacing the elements in our watch lists from (*nat* × *uint32*) to (*nat* × *uint32* × *bool*) to enable special handling of binary clauses, we got a huge and unexpected slowdown, due to a much higher number of cache misses (roughly 3.5 times as many on a eq.atree.braun.8.unsat.cnf which also took 66s instead of 50s). While toying with the generated ML code, we found out that our version of the tuples with booleans were using 40 bytes instead of 24 previously. Just merging the *uint32* and the *bool* to a single *uint64* was sufficient to get the performance back.

Remark that however, the evaluation of terms like (*2*::*uint64*) ^ *32* was not done automatically and even worse, was redone each time, leading to a complete performance blow-up (75s on my macbook for eq.atree.braun.7.unsat.cnf instead of 7s).

None of the problems appears in the LLVM code.

7.1 Definition

definition *mop-append-ll* :: ‹*'a list list* ⇒ *nat literal* ⇒ *'a* ⇒ *'a list list nres*› **where**
‹*mop-append-ll xs i x* = *do* {
ASSERT(*nat-of-lit i* < *length xs*);
RETURN (*append-ll xs* (*nat-of-lit i*) *x*)
}›

7.2 Operations

lemma *length-ll-length-ll-f*:
‹(*uncurry* (*RETURN oo length-ll*), *uncurry* (*RETURN oo length-ll-f*)) ∈
[λ(*W*, *L*). *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}_{in}$]$_f$ ((⟨*Id*⟩*map-fun-rel* (*D*$_0$ $\mathcal{A}_{in}$)) ×$_r$ *nat-lit-rel*) →
⟨*nat-rel*⟩ *nres-rel*›
unfolding *length-ll-def length-ll-f-def*
by (*fastforce simp*: *fref-def map-fun-rel-def prod-rel-def nres-rel-def p2rel-def br-def*
nat-lit-rel-def)

lemma *mop-append-ll*:
‹(*uncurry2 mop-append-ll*, *uncurry2* (*RETURN ooo* (λ*W i x*. *W*(*i* := *W i* @ [*x*])))) ∈
[λ((*W*, *i*), *x*). *i* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$ ⟨*Id*⟩*map-fun-rel* (*D*$_0$ $\mathcal{A}$) ×$_f$ *Id* ×$_f$ *Id* → ⟨⟨*Id*⟩*map-fun-rel* (*D*$_0$ $\mathcal{A}$)⟩*nres-rel*›
unfolding *uncurry-def mop-append-ll-def*

by (*intro frefI nres-relI*)
(*auto intro*!: *ASSERT-leI simp*: *map-fun-rel-def append-ll-def*)

definition *delete-index-and-swap-update* :: ‹('*a* ⇒ '*b list*) ⇒ '*a* ⇒ *nat* ⇒ '*a* ⇒ '*b list*› **where**
‹*delete-index-and-swap-update W K w* = *W*(*K* := *delete-index-and-swap* (*W K*) *w*)›

The precondition is not necessary.

lemma *delete-index-and-swap-ll-delete-index-and-swap-update*:
‹(*uncurry2* (*RETURN ooo delete-index-and-swap-ll*), *uncurry2* (*RETURN ooo delete-index-and-swap-update*))
∈[λ((*W*, *L*), *i*). *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$ (⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$) ×$_r$ *nat-lit-rel*) ×$_r$ *nat-rel* →
⟨⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)⟩*nres-rel*›
by (*auto simp*: *delete-index-and-swap-ll-def uncurry-def fref-def nres-rel-def*
delete-index-and-swap-update-def map-fun-rel-def p2rel-def nat-lit-rel-def br-def
nth-list-update′ *nat-lit-rel-def*
simp del: *literal-of-nat.simps*)

type-synonym *nat-clauses-l* = ‹*nat list list*›

Refinement of the Watched Function

definition *watched-by-nth* :: ‹*nat twl-st-wl* ⇒ *nat literal* ⇒ *nat* ⇒ *nat watcher*› **where**
‹*watched-by-nth* = (λ(*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*) *L i*. *W L* ! *i*)›

definition *watched-app*
:: ‹(*nat literal* ⇒ (*nat watcher*) *list*) ⇒ *nat literal* ⇒ *nat* ⇒ *nat watcher*› **where**
‹*watched-app M L i* ≡ *M L* ! *i*›

lemma *watched-by-nth-watched-app*:
‹*watched-by S K* ! *w* = *watched-app* ((*snd o snd o snd o snd o snd o snd o snd o snd o snd o snd o snd o snd*) *S*) *K w*›
by (*cases S*) (*auto simp*: *watched-app-def*)

lemma *nth-ll-watched-app*:
‹(*uncurry2* (*RETURN ooo nth-rll*), *uncurry2* (*RETURN ooo watched-app*)) ∈
[λ((*W*, *L*), *i*). *L* ∈# ($\mathcal{L}_{all}$ $\mathcal{A}$)]$_f$ ((⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)) ×$_r$ *nat-lit-rel*) ×$_r$ *nat-rel* →
⟨*nat-rel* ×$_r$ *Id*⟩ *nres-rel*›
unfolding *watched-app-def nth-rll-def*
by (*fastforce simp*: *fref-def map-fun-rel-def prod-rel-def nres-rel-def p2rel-def br-def*
nat-lit-rel-def)

7.3 Rewatch

definition *rewatch-heur* **where**
‹*rewatch-heur vdom arena W* = *do* {
let - = *vdom*;
nfoldli [*0*..<*length vdom*] (λ-. *True*)
(λ*i W*. *do* {
ASSERT(*i* < *length vdom*);
let C = *vdom* ! *i*;
ASSERT(*arena-is-valid-clause-vdom arena C*);
if arena-status arena C ≠ *DELETED*
then do {

```
      L1 ← mop-arena-lit2 arena C 0;
      L2 ← mop-arena-lit2 arena C 1;
      n ← mop-arena-length arena C;
      let b = (n = 2);
      ASSERT(length (W ! (nat-of-lit L1)) < length arena);
      W ← mop-append-ll W L1 (C, L2, b);
      ASSERT(length (W ! (nat-of-lit L2)) < length arena);
      W ← mop-append-ll W L2 (C, L1, b);
      RETURN W
    }
    else RETURN W
  })
  W
 }›

lemma rewatch-heur-rewatch:
 assumes
  valid: ‹valid-arena arena N vdom› and ‹set xs ⊆ vdom› and ‹distinct xs› and ‹set-mset (dom-m N)
⊆ set xs› and
  ‹(W, W') ∈ ⟨Id⟩map-fun-rel (D₀ 𝒜)› and lall: ‹literals-are-in-ℒ_in-mm 𝒜 (mset `# ran-mf N)› and
  ‹vdom-m 𝒜 W' N ⊆ set-mset (dom-m N)›
 shows
  ‹rewatch-heur xs arena W ≤ ⇓ ({(W, W'). (W, W') ∈⟨Id⟩map-fun-rel (D₀ 𝒜) ∧ vdom-m 𝒜 W' N
⊆ set-mset (dom-m N)}) (rewatch N W')›
proof −
 have [refine0]: ‹(xs, xsa) ∈ Id ⟹
   ([0..<length xs], [0..<length xsa]) ∈ ⟨{(x, x'). x = x' ∧ x < length xsa ∧ xs!x ∈ vdom}⟩list-rel›
  for xsa
  using assms unfolding list-rel-def
  by (auto simp: list-all2-same)
 show ?thesis
  unfolding rewatch-heur-def rewatch-def
  apply (subst (2) nfoldli-nfoldli-list-nth)
  apply (refine-vcg mop-arena-lit[OF valid] mop-append-ll[of 𝒜, THEN fref-to-Down-curry2, unfolded
comp-def]
     mop-arena-length[of vdom, THEN fref-to-Down-curry, unfolded comp-def])
  subgoal
   using assms by fast
  subgoal
   using assms by fast
  subgoal
   using assms by fast
  subgoal by fast
  subgoal by auto
  subgoal
   using assms
   unfolding arena-is-valid-clause-vdom-def
   by blast
  subgoal
   using assms
   by (auto simp: arena-dom-status-iff)
  subgoal for xsa xi x si s
   using assms
   by auto
  subgoal by simp
  subgoal by linarith
```

```
    subgoal for xsa xi x si s
      using assms
      unfolding arena-lit-pre-def
      by (auto)
    subgoal by simp
    subgoal by simp
    subgoal by simp
    subgoal for xsa xi x si s
      using assms
      unfolding arena-is-valid-clause-idx-and-access-def
        arena-is-valid-clause-idx-def
      by (auto simp: arena-is-valid-clause-idx-and-access-def
          intro!: exI[of - N] exI[of - vdom])
    subgoal for xsa xi x si s
      using valid-arena-size-dom-m-le-arena[OF assms(1)] assms
        literals-are-in-\mathcal{L}_in-mm-in-\mathcal{L}_all[OF lall, of ‹xs ! xi› 0]
      by (auto simp: map-fun-rel-def arena-lifting)
    subgoal for xsa xi x si s
      using valid-arena-size-dom-m-le-arena[OF assms(1)] assms
        literals-are-in-\mathcal{L}_in-mm-in-\mathcal{L}_all[OF lall, of ‹xs ! xi› 0]
      by (auto simp: map-fun-rel-def arena-lifting)
    subgoal using assms by (simp add: arena-lifting)
    subgoal for xsa xi x si s
      using literals-are-in-\mathcal{L}_in-mm-in-\mathcal{L}_all[OF lall, of ‹xs ! xi› 1]
      assms valid-arena-size-dom-m-le-arena[OF assms(1)]
      by (auto simp: arena-lifting append-ll-def map-fun-rel-def)
    subgoal for xsa xi x si s
      using literals-are-in-\mathcal{L}_in-mm-in-\mathcal{L}_all[OF lall, of ‹xs ! xi› 1]
        assms
      by (auto simp: arena-lifting append-ll-def map-fun-rel-def)
    subgoal for xsa xi x si s
      using assms
      by (auto simp: arena-lifting append-ll-def map-fun-rel-def)
    subgoal for xsa xi x si s
      using assms
      by (auto simp: arena-lifting append-ll-def map-fun-rel-def)
    done
qed

lemma rewatch-heur-alt-def:
‹rewatch-heur vdom arena W = do {
  let - = vdom;
  nfoldli [0..<length vdom] (λ-. True)
   (λi W. do {
      ASSERT(i < length vdom);
      let C = vdom ! i;
      ASSERT(arena-is-valid-clause-vdom arena C);
      if arena-status arena C ≠ DELETED
      then do {
        L1 ← mop-arena-lit2 arena C 0;
        L2 ← mop-arena-lit2 arena C 1;
        n ← mop-arena-length arena C;
        let b = (n = 2);
        ASSERT(length (W ! (nat-of-lit L1)) < length arena);
        W ← mop-append-ll W L1 (C, L2, b);
        ASSERT(length (W ! (nat-of-lit L2)) < length arena);
```

```
      W ← mop-append-ll W L2 (C, L1, b);
      RETURN W
    }
    else RETURN W
  })
  W
 }›
 unfolding Let-def rewatch-heur-def
 by auto

definition watchlist-put-binaries-first-one :: ‹-› where
 ‹watchlist-put-binaries-first-one W₀ L = do {
    ASSERT (L < length W₀);
    let m = length (W₀ ! L);
   (-, -, W) ← WHILE_T(λ(i,j,W). length W = length W₀ ∧ length (W ! L) = length (W₀ ! L) ∧ i ≤ j ∧ (∀K. mset (W ! K)
(λ(i,j,W). j < m)
     (λ(i,j,W). do {
       ASSERT (j < length (W ! L));
       let (-, -, b) = W ! L ! j;
       if b then RETURN (i+1,j+1, W[L := swap (W!L) i j])
       else RETURN (i+1, j+1, W)
     })
     (0, 0, W₀);
   RETURN W
}›

definition watchlist-put-binaries-first :: ‹-› where
 ‹watchlist-put-binaries-first W₀ = do {
    let m = length W₀;
   (-, W) ← WHILE_T λ(i, W). length W = length W₀ ∧ (∀K. mset (W ! K) = mset (W₀ ! K)) (λ(i,W).
i < m)
     (λ(i,W). do {
       ASSERT (i < length (W));
       W ← watchlist-put-binaries-first-one W i;
       RETURN (i+1, W)
     })
     (0, W₀);
   RETURN W
}›



definition rewatch-heur-and-reorder where
‹rewatch-heur-and-reorder vdom arena W = do {
   W ← rewatch-heur vdom arena W;
   watchlist-put-binaries-first W}›


end
theory Tuple17
 imports More-Sepref.WB-More-Refinement IsaSAT-Literals
begin
```

datatype ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j, 'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* = *Tuple17*
('a) *Tuple17-get-a*
('b) *Tuple17-get-b*
('c) *Tuple17-get-c*
('d) *Tuple17-get-d*
('e) *Tuple17-get-e*
('f) *Tuple17-get-f*
('g) *Tuple17-get-g*
('h) *Tuple17-get-h*
('i) *Tuple17-get-i*
('j) *Tuple17-get-j*
('k) *Tuple17-get-k*
('l) *Tuple17-get-l*
('m) *Tuple17-get-m*
('n) *Tuple17-get-n*
('o) *Tuple17-get-o*
('p) *Tuple17-get-p*
('q) *Tuple17-get-q*

Accessors context
begin
qualified fun *set-a* :: ‹'a ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j, 'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ -› **where**
‹*set-a M* (*Tuple17* - *N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs*) = (*Tuple17 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs*)›

fun *set-b* :: ‹'b ⇒('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j, 'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ -› **where**
‹*set-b N* (*Tuple17 M* - *D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs*) = (*Tuple17 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs*)›

fun *set-c* :: ‹'c ⇒('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j, 'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ -› **where**
‹*set-c D* (*Tuple17 M N* - *i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs*) = (*Tuple17 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs*)›

fun *set-d* :: ‹'d ⇒('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j, 'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ -› **where**
‹*set-d i* (*Tuple17 M N D* - *W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs*) = (*Tuple17 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs*)›

fun *set-e* :: ‹'e ⇒('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j, 'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ -› **where**
‹*set-e W* (*Tuple17 M N D i* - *ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs*) = (*Tuple17 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs*)›

fun *set-f* :: ‹'f ⇒('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j, 'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ -› **where**
‹*set-f ivmtf* (*Tuple17 M N D i W* - *icount ccach lbd outl heur stats aivdom clss opts arena occs*) = (*Tuple17 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs*)›

fun *set-g* :: ‹'g ⇒('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j, 'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ -› **where**
‹*set-g icount* (*Tuple17 M N D i W ivmtf* - *ccach lbd outl heur stats aivdom clss opts arena occs*) = (*Tuple17 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs*)›

fun *set-h* :: ‹'h ⇒('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j, 'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ -› **where**
‹*set-h ccach* (*Tuple17 M N D i W ivmtf icount* - *lbd outl heur stats aivdom clss opts arena occs*) = (*Tuple17 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs*)›

fun *set-i* :: ‹'i ⇒('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j, 'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ -› **where**
‹*set-i lbd* (*Tuple17 M N D i W ivmtf icount ccach* - *outl heur stats aivdom clss opts arena occs*) =

(*Tuple17 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs*)›

fun *set-j* :: ‹'*j* ⇒('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*, '*p*, '*q*) *tuple17* ⇒ -› **where**
‹*set-j outl* (*Tuple17 M N D i W ivmtf icount ccach lbd - heur stats aivdom clss opts arena occs*) =
(*Tuple17 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs*)›

fun *set-k* :: ‹'*k* ⇒('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*, '*p*, '*q*) *tuple17* ⇒ -› **where**
‹*set-k stats* (*Tuple17 M N D i W ivmtf icount ccach lbd outl - heur aivdom clss opts arena occs*) =
(*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*)›

fun *set-l* :: ‹'*l* ⇒('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*, '*p*, '*q*) *tuple17* ⇒ -› **where**
‹*set-l heur* (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats - aivdom clss opts arena occs*) =
(*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*)›

fun *set-m* :: ‹'*m* ⇒('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*, '*p*, '*q*) *tuple17* ⇒ -› **where**
‹*set-m aivdom* (*Tuple17 M N D i W ivmtf icount ccach lbd outl heur stats - clss opts arena occs*) =
(*Tuple17 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs*)›

fun *set-n* :: ‹'*n* ⇒('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*, '*p*, '*q*) *tuple17* ⇒ -› **where**
‹*set-n clss* (*Tuple17 M N D i W ivmtf icount ccach lbd outl heur stats aivdom - opts arena occs*) =
(*Tuple17 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs*)›

fun *set-o* :: ‹'*o* ⇒('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*, '*p*, '*q*) *tuple17* ⇒ -› **where**
‹*set-o opts* (*Tuple17 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss - arena occs*) =
(*Tuple17 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs*)›

fun *set-p* :: ‹'*p* ⇒('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*, '*p*, '*q*) *tuple17* ⇒ -› **where**
‹*set-p arena* (*Tuple17 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts - occs*) =
(*Tuple17 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs*)›

fun *set-q* :: ‹'*q* ⇒('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*, '*p*, '*q*) *tuple17* ⇒ -› **where**
‹*set-q occs* (*Tuple17 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena* -) =
(*Tuple17 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs*)›
end

named-theorems *tuple17-state-simp*
lemma [*tuple17-state-simp*]:
‹*Tuple17-get-a* (*Tuple17.set-a M S*) = *M*›
‹*Tuple17-get-b* (*Tuple17.set-a M S*) = *Tuple17-get-b S*›
‹*Tuple17-get-c*(*Tuple17.set-a M S*) = *Tuple17-get-c S*›
‹*Tuple17-get-d* (*Tuple17.set-a M S*) = *Tuple17-get-d S*›
‹*Tuple17-get-e* (*Tuple17.set-a M S*) = *Tuple17-get-e S*›
‹*Tuple17-get-f* (*Tuple17.set-a M S*) = *Tuple17-get-f S*›
‹*Tuple17-get-g* (*Tuple17.set-a M S*) = *Tuple17-get-g S*›
‹*Tuple17-get-h* (*Tuple17.set-a M S*) = *Tuple17-get-h S*›
‹*Tuple17-get-i* (*Tuple17.set-a M S*) = *Tuple17-get-i S*›
‹*Tuple17-get-j* (*Tuple17.set-a M S*) = *Tuple17-get-j S*›
‹*Tuple17-get-k* (*Tuple17.set-a M S*) = *Tuple17-get-k S*›
‹*Tuple17-get-l* (*Tuple17.set-a M S*) = *Tuple17-get-l S*›
‹*Tuple17-get-m* (*Tuple17.set-a M S*) = *Tuple17-get-m S*›
‹*Tuple17-get-n* (*Tuple17.set-a M S*) = *Tuple17-get-n S*›
‹*Tuple17-get-o* (*Tuple17.set-a M S*) = *Tuple17-get-o S*›
‹*Tuple17-get-p* (*Tuple17.set-a M S*) = *Tuple17-get-p S*›
‹*Tuple17-get-q* (*Tuple17.set-a M S*) = *Tuple17-get-q S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple17-state-simp*]:
‹*Tuple17-get-a* (*Tuple17.set-b N S*) = *Tuple17-get-a S*›
‹*Tuple17-get-b* (*Tuple17.set-b N S*) = *N*›
‹*Tuple17-get-c*(*Tuple17.set-b N S*) = *Tuple17-get-c S*›
‹*Tuple17-get-d* (*Tuple17.set-b N S*) = *Tuple17-get-d S*›
‹*Tuple17-get-e* (*Tuple17.set-b N S*) = *Tuple17-get-e S*›
‹*Tuple17-get-f* (*Tuple17.set-b N S*) = *Tuple17-get-f S*›
‹*Tuple17-get-g* (*Tuple17.set-b N S*) = *Tuple17-get-g S*›
‹*Tuple17-get-h* (*Tuple17.set-b N S*) = *Tuple17-get-h S*›
‹*Tuple17-get-i* (*Tuple17.set-b N S*) = *Tuple17-get-i S*›
‹*Tuple17-get-j* (*Tuple17.set-b N S*) = *Tuple17-get-j S*›
‹*Tuple17-get-k* (*Tuple17.set-b N S*) = *Tuple17-get-k S*›
‹*Tuple17-get-l* (*Tuple17.set-b N S*) = *Tuple17-get-l S*›
‹*Tuple17-get-m* (*Tuple17.set-b N S*) = *Tuple17-get-m S*›
‹*Tuple17-get-n* (*Tuple17.set-b N S*) = *Tuple17-get-n S*›
‹*Tuple17-get-o* (*Tuple17.set-b N S*) = *Tuple17-get-o S*›
‹*Tuple17-get-p* (*Tuple17.set-b N S*) = *Tuple17-get-p S*›
‹*Tuple17-get-q* (*Tuple17.set-b N S*) = *Tuple17-get-q S*›

‹*Tuple17-get-a* (*Tuple17.set-c D S*) = *Tuple17-get-a S*›
‹*Tuple17-get-b* (*Tuple17.set-c D S*) = *Tuple17-get-b S*›
‹*Tuple17-get-c*(*Tuple17.set-c D S*) = *D*›
‹*Tuple17-get-d* (*Tuple17.set-c D S*) = *Tuple17-get-d S*›
‹*Tuple17-get-e* (*Tuple17.set-c D S*) = *Tuple17-get-e S*›
‹*Tuple17-get-f* (*Tuple17.set-c D S*) = *Tuple17-get-f S*›
‹*Tuple17-get-g* (*Tuple17.set-c D S*) = *Tuple17-get-g S*›
‹*Tuple17-get-h* (*Tuple17.set-c D S*) = *Tuple17-get-h S*›
‹*Tuple17-get-i* (*Tuple17.set-c D S*) = *Tuple17-get-i S*›
‹*Tuple17-get-j* (*Tuple17.set-c D S*) = *Tuple17-get-j S*›
‹*Tuple17-get-k* (*Tuple17.set-c D S*) = *Tuple17-get-k S*›
‹*Tuple17-get-l* (*Tuple17.set-c D S*) = *Tuple17-get-l S*›
‹*Tuple17-get-m* (*Tuple17.set-c D S*) = *Tuple17-get-m S*›
‹*Tuple17-get-n* (*Tuple17.set-c D S*) = *Tuple17-get-n S*›
‹*Tuple17-get-o* (*Tuple17.set-c D S*) = *Tuple17-get-o S*›
‹*Tuple17-get-p* (*Tuple17.set-c D S*) = *Tuple17-get-p S*›
‹*Tuple17-get-q* (*Tuple17.set-c D S*) = *Tuple17-get-q S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple17-state-simp*]:
‹*Tuple17-get-a* (*Tuple17.set-d j S*) = *Tuple17-get-a S*›
‹*Tuple17-get-b* (*Tuple17.set-d j S*) = *Tuple17-get-b S*›
‹*Tuple17-get-c*(*Tuple17.set-d j S*) = *Tuple17-get-c S*›
‹*Tuple17-get-d* (*Tuple17.set-d j S*) = *j*›
‹*Tuple17-get-e* (*Tuple17.set-d j S*) = *Tuple17-get-e S*›
‹*Tuple17-get-f* (*Tuple17.set-d j S*) = *Tuple17-get-f S*›
‹*Tuple17-get-g* (*Tuple17.set-d j S*) = *Tuple17-get-g S*›
‹*Tuple17-get-h* (*Tuple17.set-d j S*) = *Tuple17-get-h S*›
‹*Tuple17-get-i* (*Tuple17.set-d j S*) = *Tuple17-get-i S*›
‹*Tuple17-get-j* (*Tuple17.set-d j S*) = *Tuple17-get-j S*›
‹*Tuple17-get-k* (*Tuple17.set-d j S*) = *Tuple17-get-k S*›
‹*Tuple17-get-l* (*Tuple17.set-d j S*) = *Tuple17-get-l S*›
‹*Tuple17-get-m* (*Tuple17.set-d j S*) = *Tuple17-get-m S*›
‹*Tuple17-get-n* (*Tuple17.set-d j S*) = *Tuple17-get-n S*›
‹*Tuple17-get-o* (*Tuple17.set-d j S*) = *Tuple17-get-o S*›
‹*Tuple17-get-p* (*Tuple17.set-d j S*) = *Tuple17-get-p S*›
‹*Tuple17-get-q* (*Tuple17.set-d j S*) = *Tuple17-get-q S*›

by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple17-state-simp*]:
‹*Tuple17-get-a* (*Tuple17.set-e W S*) = *Tuple17-get-a S*›
‹*Tuple17-get-b* (*Tuple17.set-e W S*) = *Tuple17-get-b S*›
‹*Tuple17-get-c*(*Tuple17.set-e W S*) = *Tuple17-get-c S*›
‹*Tuple17-get-d* (*Tuple17.set-e W S*) = *Tuple17-get-d S*›
‹*Tuple17-get-e* (*Tuple17.set-e W S*) = *W*›
‹*Tuple17-get-f* (*Tuple17.set-e W S*) = *Tuple17-get-f S*›
‹*Tuple17-get-g* (*Tuple17.set-e W S*) = *Tuple17-get-g S*›
‹*Tuple17-get-h* (*Tuple17.set-e W S*) = *Tuple17-get-h S*›
‹*Tuple17-get-i* (*Tuple17.set-e W S*) = *Tuple17-get-i S*›
‹*Tuple17-get-j* (*Tuple17.set-e W S*) = *Tuple17-get-j S*›
‹*Tuple17-get-k* (*Tuple17.set-e W S*) = *Tuple17-get-k S*›
‹*Tuple17-get-l* (*Tuple17.set-e W S*) = *Tuple17-get-l S*›
‹*Tuple17-get-m* (*Tuple17.set-e W S*) = *Tuple17-get-m S*›
‹*Tuple17-get-n* (*Tuple17.set-e W S*) = *Tuple17-get-n S*›
‹*Tuple17-get-o* (*Tuple17.set-e W S*) = *Tuple17-get-o S*›
‹*Tuple17-get-p* (*Tuple17.set-e W S*) = *Tuple17-get-p S*›
‹*Tuple17-get-q* (*Tuple17.set-e W S*) = *Tuple17-get-q S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple17-state-simp*]:
‹*Tuple17-get-a* (*Tuple17.set-f vmtf' S*) = *Tuple17-get-a S*›
‹*Tuple17-get-b* (*Tuple17.set-f vmtf' S*) = *Tuple17-get-b S*›
‹*Tuple17-get-c*(*Tuple17.set-f vmtf' S*) = *Tuple17-get-c S*›
‹*Tuple17-get-d* (*Tuple17.set-f vmtf' S*) = *Tuple17-get-d S*›
‹*Tuple17-get-e* (*Tuple17.set-f vmtf' S*) = *Tuple17-get-e S*›
‹*Tuple17-get-f* (*Tuple17.set-f vmtf' S*) = *vmtf'*›
‹*Tuple17-get-g* (*Tuple17.set-f vmtf' S*) = *Tuple17-get-g S*›
‹*Tuple17-get-h* (*Tuple17.set-f vmtf' S*) = *Tuple17-get-h S*›
‹*Tuple17-get-i* (*Tuple17.set-f vmtf' S*) = *Tuple17-get-i S*›
‹*Tuple17-get-j* (*Tuple17.set-f vmtf' S*) = *Tuple17-get-j S*›
‹*Tuple17-get-k* (*Tuple17.set-f vmtf' S*) = *Tuple17-get-k S*›
‹*Tuple17-get-l* (*Tuple17.set-f vmtf' S*) = *Tuple17-get-l S*›
‹*Tuple17-get-m* (*Tuple17.set-f vmtf' S*) = *Tuple17-get-m S*›
‹*Tuple17-get-n* (*Tuple17.set-f vmtf' S*) = *Tuple17-get-n S*›
‹*Tuple17-get-o* (*Tuple17.set-f vmtf' S*) = *Tuple17-get-o S*›
‹*Tuple17-get-p* (*Tuple17.set-f vmtf' S*) = *Tuple17-get-p S*›
‹*Tuple17-get-q* (*Tuple17.set-f vmtf' S*) = *Tuple17-get-q S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple17-state-simp*]:
‹*Tuple17-get-a* (*Tuple17.set-g count' S*) = *Tuple17-get-a S*›
‹*Tuple17-get-b* (*Tuple17.set-g count' S*) = *Tuple17-get-b S*›
‹*Tuple17-get-c*(*Tuple17.set-g count' S*) = *Tuple17-get-c S*›
‹*Tuple17-get-d* (*Tuple17.set-g count' S*) = *Tuple17-get-d S*›
‹*Tuple17-get-e* (*Tuple17.set-g count' S*) = *Tuple17-get-e S*›
‹*Tuple17-get-f* (*Tuple17.set-g count' S*) = *Tuple17-get-f S*›
‹*Tuple17-get-g* (*Tuple17.set-g count' S*) = *count'*›
‹*Tuple17-get-h* (*Tuple17.set-g count' S*) = *Tuple17-get-h S*›
‹*Tuple17-get-i* (*Tuple17.set-g count' S*) = *Tuple17-get-i S*›
‹*Tuple17-get-j* (*Tuple17.set-g count' S*) = *Tuple17-get-j S*›
‹*Tuple17-get-k* (*Tuple17.set-g count' S*) = *Tuple17-get-k S*›
‹*Tuple17-get-l* (*Tuple17.set-g count' S*) = *Tuple17-get-l S*›
‹*Tuple17-get-m* (*Tuple17.set-g count' S*) = *Tuple17-get-m S*›

‹*Tuple17-get-n* (*Tuple17.set-g count′ S*) = *Tuple17-get-n S*›
‹*Tuple17-get-o* (*Tuple17.set-g count′ S*) = *Tuple17-get-o S*›
‹*Tuple17-get-p* (*Tuple17.set-g count′ S*) = *Tuple17-get-p S*›
‹*Tuple17-get-q* (*Tuple17.set-g count′ S*) = *Tuple17-get-q S*›
by (*solves* ‹*cases S*; *auto simp*: ›)+

lemma [*tuple17-state-simp*]:
‹*Tuple17-get-a* (*Tuple17.set-h ccach S*) = *Tuple17-get-a S*›
‹*Tuple17-get-b* (*Tuple17.set-h ccach S*) = *Tuple17-get-b S*›
‹*Tuple17-get-c*(*Tuple17.set-h ccach S*) = *Tuple17-get-c S*›
‹*Tuple17-get-d* (*Tuple17.set-h ccach S*) = *Tuple17-get-d S*›
‹*Tuple17-get-e* (*Tuple17.set-h ccach S*) = *Tuple17-get-e S*›
‹*Tuple17-get-f* (*Tuple17.set-h ccach S*) = *Tuple17-get-f S*›
‹*Tuple17-get-g* (*Tuple17.set-h ccach S*) = *Tuple17-get-g S*›
‹*Tuple17-get-h* (*Tuple17.set-h ccach S*) = *ccach*›
‹*Tuple17-get-i* (*Tuple17.set-h ccach S*) = *Tuple17-get-i S*›
‹*Tuple17-get-j* (*Tuple17.set-h ccach S*) = *Tuple17-get-j S*›
‹*Tuple17-get-k* (*Tuple17.set-h ccach S*) = *Tuple17-get-k S*›
‹*Tuple17-get-l* (*Tuple17.set-h ccach S*) = *Tuple17-get-l S*›
‹*Tuple17-get-m* (*Tuple17.set-h ccach S*) = *Tuple17-get-m S*›
‹*Tuple17-get-n* (*Tuple17.set-h ccach S*) = *Tuple17-get-n S*›
‹*Tuple17-get-o* (*Tuple17.set-h ccach S*) = *Tuple17-get-o S*›
‹*Tuple17-get-p* (*Tuple17.set-h ccach S*) = *Tuple17-get-p S*›
‹*Tuple17-get-q* (*Tuple17.set-h ccach S*) = *Tuple17-get-q S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple17-state-simp*]:
‹*Tuple17-get-a* (*Tuple17.set-i lbd S*) = *Tuple17-get-a S*›
‹*Tuple17-get-b* (*Tuple17.set-i lbd S*) = *Tuple17-get-b S*›
‹*Tuple17-get-c*(*Tuple17.set-i lbd S*) = *Tuple17-get-c S*›
‹*Tuple17-get-d* (*Tuple17.set-i lbd S*) = *Tuple17-get-d S*›
‹*Tuple17-get-e* (*Tuple17.set-i lbd S*) = *Tuple17-get-e S*›
‹*Tuple17-get-f* (*Tuple17.set-i lbd S*) = *Tuple17-get-f S*›
‹*Tuple17-get-g* (*Tuple17.set-i lbd S*) = *Tuple17-get-g S*›
‹*Tuple17-get-h* (*Tuple17.set-i lbd S*) = *Tuple17-get-h S*›
‹*Tuple17-get-i* (*Tuple17.set-i lbd S*) = *lbd*›
‹*Tuple17-get-j* (*Tuple17.set-i lbd S*) = *Tuple17-get-j S*›
‹*Tuple17-get-k* (*Tuple17.set-i lbd S*) = *Tuple17-get-k S*›
‹*Tuple17-get-l* (*Tuple17.set-i lbd S*) = *Tuple17-get-l S*›
‹*Tuple17-get-m* (*Tuple17.set-i lbd S*) = *Tuple17-get-m S*›
‹*Tuple17-get-n* (*Tuple17.set-i lbd S*) = *Tuple17-get-n S*›
‹*Tuple17-get-o* (*Tuple17.set-i lbd S*) = *Tuple17-get-o S*›
‹*Tuple17-get-p* (*Tuple17.set-i lbd S*) = *Tuple17-get-p S*›
‹*Tuple17-get-q* (*Tuple17.set-i lbd S*) = *Tuple17-get-q S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple17-state-simp*]:
‹*Tuple17-get-a* (*Tuple17.set-j outl S*) = *Tuple17-get-a S*›
‹*Tuple17-get-b* (*Tuple17.set-j outl S*) = *Tuple17-get-b S*›
‹*Tuple17-get-c*(*Tuple17.set-j outl S*) = *Tuple17-get-c S*›
‹*Tuple17-get-d* (*Tuple17.set-j outl S*) = *Tuple17-get-d S*›
‹*Tuple17-get-e* (*Tuple17.set-j outl S*) = *Tuple17-get-e S*›
‹*Tuple17-get-f* (*Tuple17.set-j outl S*) = *Tuple17-get-f S*›
‹*Tuple17-get-g* (*Tuple17.set-j outl S*) = *Tuple17-get-g S*›
‹*Tuple17-get-h* (*Tuple17.set-j outl S*) = *Tuple17-get-h S*›
‹*Tuple17-get-i* (*Tuple17.set-j outl S*) = *Tuple17-get-i S*›

‹*Tuple17-get-j* (*Tuple17.set-j outl S*) = *outl*›
‹*Tuple17-get-k* (*Tuple17.set-j outl S*) = *Tuple17-get-k S*›
‹*Tuple17-get-l* (*Tuple17.set-j outl S*) = *Tuple17-get-l S*›
‹*Tuple17-get-m* (*Tuple17.set-j outl S*) = *Tuple17-get-m S*›
‹*Tuple17-get-n* (*Tuple17.set-j outl S*) = *Tuple17-get-n S*›
‹*Tuple17-get-o* (*Tuple17.set-j outl S*) = *Tuple17-get-o S*›
‹*Tuple17-get-p* (*Tuple17.set-j outl S*) = *Tuple17-get-p S*›
‹*Tuple17-get-q* (*Tuple17.set-j outl S*) = *Tuple17-get-q S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple17-state-simp*]:
‹*Tuple17-get-a* (*Tuple17.set-k stats S*) = *Tuple17-get-a S*›
‹*Tuple17-get-b* (*Tuple17.set-k stats S*) = *Tuple17-get-b S*›
‹*Tuple17-get-c*(*Tuple17.set-k stats S*) = *Tuple17-get-c S*›
‹*Tuple17-get-d* (*Tuple17.set-k stats S*) = *Tuple17-get-d S*›
‹*Tuple17-get-e* (*Tuple17.set-k stats S*) = *Tuple17-get-e S*›
‹*Tuple17-get-f* (*Tuple17.set-k stats S*) = *Tuple17-get-f S*›
‹*Tuple17-get-g* (*Tuple17.set-k stats S*) = *Tuple17-get-g S*›
‹*Tuple17-get-h* (*Tuple17.set-k stats S*) = *Tuple17-get-h S*›
‹*Tuple17-get-i* (*Tuple17.set-k stats S*) = *Tuple17-get-i S*›
‹*Tuple17-get-j* (*Tuple17.set-k stats S*) = *Tuple17-get-j S*›
‹*Tuple17-get-k* (*Tuple17.set-k stats S*) = *stats*›
‹*Tuple17-get-l* (*Tuple17.set-k stats S*) = *Tuple17-get-l S*›
‹*Tuple17-get-m* (*Tuple17.set-k stats S*) = *Tuple17-get-m S*›
‹*Tuple17-get-n* (*Tuple17.set-k stats S*) = *Tuple17-get-n S*›
‹*Tuple17-get-o* (*Tuple17.set-k stats S*) = *Tuple17-get-o S*›
‹*Tuple17-get-p* (*Tuple17.set-k stats S*) = *Tuple17-get-p S*›
‹*Tuple17-get-q* (*Tuple17.set-k stats S*) = *Tuple17-get-q S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple17-state-simp*]:
‹*Tuple17-get-a* (*Tuple17.set-l heur S*) = *Tuple17-get-a S*›
‹*Tuple17-get-b* (*Tuple17.set-l heur S*) = *Tuple17-get-b S*›
‹*Tuple17-get-c*(*Tuple17.set-l heur S*) = *Tuple17-get-c S*›
‹*Tuple17-get-d* (*Tuple17.set-l heur S*) = *Tuple17-get-d S*›
‹*Tuple17-get-e* (*Tuple17.set-l heur S*) = *Tuple17-get-e S*›
‹*Tuple17-get-f* (*Tuple17.set-l heur S*) = *Tuple17-get-f S*›
‹*Tuple17-get-g* (*Tuple17.set-l heur S*) = *Tuple17-get-g S*›
‹*Tuple17-get-h* (*Tuple17.set-l heur S*) = *Tuple17-get-h S*›
‹*Tuple17-get-i* (*Tuple17.set-l heur S*) = *Tuple17-get-i S*›
‹*Tuple17-get-j* (*Tuple17.set-l heur S*) = *Tuple17-get-j S*›
‹*Tuple17-get-k* (*Tuple17.set-l heur S*) = *Tuple17-get-k S*›
‹*Tuple17-get-l* (*Tuple17.set-l heur S*) = *heur*›
‹*Tuple17-get-m* (*Tuple17.set-l heur S*) = *Tuple17-get-m S*›
‹*Tuple17-get-n* (*Tuple17.set-l heur S*) = *Tuple17-get-n S*›
‹*Tuple17-get-o* (*Tuple17.set-l heur S*) = *Tuple17-get-o S*›
‹*Tuple17-get-p* (*Tuple17.set-l heur S*) = *Tuple17-get-p S*›
‹*Tuple17-get-q* (*Tuple17.set-l heur S*) = *Tuple17-get-q S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple17-state-simp*]:
‹*Tuple17-get-a* (*Tuple17.set-m aivdom S*) = *Tuple17-get-a S*›
‹*Tuple17-get-b* (*Tuple17.set-m aivdom S*) = *Tuple17-get-b S*›
‹*Tuple17-get-c*(*Tuple17.set-m aivdom S*) = *Tuple17-get-c S*›
‹*Tuple17-get-d* (*Tuple17.set-m aivdom S*) = *Tuple17-get-d S*›
‹*Tuple17-get-e* (*Tuple17.set-m aivdom S*) = *Tuple17-get-e S*›

‹*Tuple17-get-f* (*Tuple17.set-m aivdom S*) = *Tuple17-get-f S*›
‹*Tuple17-get-g* (*Tuple17.set-m aivdom S*) = *Tuple17-get-g S*›
‹*Tuple17-get-h* (*Tuple17.set-m aivdom S*) = *Tuple17-get-h S*›
‹*Tuple17-get-i* (*Tuple17.set-m aivdom S*) = *Tuple17-get-i S*›
‹*Tuple17-get-j* (*Tuple17.set-m aivdom S*) = *Tuple17-get-j S*›
‹*Tuple17-get-k* (*Tuple17.set-m aivdom S*) = *Tuple17-get-k S*›
‹*Tuple17-get-l* (*Tuple17.set-m aivdom S*) = *Tuple17-get-l S*›
‹*Tuple17-get-m* (*Tuple17.set-m aivdom S*) = *aivdom*›
‹*Tuple17-get-n* (*Tuple17.set-m aivdom S*) = *Tuple17-get-n S*›
‹*Tuple17-get-o* (*Tuple17.set-m aivdom S*) = *Tuple17-get-o S*›
‹*Tuple17-get-p* (*Tuple17.set-m aivdom S*) = *Tuple17-get-p S*›
‹*Tuple17-get-q* (*Tuple17.set-m aivdom S*) = *Tuple17-get-q S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple17-state-simp*]:
‹*Tuple17-get-a* (*Tuple17.set-n lcount S*) = *Tuple17-get-a S*›
‹*Tuple17-get-b* (*Tuple17.set-n lcount S*) = *Tuple17-get-b S*›
‹*Tuple17-get-c*(*Tuple17.set-n lcount S*) = *Tuple17-get-c S*›
‹*Tuple17-get-d* (*Tuple17.set-n lcount S*) = *Tuple17-get-d S*›
‹*Tuple17-get-e* (*Tuple17.set-n lcount S*) = *Tuple17-get-e S*›
‹*Tuple17-get-f* (*Tuple17.set-n lcount S*) = *Tuple17-get-f S*›
‹*Tuple17-get-g* (*Tuple17.set-n lcount S*) = *Tuple17-get-g S*›
‹*Tuple17-get-h* (*Tuple17.set-n lcount S*) = *Tuple17-get-h S*›
‹*Tuple17-get-i* (*Tuple17.set-n lcount S*) = *Tuple17-get-i S*›
‹*Tuple17-get-j* (*Tuple17.set-n lcount S*) = *Tuple17-get-j S*›
‹*Tuple17-get-k* (*Tuple17.set-n lcount S*) = *Tuple17-get-k S*›
‹*Tuple17-get-l* (*Tuple17.set-n lcount S*) = *Tuple17-get-l S*›
‹*Tuple17-get-m* (*Tuple17.set-n lcount S*) = *Tuple17-get-m S*›
‹*Tuple17-get-o* (*Tuple17.set-n lcount S*) = *Tuple17-get-o S*›
‹*Tuple17-get-p* (*Tuple17.set-n lcount S*) = *Tuple17-get-p S*›
‹*Tuple17-get-q* (*Tuple17.set-n lcount S*) = *Tuple17-get-q S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple17-state-simp*]:
‹*Tuple17-get-a* (*set-o lcount S*) = *Tuple17-get-a S*›
‹*Tuple17-get-b* (*set-o lcount S*) = *Tuple17-get-b S*›
‹*Tuple17-get-c*(*set-o lcount S*) = *Tuple17-get-c S*›
‹*Tuple17-get-d* (*set-o lcount S*) = *Tuple17-get-d S*›
‹*Tuple17-get-e* (*set-o lcount S*) = *Tuple17-get-e S*›
‹*Tuple17-get-f* (*set-o lcount S*) = *Tuple17-get-f S*›
‹*Tuple17-get-g* (*set-o lcount S*) = *Tuple17-get-g S*›
‹*Tuple17-get-h* (*set-o lcount S*) = *Tuple17-get-h S*›
‹*Tuple17-get-i* (*set-o lcount S*) = *Tuple17-get-i S*›
‹*Tuple17-get-j* (*set-o lcount S*) = *Tuple17-get-j S*›
‹*Tuple17-get-k* (*set-o lcount S*) = *Tuple17-get-k S*›
‹*Tuple17-get-l* (*set-o lcount S*) = *Tuple17-get-l S*›
‹*Tuple17-get-m* (*set-o lcount S*) = *Tuple17-get-m S*›
‹*Tuple17-get-n* (*set-o lcount S*) = *Tuple17-get-n S*›
‹*Tuple17-get-o* (*set-o lcount S*) = *lcount*›
‹*Tuple17-get-p* (*set-o lcount S*) = *Tuple17-get-p S*›
‹*Tuple17-get-q* (*Tuple17.set-o lcount S*) = *Tuple17-get-q S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple17-state-simp*]:
‹*Tuple17-get-a* (*set-p old-arena S*) = *Tuple17-get-a S*›
‹*Tuple17-get-b* (*set-p old-arena S*) = *Tuple17-get-b S*›

‹*Tuple17-get-c*(*set-p old-arena S*) = *Tuple17-get-c S*›
‹*Tuple17-get-d* (*set-p old-arena S*) = *Tuple17-get-d S*›
‹*Tuple17-get-e* (*set-p old-arena S*) = *Tuple17-get-e S*›
‹*Tuple17-get-f* (*set-p old-arena S*) = *Tuple17-get-f S*›
‹*Tuple17-get-g* (*set-p old-arena S*) = *Tuple17-get-g S*›
‹*Tuple17-get-h* (*set-p old-arena S*) = *Tuple17-get-h S*›
‹*Tuple17-get-i* (*set-p old-arena S*) = *Tuple17-get-i S*›
‹*Tuple17-get-j* (*set-p old-arena S*) = *Tuple17-get-j S*›
‹*Tuple17-get-k* (*set-p old-arena S*) = *Tuple17-get-k S*›
‹*Tuple17-get-l* (*set-p old-arena S*) = *Tuple17-get-l S*›
‹*Tuple17-get-m* (*set-p old-arena S*) = *Tuple17-get-m S*›
‹*Tuple17-get-n* (*set-p old-arena S*) = *Tuple17-get-n S*›
‹*Tuple17-get-o* (*set-p old-arena S*) = *Tuple17-get-o S*›
‹*Tuple17-get-p* (*set-p old-arena S*) = *old-arena*›
‹*Tuple17-get-q* (*Tuple17.set-p old-arena S*) = *Tuple17-get-q S*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemma [*tuple17-state-simp*]:
‹*Tuple17-get-a* (*set-q old-arena S*) = *Tuple17-get-a S*›
‹*Tuple17-get-b* (*set-q old-arena S*) = *Tuple17-get-b S*›
‹*Tuple17-get-c*(*set-q old-arena S*) = *Tuple17-get-c S*›
‹*Tuple17-get-d* (*set-q old-arena S*) = *Tuple17-get-d S*›
‹*Tuple17-get-e* (*set-q old-arena S*) = *Tuple17-get-e S*›
‹*Tuple17-get-f* (*set-q old-arena S*) = *Tuple17-get-f S*›
‹*Tuple17-get-g* (*set-q old-arena S*) = *Tuple17-get-g S*›
‹*Tuple17-get-h* (*set-q old-arena S*) = *Tuple17-get-h S*›
‹*Tuple17-get-i* (*set-q old-arena S*) = *Tuple17-get-i S*›
‹*Tuple17-get-j* (*set-q old-arena S*) = *Tuple17-get-j S*›
‹*Tuple17-get-k* (*set-q old-arena S*) = *Tuple17-get-k S*›
‹*Tuple17-get-l* (*set-q old-arena S*) = *Tuple17-get-l S*›
‹*Tuple17-get-m* (*set-q old-arena S*) = *Tuple17-get-m S*›
‹*Tuple17-get-n* (*set-q old-arena S*) = *Tuple17-get-n S*›
‹*Tuple17-get-o* (*set-q old-arena S*) = *Tuple17-get-o S*›
‹*Tuple17-get-p* (*Tuple17.set-q old-arena S*) = *Tuple17-get-p S*›
‹*Tuple17-get-q* (*set-q old-arena S*) = *old-arena*›
by (*solves* ‹*cases S*; *auto simp*:›)+

lemmas [*simp*] = *tuple17-state-simp*

named-theorems *tuple17-getters-setters* ‹*Definition of getters and setters*›

end
theory *IsaSAT-Mark*
imports
IsaSAT-Clauses
IsaSAT-Watch-List
IsaSAT-Trail
begin

Chapter 8

Clauses Encoded as Positions

We use represent the conflict in two data structures close to the one used by the most SAT solvers: We keep an array that represent the clause (for efficient iteration on the clause) and a "hash-table" to efficiently test if a literal belongs to the clause.

The first data structure is simply an array to represent the clause. This theory is only about the second data structure. We refine it from the clause (seen as a multiset) in two steps:

1. First, we represent the clause as a "hash-table", where the i-th position indicates *Some True* (respectively *Some False*, *None*) if *Pos i* is present in the clause (respectively *Neg i*, not at all). This allows to represent every not-tautological clause whose literals fits in the underlying array.

We use the first part in two different ways: once for the conflict, where we specialize it to include only information on the atoms and once in the marking structure.

This is the first level of the refinement. We tried a few different definitions (including a direct one, i.e., mapping a position to the inclusion in the set) but the inductive version turned out to the easiest one to use.

inductive *mset-as-position* :: ‹*bool option list* ⇒ *nat literal multiset* ⇒ *bool*› **where**
empty:
‹*mset-as-position* (*replicate n None*) {#}› |
add:
‹*mset-as-position xs′* (*add-mset L P*)›
if ‹*mset-as-position xs P*› **and** ‹*atm-of L* < *length xs*› **and** ‹*L* ∉# *P*› **and** ‹−*L* ∉# *P*› **and**
‹*xs′* = *xs*[*atm-of L* := *Some* (*is-pos L*)]›

lemma *mset-as-position-distinct-mset*:
‹*mset-as-position xs P* ⟹ *distinct-mset P*›
by (*induction rule*: *mset-as-position.induct*) *auto*

lemma *mset-as-position-atm-le-length*:
‹*mset-as-position xs P* ⟹ *L* ∈# *P* ⟹ *atm-of L* < *length xs*›
by (*induction rule*: *mset-as-position.induct*) (*auto simp*: *nth-list-update′ atm-of-eq-atm-of*)

lemma *mset-as-position-nth*:
‹*mset-as-position xs P* ⟹ *L* ∈# *P* ⟹ *xs* ! (*atm-of L*) = *Some* (*is-pos L*)›
by (*induction rule*: *mset-as-position.induct*)
(*auto simp*: *nth-list-update′ atm-of-eq-atm-of dest*: *mset-as-position-atm-le-length*)

lemma *mset-as-position-in-iff-nth*:

```
  ‹mset-as-position xs P ⟹ atm-of L < length xs ⟹ L ∈# P ⟷ xs ! (atm-of L) = Some (is-pos L)›
  by (induction rule: mset-as-position.induct)
    (auto simp: nth-list-update' atm-of-eq-atm-of is-pos-neg-not-is-pos
      dest: mset-as-position-atm-le-length)

lemma mset-as-position-tautology: ‹mset-as-position xs C ⟹ ¬tautology C›
  by (induction rule: mset-as-position.induct) (auto simp: tautology-add-mset)

lemma mset-as-position-right-unique:
  assumes
    map: ‹mset-as-position xs D› and
    map': ‹mset-as-position xs D'›
  shows ‹D = D'›
proof (rule distinct-set-mset-eq)
  show ‹distinct-mset D›
    using mset-as-position-distinct-mset[OF map] .
  show ‹distinct-mset D'›
    using mset-as-position-distinct-mset[OF map'] .
  show ‹set-mset D = set-mset D'›
    using mset-as-position-in-iff-nth[OF map] mset-as-position-in-iff-nth[OF map']
      mset-as-position-atm-le-length[OF map] mset-as-position-atm-le-length[OF map']
    by blast
qed

lemma mset-as-position-mset-union:
  fixes P xs
  defines ‹xs' ≡ fold (λL xs. xs[atm-of L := Some (is-pos L)]) P xs›
  assumes
    mset: ‹mset-as-position xs P'› and
    atm-P-xs: ‹∀ L ∈ set P. atm-of L < length xs› and
    uL-P: ‹∀ L ∈ set P. -L ∉# P'› and
    dist: ‹distinct P› and
    tauto: ‹¬tautology (mset P)›
  shows ‹mset-as-position xs' (mset P ∪# P')›
  using atm-P-xs uL-P dist tauto unfolding xs'-def
proof (induction P)
  case Nil
  show ?case using mset by auto
next
  case (Cons L P) note IH = this(1) and atm-P-xs = this(2) and uL-P = this(3) and dist = this(4)
    and tauto = this(5)
  then have mset:
    ‹mset-as-position (fold (λL xs. xs[atm-of L := Some (is-pos L)]) P xs) (mset P ∪# P')›
    by (auto simp: tautology-add-mset)
  show ?case
  proof (cases ‹L ∈# P'›)
    case True
    then show ?thesis
      using mset dist
      by (metis (no-types, lifting) add-mset-union assms(2) distinct.simps(2) fold-simps(2)
        insert-DiffM list-update-id mset.simps(2) mset-as-position-nth set-mset-mset
        sup-union-right1)
  next
    case False
    have [simp]: ‹length (fold (λL xs. xs[atm-of L := Some (is-pos L)]) P xs) = length xs›
      by (induction P arbitrary: xs) auto
```

```
    moreover have ‹- L ∉ set P›
      using tauto by (metis list.set-intros(1) list.set-intros(2) set-mset-mset tautology-minus)
    moreover have
      ‹fold (λL xs. xs[atm-of L := Some (is-pos L)]) P (xs[atm-of L := Some (is-pos L)]) =
       (fold (λL xs. xs[atm-of L := Some (is-pos L)]) P xs) [atm-of L := Some (is-pos L)]›
      using uL-P dist tauto
      apply (induction P arbitrary: xs)
      subgoal by auto
      subgoal for L' P
        by (cases ‹atm-of L = atm-of L'›)
          (auto simp: tautology-add-mset list-update-swap atm-of-eq-atm-of)
      done
    ultimately show ?thesis
      using mset atm-P-xs dist uL-P False by (auto intro!: mset-as-position.add)
  qed
qed

lemma mset-as-position-empty-iff: ‹mset-as-position xs {#} ⟷ (∃ n. xs = replicate n None)›
  apply (rule iffI)
  subgoal
    by (cases rule: mset-as-position.cases, assumption) auto
  subgoal
    by (auto intro: mset-as-position.intros)
  done

type-synonym (in -) lookup-clause-rel = ‹nat × bool option list›

definition lookup-clause-rel :: ‹nat multiset ⇒ (lookup-clause-rel × nat literal multiset) set› where
‹lookup-clause-rel A = {((n, xs), C). n = size C ∧ mset-as-position xs C ∧
   (∀ L∈atms-of (L_all A). L < length xs)}›

lemma lookup-clause-rel-empty-iff: ‹((n, xs), C) ∈ lookup-clause-rel A ⟹ n = 0 ⟷ C = {#}›
  by (auto simp: lookup-clause-rel-def)

lemma conflict-atm-le-length: ‹((n, xs), C) ∈ lookup-clause-rel A ⟹ L ∈ atms-of (L_all A) ⟹
   L < length xs›
  by (auto simp: lookup-clause-rel-def)


lemma conflict-le-length:
  assumes
    c-rel: ‹((n, xs), C) ∈ lookup-clause-rel A› and
    L-L_all: ‹L ∈# L_all A›
  shows ‹atm-of L < length xs›
proof -
  have
    size: ‹n = size C› and
    mset-pos: ‹mset-as-position xs C› and
    atms-le: ‹∀ L∈atms-of (L_all A). L < length xs›
    using c-rel unfolding lookup-clause-rel-def by blast+
  have ‹atm-of L ∈ atms-of (L_all A)›
    using L-L_all by (simp add: atms-of-def)
  then show ?thesis
    using atms-le by blast
qed
```

lemma *lookup-clause-rel-atm-in-iff*:
‹((*n*, *xs*), *C*) ∈ *lookup-clause-rel* $\mathcal{A}$ ⟹ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹ *L* ∈# *C* ⟷ *xs*!(*atm-of L*) = *Some* (*is-pos L*)›
by (*rule mset-as-position-in-iff-nth*)
(*auto simp*: *lookup-clause-rel-def atms-of-def*)

lemma
assumes
c: ‹((*n*,*xs*), *C*) ∈ *lookup-clause-rel* $\mathcal{A}$›
shows
lookup-clause-rel-not-tautolgy: ‹¬*tautology C*› **and**
lookup-clause-rel-distinct-mset: ‹*distinct-mset C*› **and**
lookup-clause-rel-size: ‹*isasat-input-bounded* $\mathcal{A}$ ⟹ *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ *C* ⟹ *size C* ≤ *1* + *unat32-max div 2*›
proof −
have *mset*: ‹*mset-as-position xs C*› **and** ‹*n* = *size C*› **and** ‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$). *L* < *length xs*›
using *c* **unfolding** *lookup-clause-rel-def* **by** *fast+*
show ‹¬*tautology C*›
using *mset*
apply (*induction rule*: *mset-as-position.induct*)
subgoal by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-def*)
subgoal by (*auto simp*: *tautology-add-mset*)
done
show ‹*distinct-mset C*›
using *mset mset-as-position-distinct-mset* **by** *blast*
then show ‹*isasat-input-bounded* $\mathcal{A}$ ⟹ *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ *C* ⟹ *size C* ≤ *1* + *unat32-max div 2*›
using *simple-clss-size-upper-div2*[*of* $\mathcal{A}$ ‹*C*›] ‹¬*tautology C*› **by** *auto*
qed

definition *option-bool-rel* :: ‹(*bool* × *'a option*) *set*› **where**
‹*option-bool-rel* = {(*b*, *x*). *b* ⟷ ¬(*is-None x*)}›

definition *NOTIN* :: ‹*bool option*› **where**
[*simp*]: ‹*NOTIN* = *None*›

definition *ISIN* :: ‹*bool* ⇒ *bool option*› **where**
[*simp*]: ‹*ISIN b* = *Some b*›

definition *is-NOTIN* :: ‹*bool option* ⇒ *bool*› **where**
[*simp*]: ‹*is-NOTIN x* ⟷ *x* = *NOTIN*›

lemma *is-NOTIN-alt-def*:
‹*is-NOTIN x* ⟷ *is-None x*›
by (*auto split*: *option.splits*)

lemma (**in** −) *mset-as-position-length-not-None*:
‹*mset-as-position x2 C* ⟹ *size C* = *length* (*filter* ((≠) *None*) *x2*)›
proof (*induction rule*: *mset-as-position.induct*)
case (*empty n*)
then show *?case* **by** *auto*
next
case (*add xs P L xs'*) **note** *m-as-p* = *this*(*1*) **and** *atm-L* = *this*(*2*)
have *xs-L*: ‹*xs* ! (*atm-of L*) = *None*›
proof −

```
    obtain bb :: ‹bool option ⇒ bool› where
      f1: ‹∀ z. z = None ∨ z = Some (bb z)›
      by (metis option.exhaust)
    have f2: ‹xs ! atm-of L ≠ Some (is-pos L)›
      using add.hyps(1) add.hyps(2) add.hyps(3) mset-as-position-in-iff-nth by blast
    have f3: ‹∀ z b. ((Some b = z ∨ z = None) ∨ bb z) ∨ b›
      using f1 by blast
    have f4: ‹∀ zs. (zs ! atm-of L ≠ Some (is-pos (− L)) ∨ ¬ atm-of L < length zs)
          ∨ ¬ mset-as-position zs P›
      by (metis add.hyps(4) atm-of-uminus mset-as-position-in-iff-nth)
    have ‹∀ z b. ((Some b = z ∨ z = None) ∨ ¬ bb z) ∨ ¬ b›
      using f1 by blast
    then show ?thesis
      using f4 f3 f2 by (metis add.hyps(1) add.hyps(2) is-pos-neg-not-is-pos)
  qed
  obtain xs1 xs2 where
    xs-xs12: ‹xs = xs1 @ None # xs2› and
    xs1: ‹length xs1 = atm-of L›
    using atm-L upd-conv-take-nth-drop[of ‹atm-of L› xs ‹None›] apply −
    apply (subst(asm)(2) xs-L[symmetric])
    by (force simp del: append-take-drop-id)+
  then show ?case
    using add by (auto simp: list-update-append)
qed


definition (in −) is-in-lookup-conflict where
  ‹is-in-lookup-conflict = (λ(n, xs) L. ¬is-None (xs ! atm-of L))›

lemma mset-as-position-remove:
  ‹mset-as-position xs D ⟹ L < length xs ⟹
   mset-as-position (xs[L := None]) (remove1-mset (Pos L) (remove1-mset (Neg L) D))›
proof (induction rule: mset-as-position.induct)
  case (empty n)
  then have [simp]: ‹(replicate n None)[L := None] = replicate n None›
    using list-update-id[of ‹replicate n None› L] by auto
  show ?case by (auto intro: mset-as-position.intros)
next
  case (add xs P K xs')
  show ?case
  proof (cases ‹L = atm-of K›)
    case True
    then show ?thesis
      using add by (cases K) auto
  next
    case False
    have map: ‹mset-as-position (xs[L := None]) (remove1-mset (Pos L) (remove1-mset (Neg L) P))›
      using add by auto
    have ‹K ∉# P − {#Pos L, Neg L#}› ‹−K ∉# P − {#Pos L, Neg L#}›
      by (auto simp: add.hyps dest!: in-diffD)
    then show ?thesis
      using mset-as-position.add[OF map, of ‹K› ‹xs[L := None, atm-of K := Some (is-pos K)]›]
        add False list-update-swap[of ‹atm-of K› L xs] apply simp
      apply (subst diff-add-mset-swap)
      by auto
  qed
```

qed

lemma *mset-as-position-remove2*:
‹*mset-as-position xs D* ⟹ *atm-of L* < *length xs* ⟹
mset-as-position (*xs*[*atm-of L* := *None*]) (*D* − {#*L*, −*L*#})›
using *mset-as-position-remove*[*of xs D* ‹*atm-of* (−*L*)›]
by (*smt add-mset-commute add-mset-diff-bothsides atm-of-uminus insert-DiffM literal.exhaust-sel minus-notin-trivial2 remove-1-mset-id-iff-notin uminus-not-id'*)

lemma *mset-as-position-remove3*:
‹*mset-as-position xs* (*D* − {#*L*#}) ⟹ *atm-of L* < *length xs* ⟹ *distinct-mset D* ⟹
mset-as-position (*xs*[*atm-of L* := *None*]) (*D* − {#*L*, −*L*#})›
using *mset-as-position-remove2*[*of xs* ‹*D* − {#*L*#}› ‹*L*›] *mset-as-position-tautology*[*of xs* ‹*remove1-mset L D*›]
mset-as-position-distinct-mset[*of xs* ‹*remove1-mset L D*›]
by (*cases* ‹*L* ∈# *D*›; *cases* ‹−*L* ∈# *D*›)
(*auto dest*!: *multi-member-split simp*: *minus-notin-trivial ac-simps add-mset-eq-add-mset tautology-add-mset*)

definition (**in** −) *delete-from-lookup-conflict*
:: ‹*nat literal* ⇒ *lookup-clause-rel* ⇒ *lookup-clause-rel nres*› **where**
‹*delete-from-lookup-conflict* = (λ*L* (*n*, *xs*). *do* {
ASSERT(*n*≥*1*);
ASSERT(*atm-of L* < *length xs*);
RETURN (*n* − *1*, *xs*[*atm-of L* := *None*])
})›

lemma *delete-from-lookup-conflict-op-mset-delete*:
‹(*uncurry delete-from-lookup-conflict*, *uncurry* (*RETURN oo remove1-mset*)) ∈
[λ(*L*, *D*). −*L* ∉# *D* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ *L* ∈# *D*]$_f$ *Id* ×$_f$ *lookup-clause-rel* $\mathcal{A}$ →
⟨*lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*›
apply (*intro frefI nres-relI*)
subgoal for *x y*
using *mset-as-position-remove*[*of* ‹*snd* (*snd x*)› ‹*snd y*› ‹*atm-of* (*fst y*)›]
apply (*cases x*; *cases y*; *cases* ‹*fst y*›)
by (*auto simp*: *delete-from-lookup-conflict-def lookup-clause-rel-def*
dest!: *multi-member-split*
intro!: *ASSERT-refine-left*)
done

definition *delete-from-lookup-conflict-pre* **where**
‹*delete-from-lookup-conflict-pre* $\mathcal{A}$ = (λ(*a*, *b*). − *a* ∉# *b* ∧ *a* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ *a* ∈# *b*)›

definition *add-to-lookup-conflict* :: ‹*nat literal* ⇒ *lookup-clause-rel* ⇒ *lookup-clause-rel*› **where**
‹*add-to-lookup-conflict* = (λ*L* (*n*, *xs*). (*if xs* ! *atm-of L* = *NOTIN then n* + *1 else n*,
xs[*atm-of L* := *ISIN* (*is-pos L*)]))›

lemma *add-to-lookup-conflict-lookup-clause-rel*:
assumes
confl: ‹((*n*, *xs*), *C*) ∈ *lookup-clause-rel* $\mathcal{A}$› **and**
uL-C: ‹−*L* ∉# *C*› **and**
L-$\mathcal{L}_{all}$: ‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
shows ‹(*add-to-lookup-conflict L* (*n*, *xs*), {#*L*#} ∪# *C*) ∈ *lookup-clause-rel* $\mathcal{A}$›

proof −
  have
    n: ‹*n* = *size C*› **and**
    mset: ‹*mset-as-position xs C*› **and**
    atm: ‹∀ *L*∈*atms-of* (*$\mathcal{L}_{all}$ $\mathcal{A}$*). *L* < *length xs*›
    using *confl* **unfolding** *lookup-clause-rel-def* **by** *blast*+
  have ‹*distinct-mset C*›
    using *mset* **by** (*blast dest*: *mset-as-position-distinct-mset*)
  have *atm-L-xs*: ‹*atm-of L* < *length xs*›
    using *atm L-$\mathcal{L}_{all}$* **by** (*auto simp*: *atms-of-def*)
  then show *?thesis*
  proof (*cases* ‹*L* ∈# *C*›)
    case *True*
    with *mset* **have** *xs*: ‹*xs* ! *atm-of L* = *Some* (*is-pos L*)› ‹*xs* ! *atm-of L* ≠ *None*›
      by (*auto dest*: *mset-as-position-nth*)
    moreover have ‹{#*L*#} ∪# *C* = *C*›
      using *True* **by** (*metis mset-contains-eq union-mset-def*)
    ultimately show *?thesis*
      using *n mset atm True*
      by (*auto simp*: *lookup-clause-rel-def add-to-lookup-conflict-def xs*[*symmetric*])
  next
    case *False*
    with *mset* **have** ‹*xs* ! *atm-of L* = *None*›
      using *mset-as-position-in-iff-nth*[*of xs C L*]
       mset-as-position-in-iff-nth[*of xs C* ‹−*L*›] *atm-L-xs uL-C*
      by (*cases L*; *cases* ‹*xs* ! *atm-of L*›) *auto*
    then show *?thesis*
      using *n mset atm False atm-L-xs uL-C*
      by (*auto simp*: *lookup-clause-rel-def add-to-lookup-conflict-def*
          intro!: *mset-as-position.intros*)
  qed
qed

definition *conflict-from-lookup* **where**
  ‹*conflict-from-lookup* = (λ(*n*, *xs*). *SPEC*(λ*D*. *mset-as-position xs D* ∧ *n* = *size D*))›

lemma *Ex-mset-as-position*:
  ‹*Ex* (*mset-as-position xs*)›
proof (*induction* ‹*size* {#*x* ∈# *mset xs*. *x* ≠ *None*#}› *arbitrary*: *xs*)
  case *0*
  then have *xs*: ‹*xs* = *replicate* (*length xs*) *None*›
    by (*auto simp*: *filter-mset-empty-conv dest*: *replicate-length-same*)
  show *?case*
    by (*subst xs*) (*auto simp*: *mset-as-position.empty intro*!: *exI*[*of* - ‹{#}›])
next
  case (*Suc x*) **note** *IH* = *this*(*1*) **and** *xs* = *this*(*2*)
  obtain *i* **where**
    [*simp*]: ‹*i* < *length xs*› **and**
    xs-i: ‹*xs* ! *i* ≠ *None*›
    using *xs*[*symmetric*]
    by (*auto dest*!: *size-eq-Suc-imp-elem simp*: *in-set-conv-nth*)
  let *?xs* = ‹*xs* [*i* := *None*]›
  have ‹*x* = *size* {#*x* ∈# *mset ?xs*. *x* ≠ *None*#}›
    using *xs*[*symmetric*] *xs-i* **by** (*auto simp*: *mset-update size-remove1-mset-If*)
  from *IH*[*OF this*] **obtain** *D* **where**

```
    map: ‹mset-as-position ?xs D›
    by blast
  have [simp]: ‹Pos i ∉# D› ‹Neg i ∉# D›
    using xs-i mset-as-position-nth[OF map, of ‹Pos i›]
      mset-as-position-nth[OF map, of ‹Neg i›]
    by auto
  have [simp]: ‹xs ! i = a ⟹ xs[i := a] = xs› for a
    by auto

  have ‹mset-as-position xs (add-mset (if the (xs ! i) then Pos i else Neg i) D)›
    using mset-as-position.add[OF map, of ‹if the (xs ! i) then Pos i else Neg i› xs]
      xs-i[symmetric]
    by (cases ‹xs ! i›) auto
  then show ?case by blast
qed

lemma id-conflict-from-lookup:
  ‹(RETURN o id, conflict-from-lookup) ∈ [λ(n, xs). ∃ D. ((n, xs), D) ∈ lookup-clause-rel 𝒜]_f Id →
    ⟨lookup-clause-rel 𝒜⟩nres-rel›
  by (intro frefI nres-relI)
    (auto simp: lookup-clause-rel-def conflict-from-lookup-def RETURN-RES-refine-iff)


lemma lookup-clause-rel-exists-le-unat32-max:
  assumes ocr: ‹((n, xs), D) ∈ lookup-clause-rel 𝒜› and ‹n > 0› and
    le-i: ‹∀ k<i. xs ! k = None› and lits: ‹literals-are-in-ℒ_in 𝒜 D› and
    bounded: ‹isasat-input-bounded 𝒜›
  shows
    ‹∃ j. j ≥ i ∧ j < length xs ∧ j < unat32-max ∧ xs ! j ≠ None›
proof −
  have
    n-D: ‹n = size D› and
    map: ‹mset-as-position xs D› and
    le-xs: ‹∀ L∈atms-of (ℒ_all 𝒜). L < length xs›
    using ocr unfolding lookup-clause-rel-def by auto
  have map-empty: ‹mset-as-position xs {#} ⟷ (xs = [] ∨ set xs = {None})›
    by (subst mset-as-position.simps) (auto simp add: list-eq-replicate-iff)
  have ex-not-none: ‹∃ j. j ≥ i ∧ j < length xs ∧ xs ! j ≠ None›
  proof (rule ccontr)
    assume ‹¬ ?thesis›
    then have ‹xs = [] ∨ set xs = {None}›
      using le-i by (fastforce simp: in-set-conv-nth)
    then have ‹mset-as-position xs {#}›
      using map-empty by auto
    then show False
      using mset-as-position-right-unique[OF map] ‹n > 0› n-D by (cases D) auto
  qed
  then obtain j where
    j: ‹j ≥ i›‹j < length xs›‹xs ! j ≠ None›
    by blast
  let ?L = ‹if the (xs ! j) then Pos j else Neg j›
  have ‹?L ∈# D›
    using j mset-as-position-in-iff-nth[OF map, of ?L] by auto
  then have ‹nat-of-lit ?L ≤ unat32-max›
    using lits bounded
    by (auto 5 5 dest!: multi-member-split[of - D]
```

```
      simp: literals-are-in-\mathcal{L}_{in}-add-mset split: if-splits)
  then have ‹j < unat32-max›
    by (auto simp: unat32-max-def split: if-splits)
  then show ?thesis
    using j by blast
qed

definition pre-simplify-clause-inv where
  ‹pre-simplify-clause-inv C = (λ(i, tauto, D, D'').
    i ≤ length C ∧ (¬tauto ⟷ ¬tautology (mset (take i C))) ∧
    (¬tauto ⟶ D = remdups-mset (mset (take i C))) ∧
    set D'' ⊆ set C ∧
    mset D'' = D ∧
    ¬tautology D ∧
    distinct-mset D)›

definition pre-simplify-clause :: ‹'v clause-l ⇒ (bool × 'v clause-l) nres› where
  ‹pre-simplify-clause C = do {
    (-, tauto, D₀, D) ←
      WHILE_T pre-simplify-clause-inv C
       (λ(i, tauto, D, D''). i < length C ∧ ¬tauto)
       (λ(i, tauto, D, D''). do {
         ASSERT(i < length C);
         let L = C!i;
         if -L ∈# D
         then RETURN (i+1,True, D, D'')
         else if L ∈# D
         then RETURN (i+1, tauto, D, D'')
         else RETURN (i+1,tauto, add-mset L D, D'' @ [L])
       })
       (0, False, {#}, []);
    ASSERT(D₀ = mset D ∧ set D ⊆ set C);
    RETURN (tauto, D)
  }›

definition pre-simplify-clause-spec where
  ‹pre-simplify-clause-spec C = (λ(tauto, D).
    (tauto ⟷ tautology (mset C)) ∧
    (¬tauto ⟶ mset D = remdups-mset (mset C)))›

lemma pre-simplify-clause-spec:
  ‹pre-simplify-clause C ≤ ⇓ Id (SPEC(pre-simplify-clause-spec C))›
proof -
  have [refine0]: ‹wf (measure (λ(i, -). length C - i))›
    by auto
  show ?thesis
    unfolding pre-simplify-clause-def
    apply refine-vcg
    subgoal by (auto simp: pre-simplify-clause-inv-def)
    subgoal by auto
    subgoal for state i b tauto ba D D''
      by (auto simp: pre-simplify-clause-inv-def
        take-Suc-conv-app-nth tautology-add-mset)
    subgoal
      by auto
    subgoal for state i b tauto ba D D''
```

```
    by (auto simp: pre-simplify-clause-inv-def
      take-Suc-conv-app-nth tautology-add-mset)
  subgoal
    by auto
  subgoal for state i b tauto ba D D'
    by (auto simp: pre-simplify-clause-inv-def
      take-Suc-conv-app-nth tautology-add-mset)
  subgoal
    by auto
  subgoal by (auto simp: pre-simplify-clause-inv-def)
  subgoal by (auto simp: pre-simplify-clause-inv-def)
  subgoal for state i b tauto ba D D'
    using not-tautology-mono[of ‹mset (take i C)› ‹mset C›]
    by (auto simp: pre-simplify-clause-inv-def mset-take-subseteq
      pre-simplify-clause-spec-def)
  done
qed

definition (in −) lit-is-in-lookup where
  ‹lit-is-in-lookup = (λL (n, xs). do {
    ASSERT(atm-of L < length xs);
    RETURN ((xs ! atm-of L) = Some (is-pos L))})›

definition unmark-clause :: ‹-› where
  ‹unmark-clause C lup = do {
    (-, lup) ← WHILE_T
      (λ(i,lup). i < length C)
      (λ(i,lup). do {
        ASSERT(i < length C);
        lup ← delete-from-lookup-conflict (C!i) lup;
        RETURN (i+1, lup)
      })
      (0, lup);
    RETURN lup
  }›

lemma unmark-clause-spec:
  assumes ‹(lup, mset C) ∈ lookup-clause-rel A› ‹atm-of ‘ set C ⊆ set-mset A›
  shows ‹unmark-clause C lup ≤ (SPEC(λlup'. (lup', {#}) ∈ lookup-clause-rel A))›
proof −
  have [refine]: ‹wf (measure (λ(i, -). length C − i))›
    by auto
  show ?thesis
    unfolding unmark-clause-def
    apply (refine-vcg WHILET-rule[where R = ‹measure (λ(i, -). length C − i)› and
      I = ‹λ(i,lup). (lup, mset (drop i C)) ∈ lookup-clause-rel A›])
    subgoal using assms by auto
    subgoal using assms by auto
    subgoal for s a b
      using lookup-clause-rel-not-tautolgy[of ‹fst b› ‹snd b› ‹mset (drop a C)›] assms(2) apply −
      apply (rule
        delete-from-lookup-conflict-op-mset-delete[of A, THEN fref-to-Down-curry,
          of ‹C!a› ‹mset (drop a C)›, THEN order-trans])
      by (auto simp: in-Lall-atm-of-Ain conc-fun-RES RETURN-def tautology-add-mset
        simp flip: Cons-nth-drop-Suc)
    subgoal
```

```
    by auto
  done
qed

lemma lit-is-in-lookup-spec:
  assumes ‹(lup, C) ∈ lookup-clause-rel 𝒜› ‹atm-of L ∈# 𝒜›
  shows ‹lit-is-in-lookup L lup = RES {L ∈# C}›
  using assms unfolding lit-is-in-lookup-def
  apply refine-rcg
  using mset-as-position-in-iff-nth[of ‹snd lup› C L]
  by (auto simp: lit-is-in-lookup-def lookup-clause-rel-def atms-of-ℒ_all-𝒜_in
    dest: multi-member-split
    dest: mset-as-position-nth)

definition pre-simplify-clause-lookup
  :: ‹nat clause-l ⇒ nat clause-l ⇒ lookup-clause-rel ⇒
    (bool × nat clause-l × lookup-clause-rel) nres›
where
  ‹pre-simplify-clause-lookup C D lup = do {
  ASSERT(D = []);
  (-, tauto, lup, D) ←
  WHILE_T λ-. True
    (λ(i, tauto, D, D'). i < length C ∧ ¬tauto)
    (λ(i, tauto, D, D'). do {
      ASSERT(i < length C);
      ASSERT(fst D < unat32-max ∧ atm-of (C!i) < length (snd D));
      ASSERT(length D' = fst D);
      let L = C!i;
      b ← lit-is-in-lookup (−L) D;
      if b
      then RETURN (i+1,True, D, D')
      else do {
        b ← lit-is-in-lookup L D;
        if b
        then RETURN (i+1, tauto, D, D')
        else RETURN (i+1,tauto, add-to-lookup-conflict L D, D' @ [L])
      }
    })
    (0, False, lup, D);
  lup ← unmark-clause D lup;
  RETURN (tauto, D, lup)
}›

lemma pre-simplify-clause-lookup-pre-simplify-clause:
  assumes ‹(lup, {#}) ∈ lookup-clause-rel 𝒜› ‹atm-of ‘ set C ⊆ set-mset 𝒜›
      ‹isasat-input-bounded 𝒜› and
    [simp]: ‹E = []›
  shows ‹pre-simplify-clause-lookup C E lup ≤
    ⇓{((tauto, D, lup), (tauto', D')). tauto=tauto' ∧ D=D' ∧ (lup, {#}) ∈ lookup-clause-rel 𝒜}
      (pre-simplify-clause C)›
proof −
  have [refine0]: ‹((0, False, lup, E), 0, False, {#}, []) ∈
     nat-rel ×_r bool-rel ×_r lookup-clause-rel 𝒜 ×_r ⟨Id⟩list-rel›
    using assms by auto
  have [simp]: ‹x < length C ⟹ atm-of (− C ! x) ∈# 𝒜›
    ‹x < length C ⟹ atm-of (C ! x) ∈# 𝒜› for x
```

```
    using assms by auto
  show ?thesis
    using assms
    unfolding pre-simplify-clause-lookup-def pre-simplify-clause-def
    apply (refine-vcg lit-is-in-lookup-spec lhs-step-If)
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal for x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e
      using simple-clss-size-upper-div2[of 𝒜 ‹mset x2b›] assms
      unfolding literals-are-in-ℒ_in-alt-def apply −
      by (subgoal-tac ‹atm-of ‘ set (take x1 C) ⊆ atms-of (ℒ_all 𝒜)›)
        (auto simp: lit-is-in-lookup-spec in-ℒ_all-atm-of-𝒜_in pre-simplify-clause-inv-def
        lookup-clause-rel-def unat32-max-def atms-of-ℒ_all-𝒜_in dest!: in-set-takeD)
    subgoal for x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e
      using simple-clss-size-upper-div2[of 𝒜 ‹mset x2b›] assms
      unfolding literals-are-in-ℒ_in-alt-def apply −
      by (subgoal-tac ‹atm-of ‘ set (take x1 C) ⊆ atms-of (ℒ_all 𝒜)›)
        (auto simp: lit-is-in-lookup-spec in-ℒ_all-atm-of-𝒜_in pre-simplify-clause-inv-def
        lookup-clause-rel-def unat32-max-def atms-of-ℒ_all-𝒜_in dest!: in-set-takeD)
    subgoal by (clarsimp simp add: lookup-clause-rel-def pre-simplify-clause-inv-def
      simp del: size-mset simp flip: size-mset)
    subgoal for x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e
      using add-to-lookup-conflict-lookup-clause-rel[of ‹fst x1e› ‹snd x1e› x1b 𝒜 ‹C ! x1›]
      by (auto simp: lit-is-in-lookup-spec in-ℒ_all-atm-of-𝒜_in)
    subgoal for x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e
      by (rule unmark-clause-spec[of - - 𝒜, THEN order-trans])
        auto
    done
qed


end
theory IsaSAT-Profile
  imports IsaSAT-Literals
begin
```

Chapter 9

Profiling

For profiling, we don't do anything except calling some C functions to measure time. As for printing, the functions to nothing in the refinement and are only removed from the generated code. The aim is to better understand the behaviour of the generated code and find performance bug.

context
begin
  qualified definition *start* :: ‹*8 word ⇒ unit*› **where** ‹*start a* = ()›
  qualified definition *stop* :: ‹*8 word ⇒ unit*› **where** ‹*stop a* = ()›
  qualified definition *PROPAGATE* :: ‹*8 word*› **where** ‹*PROPAGATE* = *0*›
  qualified definition *ANALYZE* :: ‹*8 word*› **where** ‹*ANALYZE* = *1*›
  qualified definition *GC* :: ‹*8 word*› **where** ‹*GC* = *2*›
  qualified definition *REDUCE* :: ‹*8 word*› **where** ‹*REDUCE* = *3*›
  qualified definition *INITIALISATION* :: ‹*8 word*› **where** ‹*INITIALISATION* = *4*›
  qualified definition *MINIMIZATION* :: ‹*8 word*› **where** ‹*MINIMIZATION* = *5*›
  qualified definition *SUBSUMPTION* :: ‹*8 word*› **where** ‹*SUBSUMPTION* = *6*›
  qualified definition *PURE-LITERAL* :: ‹*8 word*› **where** ‹*PURE-LITERAL* = *7*›
  qualified definition *BINARY-SIMP* :: ‹*8 word*› **where** ‹*BINARY-SIMP* = *8*›

qualified abbreviation *start-propagate* :: ‹*unit*› **where**
  ‹*start-propagate* ≡ *IsaSAT-Profile.start IsaSAT-Profile.PROPAGATE*›
qualified abbreviation *stop-propagate* :: ‹*unit*› **where**
  ‹*stop-propagate* ≡ *IsaSAT-Profile.stop IsaSAT-Profile.PROPAGATE*›
  qualified abbreviation *start-analyze* :: ‹*unit*› **where**
  ‹*start-analyze* ≡ *IsaSAT-Profile.start IsaSAT-Profile.ANALYZE*›
  qualified abbreviation *stop-analyze* :: ‹*unit*› **where**
  ‹*stop-analyze* ≡ *IsaSAT-Profile.stop IsaSAT-Profile.ANALYZE*›
qualified abbreviation *start-GC* :: ‹*unit*› **where**
  ‹*start-GC* ≡ *IsaSAT-Profile.start IsaSAT-Profile.GC*›
qualified abbreviation *stop-GC* :: ‹*unit*› **where**
  ‹*stop-GC* ≡ *IsaSAT-Profile.stop IsaSAT-Profile.GC*›

qualified abbreviation *start-reduce* :: ‹*unit*› **where**
  ‹*start-reduce* ≡ *IsaSAT-Profile.start IsaSAT-Profile.REDUCE*›
qualified abbreviation *stop-reduce* :: ‹*unit*› **where**
  ‹*stop-reduce* ≡ *IsaSAT-Profile.stop IsaSAT-Profile.REDUCE*›

qualified abbreviation *start-initialisation* :: ‹*unit*› **where**
  ‹*start-initialisation* ≡ *IsaSAT-Profile.start IsaSAT-Profile.INITIALISATION*›
qualified abbreviation *stop-initialisation* :: ‹*unit*› **where**
  ‹*stop-initialisation* ≡ *IsaSAT-Profile.stop IsaSAT-Profile.INITIALISATION*›

qualified abbreviation *start-minimization* :: ‹*unit*› **where**
‹*start-minimization* ≡ *IsaSAT-Profile.start IsaSAT-Profile.MINIMIZATION*›
qualified abbreviation *stop-minimization* :: ‹*unit*› **where**
‹*stop-minimization* ≡ *IsaSAT-Profile.stop IsaSAT-Profile.MINIMIZATION*›

qualified abbreviation *start-subsumption* :: ‹*unit*› **where**
‹*start-subsumption* ≡ *IsaSAT-Profile.start IsaSAT-Profile.SUBSUMPTION*›
qualified abbreviation *stop-subsumption* :: ‹*unit*› **where**
‹*stop-subsumption* ≡ *IsaSAT-Profile.stop IsaSAT-Profile.SUBSUMPTION*›

qualified abbreviation *start-binary-simp* :: ‹*unit*› **where**
‹*start-binary-simp* ≡ *IsaSAT-Profile.start IsaSAT-Profile.BINARY-SIMP*›
qualified abbreviation *stop-binary-simp* :: ‹*unit*› **where**
‹*stop-binary-simp* ≡ *IsaSAT-Profile.stop IsaSAT-Profile.BINARY-SIMP*›

qualified abbreviation *start-pure-literal* :: ‹*unit*› **where**
‹*start-pure-literal* ≡ *IsaSAT-Profile.start IsaSAT-Profile.PURE-LITERAL*›
qualified abbreviation *stop-pure-literal* :: ‹*unit*› **where**
‹*stop-pure-literal* ≡ *IsaSAT-Profile.stop IsaSAT-Profile.PURE-LITERAL*›

end

end
theory *IsaSAT-Lookup-Conflict*
imports
IsaSAT-Literals
Watched-Literals.CDCL-Conflict-Minimisation
LBD
IsaSAT-Clauses
IsaSAT-Watch-List
IsaSAT-Mark
IsaSAT-Profile
begin

We refine it to an array of booleans indicating if the atom is present or not. This information is redundant because we already know that a literal can only appear negated compared to the trail.

The first step makes it easier to reason about the clause (since we have the full clause), while the second step should generate (slightly) more efficient code.

Most solvers also merge the underlying array with the array used to cache information for the conflict minimisation (see theory *Watched-Literals.CDCL-Conflict-Minimisation*, where we only test if atoms appear in the clause, not literals).

As far as we know, versat stops at the first refinement (stating that there is no significant overhead, which is probably true, but the second refinement is not much additional work anyhow and we don't have to rely on the ability of the compiler to not represent the option type on booleans as a pointer, which it might be able to or not).

definition *option-lookup-clause-rel* **where**
‹*option-lookup-clause-rel* $\mathcal{A}$ = {((*b*,(*n*,*xs*)), *C*). *b* = (*C* = *None*) ∧
(*C* = *None* ⟶ ((*n*,*xs*), {#}) ∈ *lookup-clause-rel* $\mathcal{A}$) ∧
(*C* ≠ *None* ⟶ ((*n*,*xs*), *the C*) ∈ *lookup-clause-rel* $\mathcal{A}$)}
›

lemma *option-lookup-clause-rel-lookup-clause-rel-iff*:

‹((False, (n, xs)), Some C) ∈ option-lookup-clause-rel 𝒜 ⟷
((n, xs), C) ∈ lookup-clause-rel 𝒜›
unfolding *option-lookup-clause-rel-def* **by** *auto*

type-synonym (**in** −) *conflict-option-rel* = ‹*bool* × *nat* × *bool option list*›

definition (**in** −) *lookup-clause-assn-is-None* :: ‹- ⇒ *bool*› **where**
‹*lookup-clause-assn-is-None* = (λ(*b*, -, -). *b*)›

lemma *lookup-clause-assn-is-None-is-None*:
‹(*RETURN o lookup-clause-assn-is-None*, *RETURN o is-None*) ∈
option-lookup-clause-rel 𝒜 →$_f$ ⟨*bool-rel*⟩*nres-rel*›
by (*intro nres-relI frefI*)
(*auto simp*: *option-lookup-clause-rel-def lookup-clause-assn-is-None-def split*: *option.splits*)

definition (**in** −) *lookup-clause-assn-is-empty* :: ‹- ⇒ *bool*› **where**
‹*lookup-clause-assn-is-empty* = (λ(-, *n*, -). *n* = *0*)›

lemma *lookup-clause-assn-is-empty-is-empty*:
‹(*RETURN o lookup-clause-assn-is-empty*, *RETURN o* (λ*D*. *Multiset.is-empty*(*the D*))) ∈
[λ*D*. *D* ≠ *None*]$_f$ *option-lookup-clause-rel* 𝒜 → ⟨*bool-rel*⟩*nres-rel*›
by (*intro nres-relI frefI*)
(*auto simp*: *option-lookup-clause-rel-def lookup-clause-assn-is-empty-def lookup-clause-rel-def*
Multiset.is-empty-def split: *option.splits*)

lemma *option-lookup-clause-rel-update-None*:
assumes ‹((*False*, (*n*, *xs*)), *Some D*) ∈ *option-lookup-clause-rel* 𝒜› **and** *L-xs* : ‹*L* < *length xs*›
shows ‹((*False*, (*if xs*!*L* = *None then n else n* − *1*, *xs*[*L* := *None*])),
Some (*D* − {# *Pos L*, *Neg L* #}))) ∈ *option-lookup-clause-rel* 𝒜›
proof −
have [*simp*]: ‹*L* ∉# *A* ⟹ *A* − *add-mset L′* (*add-mset L B*) = *A* − *add-mset L′ B*›
for *A B* :: ‹*'a multiset*› **and** *L L′*
by (*metis add-mset-commute minus-notin-trivial*)
have ‹*n* = *size D*› **and** *map*: ‹*mset-as-position xs D*›
using *assms* **by** (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def*)
have *xs-None-iff*: ‹*xs* ! *L* = *None* ⟷ *Pos L* ∉# *D* ∧ *Neg L* ∉# *D*›
using *map L-xs mset-as-position-in-iff-nth*[*of xs D* ‹*Pos L*›]
mset-as-position-in-iff-nth[*of xs D* ‹*Neg L*›]
by (*cases* ‹*xs* ! *L*›) *auto*

have *1*: ‹*xs* ! *L* = *None* ⟹ *D* − {#*Pos L*, *Neg L*#} = *D*›
using *assms* **by** (*auto simp*: *xs-None-iff minus-notin-trivial*)
have *2*: ‹*xs* ! *L* = *None* ⟹ *xs*[*L* := *None*] = *xs*›
using *map list-update-id*[*of xs L*] **by** (*auto simp*: *1*)
have *3*: ‹*xs* ! *L* = *Some y* ⟷ (*y* ∧ *Pos L* ∈# *D* ∧ *Neg L* ∉# *D*) ∨ (¬*y* ∧ *Pos L* ∉# *D* ∧ *Neg L* ∈#
D)›
for *y*
using *map L-xs mset-as-position-in-iff-nth*[*of xs D* ‹*Pos L*›]
mset-as-position-in-iff-nth[*of xs D* ‹*Neg L*›]
by (*cases* ‹*xs* ! *L*›) *auto*

show *?thesis*
using *assms mset-as-position-remove*[*of xs D L*]
by (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def 1 2 3 size-remove1-mset-If*
minus-notin-trivial mset-as-position-remove)

qed

definition *size-lookup-conflict* :: ‹- ⇒ *nat*› **where**
‹*size-lookup-conflict* = (λ(-, *n*, -). *n*)›

definition *size-conflict-wl-heur* :: ‹- ⇒ *nat*› **where**
‹*size-conflict-wl-heur* = (λ(*M*, *N*, *U*, *D*, -, -, -, -). *size-lookup-conflict D*)›

definition (**in** −) *is-in-conflict* :: ‹*nat literal* ⇒ *nat clause option* ⇒ *bool*› **where**
[*simp*]: ‹*is-in-conflict L C* ⟷ *L* ∈# *the C*›

definition (**in** −) *is-in-lookup-option-conflict*
:: ‹*nat literal* ⇒ (*bool* × *nat* × *bool option list*) ⇒ *bool*›
where
‹*is-in-lookup-option-conflict* = (λ*L* (-, -, *xs*). *xs* ! *atm-of L* = *Some* (*is-pos L*))›

lemma *is-in-lookup-option-conflict-is-in-conflict*:
‹(*uncurry* (*RETURN oo is-in-lookup-option-conflict*),
uncurry (*RETURN oo is-in-conflict*)) ∈
[λ(*L*, *C*). *C* ≠ *None* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$ *Id* ×$_r$ *option-lookup-clause-rel* $\mathcal{A}$ →
⟨*Id*⟩*nres-rel*›
apply (*intro nres-relI frefI*)
subgoal for *Lxs LC*
using *lookup-clause-rel-atm-in-iff*[*of* - ‹*snd* (*snd* (*snd Lxs*))›]
apply (*cases Lxs*)
by (*auto simp*: *is-in-lookup-option-conflict-def option-lookup-clause-rel-def*)
done

definition *set-conflict-m*
:: ‹(*nat*, *nat*) *ann-lits* ⇒ *nat clauses-l* ⇒ *nat* ⇒ *nat clause option* ⇒ *nat* ⇒
out-learned ⇒ (*nat clause option* × *nat* × *out-learned*) *nres*›
where
‹*set-conflict-m M N i* - - - =
SPEC (λ(*C*, *n*, *outl*). *C* = *Some* (*mset* (*N*∝*i*)) ∧ *n* = *card-max-lvl M* (*mset* (*N*∝*i*)) ∧
out-learned M C outl)›

definition *merge-conflict-m*
:: ‹(*nat*, *nat*) *ann-lits* ⇒ *nat clauses-l* ⇒ *nat* ⇒ *nat clause option* ⇒ *nat* ⇒
out-learned ⇒ (*nat clause option* × *nat* × *out-learned*) *nres*›
where
‹*merge-conflict-m M N i D* - - =
SPEC (λ(*C*, *n*, *outl*). *C* = *Some* (*mset* (*tl* (*N*∝*i*)) ∪# *the D*) ∧
n = *card-max-lvl M* (*mset* (*tl* (*N*∝*i*)) ∪# *the D*) ∧
out-learned M C outl)›

definition *merge-conflict-m-g*
:: ‹*nat* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *nat clause-l* ⇒ *nat clause option* ⇒
(*nat clause option* × *nat* × *out-learned*) *nres*›
where
‹*merge-conflict-m-g init′ M Ni D* =
SPEC (λ(*C*, *n*, *outl*). *C* = *Some* (*mset* (*drop init′* (*Ni*)) ∪# *the D*) ∧
n = *card-max-lvl M* (*mset* (*drop init′* (*Ni*)) ∪# *the D*) ∧
out-learned M C outl)›

definition *outlearned-add*
:: ‹(*nat*,*nat*)*ann-lits* ⇒ *nat literal* ⇒ *nat* × *bool option list* ⇒ *out-learned* ⇒ *out-learned*› **where**
‹*outlearned-add* = (λ*M L zs outl*.
(*if get-level M L* < *count-decided M* ∧ ¬*is-in-lookup-conflict zs L then outl* @ [*L*]
else outl))›

definition *clvls-add*
:: ‹(*nat*,*nat*)*ann-lits* ⇒ *nat literal* ⇒ *nat* × *bool option list* ⇒ *nat* ⇒ *nat*› **where**
‹*clvls-add* = (λ*M L zs clvls*.
(*if get-level M L* = *count-decided M* ∧ ¬*is-in-lookup-conflict zs L then clvls* + *1*
else clvls))›

definition *lookup-conflict-merge*
:: ‹*nat* ⇒ (*nat*,*nat*)*ann-lits* ⇒ *nat clause-l* ⇒ *conflict-option-rel* ⇒ *nat* ⇒
out-learned ⇒ (*conflict-option-rel* × *nat* × *out-learned*) *nres*›
where
‹*lookup-conflict-merge init′ M D* = (λ(*b*, *xs*) *clvls outl*. *do* {
(-, *clvls*, *zs*, *outl*) ← *WHILE*$_T^{\lambda(i::nat,\ clvls::nat,\ zs,\ outl).}$ *length* (*snd zs*) = *length* (*snd xs*) ∧ *Suc i* ≤ *una*
(λ(*i* :: *nat*, *clvls*, *zs*, *outl*). *i* < *length-uint32-nat D*)
(λ(*i* :: *nat*, *clvls*, *zs*, *outl*). *do* {
ASSERT(*i* < *length-uint32-nat D*);
ASSERT(*Suc i* ≤ *unat32-max*);
ASSERT(¬*is-in-lookup-conflict zs* (*D*!*i*) ⟶ *length outl* < *unat32-max*);
let outl = *outlearned-add M* (*D*!*i*) *zs outl*;
let clvls = *clvls-add M* (*D*!*i*) *zs clvls*;
let zs = *add-to-lookup-conflict* (*D*!*i*) *zs*;
RETURN(*Suc i*, *clvls*, *zs*, *outl*)
})
(*init′*, *clvls*, *xs*, *outl*);
RETURN ((*False*, *zs*), *clvls*, *outl*)
})›

definition *lookup-conflict-merge′-step*
:: ‹*nat multiset* ⇒ *nat* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *nat* ⇒ *nat* ⇒ *lookup-clause-rel* ⇒ *nat clause-l* ⇒
nat clause ⇒ *out-learned* ⇒ *bool*›
where
‹*lookup-conflict-merge′-step* $\mathcal{A}$ *init′ M i clvls zs D C outl* = (
let D′ = *mset* (*take* (*i* − *init′*) (*drop init′ D*));
E = *remdups-mset* (*D′* + *C*) *in*
((*False*, *zs*), *Some E*) ∈ *option-lookup-clause-rel* $\mathcal{A}$ ∧
out-learned M (*Some E*) *outl* ∧
literals-are-in-$\mathcal{L}_{in}$ $\mathcal{A}$ *E* ∧ *clvls* = *card-max-lvl M E*)›

definition *resolve-lookup-conflict-aa*
:: ‹(*nat*,*nat*)*ann-lits* ⇒ *nat clauses-l* ⇒ *nat* ⇒ *conflict-option-rel* ⇒ *nat* ⇒
out-learned ⇒ (*conflict-option-rel* × *nat* × *out-learned*) *nres*›
where
‹*resolve-lookup-conflict-aa M N i xs clvls outl* =
lookup-conflict-merge 1 M (*N* ∝ *i*) *xs clvls outl*›

definition *set-lookup-conflict-aa*
:: ‹(*nat*,*nat*)*ann-lits* ⇒ *nat clauses-l* ⇒ *nat* ⇒ *conflict-option-rel* ⇒ *nat* ⇒
out-learned ⇒(*conflict-option-rel* × *nat* × *out-learned*) *nres*›
where

‹*set-lookup-conflict-aa M C i xs clvls outl* =
lookup-conflict-merge 0 M (*C*∝*i*) *xs clvls outl*›

definition *isa-outlearned-add*
:: ‹*trail-pol* ⇒ *nat literal* ⇒ *nat* × *bool option list* ⇒ *out-learned* ⇒ *out-learned*› **where**
‹*isa-outlearned-add* = (λ*M L zs outl*.
(*if get-level-pol M L* < *count-decided-pol M* ∧ ¬*is-in-lookup-conflict zs L then outl* @ [*L*]
else outl))›

lemma *isa-outlearned-add-outlearned-add*:
‹(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$ ⟹ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹
isa-outlearned-add M' L zs outl = *outlearned-add M L zs outl*›
by (*auto simp*: *isa-outlearned-add-def outlearned-add-def get-level-get-level-pol*
count-decided-trail-ref[*THEN fref-to-Down-unRET-Id*])

definition *isa-clvls-add*
:: ‹*trail-pol* ⇒ *nat literal* ⇒ *nat* × *bool option list* ⇒ *nat* ⇒ *nat*› **where**
‹*isa-clvls-add* = (λ*M L zs clvls*.
(*if get-level-pol M L* = *count-decided-pol M* ∧ ¬*is-in-lookup-conflict zs L then clvls* + 1
else clvls))›

lemma *isa-clvls-add-clvls-add*:
‹(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$ ⟹ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹
isa-clvls-add M' L zs outl = *clvls-add M L zs outl*›
by (*auto simp*: *isa-clvls-add-def clvls-add-def get-level-get-level-pol*
count-decided-trail-ref[*THEN fref-to-Down-unRET-Id*])

definition *isa-lookup-conflict-merge*
:: ‹*nat* ⇒ *trail-pol* ⇒ *arena* ⇒ *nat* ⇒ *conflict-option-rel* ⇒ *nat* ⇒
out-learned ⇒ (*conflict-option-rel* × *nat* × *out-learned*) *nres*›
where
‹*isa-lookup-conflict-merge init' M N i* = (λ(*b*, *xs*) *clvls outl*. *do* {
ASSERT(*arena-is-valid-clause-idx N i*);
(-, *clvls*, *zs*, *outl*) ← $WHILE_T^{\lambda(i::nat,\ clvls::nat,\ zs,\ outl).}$ *length* (*snd zs*) = *length* (*snd xs*) ∧ *Suc* (*fst zs*)
(λ(*j* :: *nat*, *clvls*, *zs*, *outl*). *j* < *i* + *arena-length N i*)
(λ(*j* :: *nat*, *clvls*, *zs*, *outl*). *do* {
ASSERT(*j* < *length N*);
ASSERT(*arena-lit-pre N j*);
ASSERT(*get-level-pol-pre* (*M*, *arena-lit N j*));
ASSERT(*get-level-pol M* (*arena-lit N j*) ≤ *Suc* (*unat32-max div 2*));
ASSERT(*atm-of* (*arena-lit N j*) < *length* (*snd zs*));
ASSERT(¬*is-in-lookup-conflict zs* (*arena-lit N j*) ⟶ *length outl* < *unat32-max*);
let outl = *isa-outlearned-add M* (*arena-lit N j*) *zs outl*;
let clvls = *isa-clvls-add M* (*arena-lit N j*) *zs clvls*;
let zs = *add-to-lookup-conflict* (*arena-lit N j*) *zs*;
RETURN(*Suc j*, *clvls*, *zs*, *outl*)
})
(*i*+*init'*, *clvls*, *xs*, *outl*);
RETURN ((*False*, *zs*), *clvls*, *outl*)
})›

lemma *isa-lookup-conflict-merge-lookup-conflict-merge-ext*:
assumes *valid*: ‹*valid-arena arena N vdom*› **and** *i*: ‹*i* ∈# *dom-m N*› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf N*)› **and**
bxs: ‹((*b*, *xs*), *C*) ∈ *option-lookup-clause-rel* $\mathcal{A}$› **and**

M'M: ‹(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$› **and**
bound: ‹*isasat-input-bounded* $\mathcal{A}$›
shows
‹*isa-lookup-conflict-merge init' M' arena i* (*b*, *xs*) *clvls outl* ≤ ⇓ *Id*
(*lookup-conflict-merge init' M* (*N* ∝ *i*) (*b*, *xs*) *clvls outl*)›
proof −
have [*refine0*]: ‹((*i* + *init'*, *clvls*, *xs*, *outl*), *init'*, *clvls*, *xs*, *outl*) ∈
{(*k*, *l*). *k* = *l* + *i*} ×$_r$ *nat-rel* ×$_r$ *Id* ×$_r$ *Id*›
by *auto*
have ‹*no-dup M*›
using *assms* **by** (*auto simp*: *trail-pol-def*)
have ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *M*›
using *M'M* **by** (*auto simp*: *trail-pol-def literals-are-in-*$\mathcal{L}_{in}$*-trail-def*)
from *literals-are-in-*$\mathcal{L}_{in}$*-trail-get-level-unat32-max*[*OF bound this* ‹*no-dup M*›]
have *lev-le*: ‹*get-level M L* ≤ *Suc* (*unat32-max div 2*)› **for** *L* **.**

show *?thesis*
unfolding *isa-lookup-conflict-merge-def lookup-conflict-merge-def prod.case*
apply *refine-vcg*
subgoal using *assms* **unfolding** *arena-is-valid-clause-idx-def* **by** *fast*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *valid i* **by** (*auto simp*: *arena-lifting*)
subgoal using *valid i* **by** (*auto simp*: *arena-lifting ac-simps*)
subgoal using *valid i*
by (*auto simp*: *arena-lifting arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
intro!: *exI*[*of* - *i*])
subgoal for *x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e*
using *i literals-are-in-*$\mathcal{L}_{in}$*-mm-in-*$\mathcal{L}_{all}$[*of* $\mathcal{A}$ *N i x1*] *lits valid M'M*
by (*auto simp*: *arena-lifting ac-simps image-image intro*!: *get-level-pol-pre*)
subgoal for *x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e*
using *valid i literals-are-in-*$\mathcal{L}_{in}$*-mm-in-*$\mathcal{L}_{all}$[*of* $\mathcal{A}$ *N i x1*] *lits*
by (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def*
in-$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff arena-lifting ac-simps get-level-get-level-pol*[*OF M'M*, *symmetric*]
isa-outlearned-add-outlearned-add[*OF M'M*] *isa-clvls-add-clvls-add*[*OF M'M*] *lev-le*)
subgoal for *x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e*
using *i literals-are-in-*$\mathcal{L}_{in}$*-mm-in-*$\mathcal{L}_{all}$[*of* $\mathcal{A}$ *N i x1*] *lits valid M'M*
using *bxs* **by** (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def*
in-$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff arena-lifting ac-simps*)
subgoal for *x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e*
using *valid i literals-are-in-*$\mathcal{L}_{in}$*-mm-in-*$\mathcal{L}_{all}$[*of* $\mathcal{A}$ *N i x1*] *lits*
by (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def*
in-$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff arena-lifting ac-simps get-level-get-level-pol*[*OF M'M*]
isa-outlearned-add-outlearned-add[*OF M'M*] *isa-clvls-add-clvls-add*[*OF M'M*])
subgoal for *x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e*
using *valid i literals-are-in-*$\mathcal{L}_{in}$*-mm-in-*$\mathcal{L}_{all}$[*of* $\mathcal{A}$ *N i x1*] *lits*
by (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def*
in-$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff arena-lifting ac-simps get-level-get-level-pol*[*OF M'M*]
isa-outlearned-add-outlearned-add[*OF M'M*] *isa-clvls-add-clvls-add*[*OF M'M*])
subgoal using *bxs* **by** (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def*
in-$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff get-level-get-level-pol*[*OF M'M*])
done
qed

lemma (**in** −) *arena-is-valid-clause-idx-le-unat64-max*:

‹*arena-is-valid-clause-idx be bd* ⟹
length be ≤ *unat64-max* ⟹
bd + *arena-length be bd* ≤ *unat64-max*›
‹*arena-is-valid-clause-idx be bd* ⟹ *length be* ≤ *unat64-max* ⟹
bd ≤ *unat64-max*›
using *arena-lifting*(*10*)[*of be - - bd*]
by (*fastforce simp*: *arena-lifting arena-is-valid-clause-idx-def*)+

definition *isa-set-lookup-conflict-aa* **where**
‹*isa-set-lookup-conflict-aa* = *isa-lookup-conflict-merge 0*›

definition *isa-set-lookup-conflict-aa-pre* **where**
‹*isa-set-lookup-conflict-aa-pre* =
(λ(((((*M*, *N*), *i*), (-, *xs*)), -), *out*). *i* < *length N*)›

lemma *lookup-conflict-merge′-spec*:
assumes
o: ‹((*b*, *n*, *xs*), *Some C*) ∈ *option-lookup-clause-rel* 𝒜› **and**
dist: ‹*distinct D*› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ 𝒜 (*mset D*)› **and**
tauto: ‹¬*tautology* (*mset D*)› **and**
lits-C: ‹*literals-are-in-*$\mathcal{L}_{in}$ 𝒜 *C*› **and**
‹*clvls* = *card-max-lvl M C*› **and**
CD: ‹⋀*L*. *L* ∈ *set* (*drop init′ D*) ⟹ −*L* ∉# *C*› **and**
‹*Suc init′* ≤ *unat32-max*› **and**
‹*out-learned M* (*Some C*) *outl*› **and**
bounded: ‹*isasat-input-bounded* 𝒜›
shows
‹*lookup-conflict-merge init′ M D* (*b*, *n*, *xs*) *clvls outl* ≤
⇓(*option-lookup-clause-rel* 𝒜 ×$_r$ *Id* ×$_r$ *Id*)
(*merge-conflict-m-g init′ M D* (*Some C*))›
(**is** ‹- ≤ ⇓ *?Ref ?Spec*›)
proof −
let *?D* = ‹*drop init′ D*›
have *le-D-le-upper*[*simp*]: ‹*a* < *length D* ⟹ *Suc* (*Suc a*) ≤ *unat32-max*› **for** *a*
using *simple-clss-size-upper-div2*[*of* 𝒜 ‹*mset D*›] *assms* **by** (*auto simp*: *unat32-max-def*)
have *Suc-N-unat32-max*: ‹*Suc n* ≤ *unat32-max*› **and**
size-C-unat32-max: ‹*size C* ≤ *1* + *unat32-max div 2*› **and**
clvls: ‹*clvls* = *card-max-lvl M C*› **and**
tauto-C: ‹¬ *tautology C*› **and**
dist-C: ‹*distinct-mset C*› **and**
atms-le-xs: ‹∀ *L*∈*atms-of* (*$\mathcal{L}_{all}$* 𝒜). *L* < *length xs*› **and**
map: ‹*mset-as-position xs C*›
using *assms simple-clss-size-upper-div2*[*of* 𝒜 *C*] *mset-as-position-distinct-mset*[*of xs C*]
lookup-clause-rel-not-tautolgy[*of n xs C*] *bounded*
unfolding *option-lookup-clause-rel-def lookup-clause-rel-def*
by (*auto simp*: *unat32-max-def*)
then have *clvls-unat32-max*: ‹*clvls* ≤ *1* + *unat32-max div 2*›
using *size-filter-mset-lesseq*[*of* ‹λ*L*. *get-level M L* = *count-decided M*› *C*]
unfolding *unat32-max-def card-max-lvl-def* **by** *linarith*
have [*intro*]: ‹((*b*, *a*, *ba*), *Some C*) ∈ *option-lookup-clause-rel* 𝒜 ⟹ *literals-are-in-*$\mathcal{L}_{in}$ 𝒜 *C* ⟹
Suc (*Suc a*) ≤ *unat32-max*› **for** *b a ba C*
using *lookup-clause-rel-size*[*of a ba C*, *OF - bounded*] **by** (*auto simp*: *option-lookup-clause-rel-def*
lookup-clause-rel-def unat32-max-def)
have [*simp*]: ‹*remdups-mset C* = *C*›

using *o mset-as-position-distinct-mset*[*of xs C*] **by** (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def distinct-mset-remdups-mset-id*)
have ‹¬*tautology C*›
using *mset-as-position-tautology o* **by** (*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def*)
have ‹*distinct-mset C*›
using *mset-as-position-distinct-mset*[*of - C*] *o*
unfolding *option-lookup-clause-rel-def lookup-clause-rel-def* **by** *auto*
let *?C'* = ‹λ*a*. (*mset* (*take* (*a* − *init'*) (*drop init' D*)) + *C*)›
have *tauto-C'*: ‹¬ *tautology* (*?C' a*)› **if** ‹*a* ≥ *init'*› **for** *a*
using *that tauto tauto-C dist dist-C CD* **unfolding** *tautology-decomp'*
by (*force dest*: *in-set-takeD in-diffD dest*: *in-set-dropD simp*: *distinct-mset-in-diff in-image-uminus-uminus*)

define *I* **where**
‹*I xs* = (λ(*i*, *clvls*, *zs* :: *lookup-clause-rel*, *outl* :: *out-learned*).
length (*snd zs*) =
length (*snd xs*) ∧
Suc i ≤ *unat32-max* ∧
Suc (*fst zs*) ≤ *unat32-max* ∧
Suc clvls ≤ *unat32-max*)›
for *xs* :: *lookup-clause-rel*
define *I'* **where** ‹*I'* = (λ(*i*, *clvls*, *zs*, *outl*).
lookup-conflict-merge'-step 𝒜 *init' M i clvls zs D C outl* ∧ *i* ≥ *init'*)›

have *dist-D*: ‹*distinct-mset* (*mset D*)›
using *dist* **by** *auto*
have *dist-CD*: ‹*distinct-mset* (*C* − *mset D* − *uminus* '# *mset D*)›
using ‹*distinct-mset C*› **by** *auto*
have [*simp*]: ‹*remdups-mset* (*mset* (*drop init' D*) + *C*) = *mset* (*drop init' D*) ∪# *C*›
apply (*rule distinct-mset-rempdups-union-mset*[*symmetric*])
using *dist-C dist* **by** *auto*
have ‹*literals-are-in-*$\mathcal{L}_{in}$ 𝒜 (*mset* (*take* (*a* − *init'*) (*drop init' D*)) ∪# *C*)› **for** *a*
using *lits-C lits* **by** (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-def all-lits-of-m-def dest*!: *in-set-takeD in-set-dropD*)
then have *size-outl-le*: ‹*size* (*mset* (*take* (*a* − *init'*) (*drop init' D*)) ∪# *C*) ≤ *Suc unat32-max div 2*›
if ‹*a* ≥ *init'*› **for** *a*
using *simple-clss-size-upper-div2*[*OF bounded*, *of* ‹*mset* (*take* (*a* − *init'*) (*drop init' D*)) ∪# *C*›] *tauto-C'*[*OF that*] ‹*distinct-mset C*› *dist-D*
by (*auto simp*: *unat32-max-def*)

have
if-True-I: ‹*I x2* (*Suc a*, *clvls-add M* (*D* ! *a*) *baa aa*,
add-to-lookup-conflict (*D* ! *a*) *baa*,
outlearned-add M (*D* ! *a*) *baa outl*)› (**is** *?I*) **and**
if-true-I': ‹*I'* (*Suc a*, *clvls-add M* (*D* ! *a*) *baa aa*,
add-to-lookup-conflict (*D* ! *a*) *baa*,
outlearned-add M (*D* ! *a*) *baa outl*)› (**is** *?I'*)
if
I: ‹*I x2 s*› **and**
I': ‹*I' s*› **and**
cond: ‹*case s of* (*i*, *clvls*, *zs*, *outl*) ⇒ *i* < *length D*› **and**
s: ‹*s* = (*a*, *ba*)› ‹*ba* = (*aa*, *baa2*)› ‹*baa2* = (*baa*, *outl*)› ‹(*b*, *n*, *xs*) = (*x1*, *x2*)› **and**
a-le-D: ‹*a* < *length D*› **and**
a-unat32-max: ‹*Suc a* ≤ *unat32-max*›
for *x1 x2 s a ba aa baa baa2 lbd' lbdL' outl x*

proof −
have [*simp*]:
‹*s* = (*a*, *aa*, *baa*, *outl*)›
‹*ba* = (*aa*, *baa*, *outl*)›
‹*x2* = (*n*, *xs*)›
using *s* **by** *auto*
obtain *ab b* **where** *baa*[*simp*]: ‹*baa* = (*ab*, *b*)› **by** (*cases baa*)

have *aa*: ‹*aa* = *card-max-lvl M* (*remdups-mset* (*?C′ a*))› **and**
ocr: ‹((*False*, *ab*, *b*), *Some* (*remdups-mset* (*?C′ a*))) ∈ *option-lookup-clause-rel* 𝒜› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ 𝒜 (*remdups-mset* (*?C′ a*))› **and**
outl: ‹*out-learned M* (*Some* (*remdups-mset* (*?C′ a*))) *outl*›
using *I′*
unfolding *I′-def lookup-conflict-merge′-step-def Let-def*
by *auto*
have
ab: ‹*ab* = *size* (*remdups-mset* (*?C′ a*))› **and**
map: ‹*mset-as-position b* (*remdups-mset* (*?C′ a*))› **and**
‹∀ *L*∈*atms-of* ($\mathcal{L}_{all}$ 𝒜). *L* < *length b*› **and**
cr: ‹((*ab*, *b*), *remdups-mset* (*?C′ a*)) ∈ *lookup-clause-rel* 𝒜›
using *ocr* **unfolding** *option-lookup-clause-rel-def lookup-clause-rel-def*
by *auto*
have *a-init*: ‹*a* ≥ *init′*›
using *I′* **unfolding** *I′-def* **by** *auto*
have ‹*size* (*card-max-lvl M* (*remdups-mset* (*?C′ a*))) ≤ *size* (*remdups-mset* (*?C′ a*))›
unfolding *card-max-lvl-def*
by *auto*
then have [*simp*]: ‹*Suc* (*Suc aa*) ≤ *unat32-max*› ‹*Suc aa* ≤ *unat32-max*›
using *size-C-unat32-max lits a-init*
simple-clss-size-upper-div2[*of* 𝒜 ‹*remdups-mset* (*?C′ a*)›, *OF bounded*]
unfolding *unat32-max-def aa*[*symmetric*]
by (*auto simp*: *tauto-C′*)
have [*simp*]: ‹*length b* = *length xs*›
using *I* **unfolding** *I-def* **by** *simp-all*

have *ab-upper*: ‹*Suc* (*Suc ab*) ≤ *unat32-max*›
using *simple-clss-size-upper-div2*[*OF bounded*, *of* ‹*remdups-mset* (*?C′ a*)›]
lookup-clause-rel-not-tautolgy[*OF cr*] *a-le-D lits mset-as-position-distinct-mset*[*OF map*]
unfolding *ab literals-are-in-*$\mathcal{L}_{in}$*-remdups unat32-max-def* **by** *auto*
show *?I*
using *le-D-le-upper a-le-D ab-upper a-init*
unfolding *I-def add-to-lookup-conflict-def baa clvls-add-def* **by** *auto*

have *take-Suc-a*[*simp*]: ‹*take* (*Suc a* − *init′*) *?D* = *take* (*a* − *init′*) *?D* @ [*D* ! *a*]›
by (*smt Suc-diff-le a-init a-le-D append-take-drop-id diff-less-mono drop-take-drop-drop*
length-drop same-append-eq take-Suc-conv-app-nth take-hd-drop)
have [*simp*]: ‹*D* ! *a* ∉ *set* (*take* (*a* − *init′*) *?D*)›
using *dist tauto a-le-D* **apply** (*subst* (*asm*) *append-take-drop-id*[*symmetric*, *of* - ‹*Suc a* − *init′*›],
subst append-take-drop-id[*symmetric*, *of* - ‹*Suc a* − *init′*›])
apply (*subst* (*asm*) *distinct-append*, *subst nth-append*)
by (*auto simp*: *in-set-distinct-take-drop-iff*)
have [*simp*]: ‹− *D* ! *a* ∉ *set* (*take* (*a* − *init′*) *?D*)›
proof
assume ‹− *D* ! *a* ∈ *set* (*take* (*a* − *init′*) (*drop init′ D*))›
then have ‹(*if is-pos* (*D* ! *a*) *then Neg else Pos*) (*atm-of* (*D* ! *a*)) ∈ *set D*›
by (*metis* (*no-types*) *in-set-dropD in-set-takeD uminus-literal-def*)

```
    then show False
      using a-le-D tauto by force
  qed

  have D-a-notin: ‹D ! a ∉# (mset (take (a − init′) ?D) + uminus ‘# mset (take (a − init′) ?D))›
    by (auto simp: uminus-lit-swap[symmetric])
  have uD-a-notin: ‹−D ! a ∉# (mset (take (a − init′) ?D) + uminus ‘# mset (take (a − init′) ?D))›
    by (auto simp: uminus-lit-swap[symmetric])

  show ?I′
  proof (cases ‹(get-level M (D ! a) = count-decided M ∧ ¬ is-in-lookup-conflict baa (D ! a))›)
    case if-cond: True
    have [simp]: ‹D ! a ∉# C› ‹−D ! a ∉# C› ‹b ! atm-of (D ! a) = None›
      using if-cond mset-as-position-nth[OF map, of ‹D ! a›]
        if-cond mset-as-position-nth[OF map, of ‹−D ! a›] D-a-notin uD-a-notin
      by (auto simp: is-in-lookup-conflict-def split: option.splits bool.splits
          dest: in-diffD)
    have [simp]: ‹atm-of (D ! a) < length xs› ‹D ! a ∈# \mathcal{L}_{all} \mathcal{A}›
      using literals-are-in-\mathcal{L}_{in}-in-\mathcal{L}_{all}[OF ‹literals-are-in-\mathcal{L}_{in} \mathcal{A} (mset D)› a-le-D] atms-le-xs
      by (auto simp: in-\mathcal{L}_{all}-atm-of-in-atms-of-iff)

    have ocr: ‹((False, add-to-lookup-conflict (D ! a) (ab, b)), Some (remdups-mset (?C′ (Suc a))))
      ∈ option-lookup-clause-rel \mathcal{A}›
      using ocr D-a-notin uD-a-notin
      unfolding option-lookup-clause-rel-def lookup-clause-rel-def add-to-lookup-conflict-def
      by (auto dest: in-diffD simp: minus-notin-trivial
          intro!: mset-as-position.intros)
     have ‹out-learned M (Some (remdups-mset (?C′ (Suc a)))) (outlearned-add M (D ! a) (ab, b)
outl)›
      using D-a-notin uD-a-notin ocr lits if-cond a-init outl
      unfolding outlearned-add-def out-learned-def
      by auto
    then show ?I′
      using D-a-notin uD-a-notin ocr lits if-cond a-init
      unfolding I′-def lookup-conflict-merge′-step-def Let-def clvls-add-def
      by (auto simp: minus-notin-trivial literals-are-in-\mathcal{L}_{in}-add-mset
          card-max-lvl-add-mset aa)
  next
    case if-cond: False
    have atm-D-a-le-xs: ‹atm-of (D ! a) < length xs› ‹D ! a ∈# \mathcal{L}_{all} \mathcal{A}›
      using literals-are-in-\mathcal{L}_{in}-in-\mathcal{L}_{all}[OF ‹literals-are-in-\mathcal{L}_{in} \mathcal{A} (mset D)› a-le-D] atms-le-xs
      by (auto simp: in-\mathcal{L}_{all}-atm-of-in-atms-of-iff)
    have [simp]: ‹D ! a ∉# C − add-mset (− D ! a)
          (add-mset (D ! a)
            (mset (take a D) + uminus ‘# mset (take a D)))›
      using dist-C in-diffD[of ‹D ! a› C ‹add-mset (− D ! a)
            (mset (take a D) + uminus ‘# mset (take a D))›,
        THEN multi-member-split]
      by (meson distinct-mem-diff-mset member-add-mset)
    have a-init: ‹a ≥ init′›
      using I′ unfolding I′-def by auto
    have take-Suc-a[simp]: ‹take (Suc a − init′) ?D = take (a − init′) ?D @ [D ! a]›
      by (smt Suc-diff-le a-init a-le-D append-take-drop-id diff-less-mono drop-take-drop-drop
          length-drop same-append-eq take-Suc-conv-app-nth take-hd-drop)
    have [iff]: ‹D ! a ∉ set (take (a − init′) ?D)›
      using dist tauto a-le-D
```

```
  apply (subst (asm) append-take-drop-id[symmetric, of - ‹Suc a − init'›],
      subst append-take-drop-id[symmetric, of - ‹Suc a − init'›])
  apply (subst (asm) distinct-append, subst nth-append)
  by (auto simp: in-set-distinct-take-drop-iff)
have [simp]: ‹− D ! a ∉ set (take (a − init') ?D)›
proof
  assume ‹− D ! a ∈ set (take (a − init') (drop init' D))›
  then have ‹(if is-pos (D ! a) then Neg else Pos) (atm-of (D ! a)) ∈ set D›
    by (metis (no-types) in-set-dropD in-set-takeD uminus-literal-def)
  then show False
    using a-le-D tauto by force
qed
have ‹D ! a ∈ set (drop init' D)›
  using a-init a-le-D by (meson in-set-drop-conv-nth)
from CD[OF this] have [simp]: ‹−D ! a ∉# C› .
consider
  (None) ‹b ! atm-of (D ! a) = None› |
  (Some-in) i where ‹b ! atm-of (D ! a) = Some i› and
  ‹(if i then Pos (atm-of (D ! a)) else Neg (atm-of (D ! a))) ∈# C›
  using if-cond mset-as-position-in-iff-nth[OF map, of ‹D ! a›]
    if-cond mset-as-position-in-iff-nth[OF map, of ‹−D ! a›] atm-D-a-le-xs(1)
  by  (cases ‹b ! atm-of (D ! a)›) (auto simp: is-pos-neg-not-is-pos)
then have ocr: ‹((False, add-to-lookup-conflict (D ! a) (ab, b)),
 Some (remdups-mset (?C' (Suc a)))) ∈ option-lookup-clause-rel A›
proof cases
  case [simp]: None
  have [simp]: ‹D ! a ∉# C›
    using if-cond mset-as-position-nth[OF map, of ‹D ! a›]
      if-cond mset-as-position-nth[OF map, of ‹−D ! a›]
    by (auto simp: is-in-lookup-conflict-def split: option.splits bool.splits
        dest: in-diffD)
  have [simp]: ‹atm-of (D ! a) < length xs› ‹D ! a ∈# L_all A›
    using literals-are-in-L_in-in-L_all[OF ‹literals-are-in-L_in A (mset D)› a-le-D] atms-le-xs
    by (auto simp: in-L_all-atm-of-in-atms-of-iff)

  show ocr: ‹((False, add-to-lookup-conflict (D ! a) (ab, b)),
    Some (remdups-mset (?C' (Suc a)))) ∈ option-lookup-clause-rel A›
    using ocr
    unfolding option-lookup-clause-rel-def lookup-clause-rel-def add-to-lookup-conflict-def
    by (auto dest: in-diffD simp: minus-notin-trivial
        intro!: mset-as-position.intros)
next
  case Some-in
  then have ‹remdups-mset (?C' a) = remdups-mset (?C' (Suc a))›
    using if-cond mset-as-position-in-iff-nth[OF map, of ‹D ! a›] a-init
      if-cond mset-as-position-in-iff-nth[OF map, of ‹−D ! a›] atm-D-a-le-xs(1)
    by (auto simp: is-neg-neg-not-is-neg)
  moreover
  have 1: ‹Some i = Some (is-pos (D ! a))›
    using if-cond mset-as-position-in-iff-nth[OF map, of ‹D ! a›] a-init Some-in
      if-cond mset-as-position-in-iff-nth[OF map, of ‹−D ! a›] atm-D-a-le-xs(1)
      ‹D ! a ∉ set (take (a − init') ?D)› ‹−D ! a ∉# C›
      ‹− D ! a ∉ set (take (a − init') ?D)›
    by (cases ‹D ! a›) (auto simp: is-neg-neg-not-is-neg)
  moreover have ‹b[atm-of (D ! a) := Some i] = b›
    unfolding 1[symmetric] Some-in(1)[symmetric] by simp
```

```
      ultimately show ?thesis
        using dist-C atms-le-xs Some-in(1) map
        unfolding option-lookup-clause-rel-def lookup-clause-rel-def add-to-lookup-conflict-def ab
        by (auto simp: distinct-mset-in-diff minus-notin-trivial
            intro: mset-as-position.intros
            simp del: remdups-mset-singleton-sum)
    qed
    have notin-lo-in-C: ‹¬is-in-lookup-conflict (ab, b) (D ! a) ⟹ D ! a ∉# C›
      using mset-as-position-in-iff-nth[OF map, of ‹Pos (atm-of (D!a))›]
        mset-as-position-in-iff-nth[OF map, of ‹Neg (atm-of (D!a))›] atm-D-a-le-xs(1)
        ‹- D ! a ∉ set (take (a - init') (drop init' D))›
        ‹D ! a ∉ set (take (a - init') (drop init' D))›
        ‹-D ! a ∉# C› a-init
      by (cases ‹b ! (atm-of (D ! a))›; cases ‹D ! a›)
        (auto simp: is-in-lookup-conflict-def dist-C distinct-mset-in-diff
          split: option.splits bool.splits
          dest: in-diffD)
    have in-lo-in-C: ‹is-in-lookup-conflict (ab, b) (D ! a) ⟹ D ! a ∈# C›
      using mset-as-position-in-iff-nth[OF map, of ‹Pos (atm-of (D!a))›]
        mset-as-position-in-iff-nth[OF map, of ‹Neg (atm-of (D!a))›] atm-D-a-le-xs(1)
        ‹- D ! a ∉ set (take (a - init') (drop init' D))›
        ‹D ! a ∉ set (take (a - init') (drop init' D))›
        ‹-D ! a ∉# C› a-init
      by (cases ‹b ! (atm-of (D ! a))›; cases ‹D ! a›)
        (auto simp: is-in-lookup-conflict-def dist-C distinct-mset-in-diff
          split: option.splits bool.splits
          dest: in-diffD)
    moreover have ‹out-learned M (Some (remdups-mset (?C' (Suc a))))
      (outlearned-add M (D ! a) (ab, b) outl)›
      using D-a-notin uD-a-notin ocr lits if-cond a-init outl in-lo-in-C notin-lo-in-C
      unfolding outlearned-add-def out-learned-def
      by auto
    ultimately show ?I'
      using ocr lits if-cond atm-D-a-le-xs a-init
      unfolding I'-def lookup-conflict-merge'-step-def Let-def clvls-add-def
      by (auto simp: minus-notin-trivial literals-are-in-\mathcal{L}_{in}-add-mset
          card-max-lvl-add-mset aa)
  qed
qed
have uL-C-if-L-C: ‹-L ∉# C› if ‹L ∈# C› for L
  using tauto-C that unfolding tautology-decomp' by blast

have outl-le: ‹length bc < unat32-max›
  if
    ‹I x2 s› and
    ‹I' s› and
    ‹s = (a, ba)› and
    ‹ba = (aa, baa)› and
    ‹baa = (ab, bc)› for x1 x2 s a ba aa baa ab bb ac bc
proof -
  have ‹mset (tl bc) ⊆# (remdups-mset (mset (take (a - init') (drop init' D)) + C))› and ‹init' ≤
a›
    using that by (auto simp: I-def I'-def lookup-conflict-merge'-step-def Let-def out-learned-def)
  from size-mset-mono[OF this(1)] this(2) show ?thesis using size-outl-le[of a] dist-C dist-D
    by (auto simp: unat32-max-def distinct-mset-rempdups-union-mset)
qed
```

show *confl*: ‹*lookup-conflict-merge init′ M D* (*b*, *n*, *xs*) *clvls outl*
≤ ⇓ *?Ref* (*merge-conflict-m-g init′ M D* (*Some C*))›
supply [[*goals-limit=1*]]
unfolding *resolve-lookup-conflict-aa-def lookup-conflict-merge-def*
distinct-mset-rempdups-union-mset[*OF dist-D dist-CD*] *I-def*[*symmetric*] *conc-fun-SPEC*
Let-def length-uint32-nat-def merge-conflict-m-g-def
apply (*refine-vcg WHILEIT-rule-stronger-inv*[**where** *R* = ‹*measure* (λ(*j*, -). *length D* − *j*)› **and**
I′ = *I′*])
subgoal by *auto*
subgoal
using *clvls-unat32-max Suc-N-unat32-max* ‹*Suc init′* ≤ *unat32-max*›
unfolding *unat32-max-def I-def* **by** *auto*
subgoal using *assms*
unfolding *lookup-conflict-merge′-step-def Let-def option-lookup-clause-rel-def I′-def*
by (*auto simp add*: *unat32-max-def lookup-conflict-merge′-step-def option-lookup-clause-rel-def*)
subgoal by *auto*
subgoal unfolding *I-def* **by** *fast*
subgoal for *x1 x2 s a ba aa baa ab bb* **by** (*rule outl-le*)
subgoal by (*rule if-True-I*)
subgoal by (*rule if-true-I′*)
subgoal for *b′ n′ s j zs*
using *dist lits tauto*
by (*auto simp*: *option-lookup-clause-rel-def take-Suc-conv-app-nth*
literals-are-in-$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$)
subgoal using *assms* **by** (*auto simp*: *option-lookup-clause-rel-def lookup-conflict-merge′-step-def*
Let-def I-def I′-def)
done
qed

lemma *literals-are-in-*$\mathcal{L}_{in}$*-mm-literals-are-in-*$\mathcal{L}_{in}$:
assumes *lits*: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf N*)› **and**
i: ‹*i* ∈# *dom-m N*›
shows ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*N* ∝ *i*))›
unfolding *literals-are-in-*$\mathcal{L}_{in}$*-def*
proof (*standard*)
fix *L*
assume ‹*L* ∈# *all-lits-of-m* (*mset* (*N* ∝ *i*))›
then have ‹*atm-of L* ∈ *atms-of-mm* (*mset* ‘# *ran-mf N*)›
using *i* **unfolding** *ran-m-def in-all-lits-of-m-ain-atms-of-iff*
by (*auto dest*!: *multi-member-split*)
then show ‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
using *lits atm-of-notin-atms-of-iff in-all-lits-of-mm-ain-atms-of-iff*
unfolding *literals-are-in-*$\mathcal{L}_{in}$*-mm-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*
by *blast*
qed

lemma *isa-set-lookup-conflict*:
‹(*uncurry5 isa-set-lookup-conflict-aa*, *uncurry5 set-conflict-m*) ∈
[λ(((((*M*, *N*), *i*), *xs*), *clvls*), *outl*). *i* ∈# *dom-m N* ∧ *xs* = *None* ∧ *distinct* (*N* ∝ *i*) ∧
literals-are-in-$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf N*) ∧
¬*tautology* (*mset* (*N* ∝ *i*)) ∧ *clvls* = *0* ∧
out-learned M None outl ∧
isasat-input-bounded $\mathcal{A}$]$_f$
trail-pol $\mathcal{A}$ ×$_f$ {(*arena*, *N*). *valid-arena arena N vdom*} ×$_f$ *nat-rel* ×$_f$
option-lookup-clause-rel $\mathcal{A}$ ×$_f$ *nat-rel* ×$_f$ *Id* →
⟨*option-lookup-clause-rel* $\mathcal{A}$ ×$_r$ *nat-rel* ×$_r$ *Id*⟩*nres-rel*›

proof −
have *H*: ‹*set-lookup-conflict-aa M N i* (*b*, *n*, *xs*) *clvls outl*
≤ ⇓ (*option-lookup-clause-rel* $\mathcal{A}$ ×$_r$ *Id*)
(*set-conflict-m M N i None clvls outl*)›
if
i: ‹*i* ∈# *dom-m N*› **and**
ocr: ‹((*b*, *n*, *xs*), *None*) ∈ *option-lookup-clause-rel* $\mathcal{A}$› **and**
dist: ‹*distinct* (*N* ∝ *i*)› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* '# *ran-mf N*)› **and**
tauto: ‹¬*tautology* (*mset* (*N* ∝ *i*))› **and**
‹*clvls* = *0*› **and**
out: ‹*out-learned M None outl*› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
for *b n xs N i M clvls lbd outl*
proof −
have *lookup-conflict-merge-normalise*:
‹*lookup-conflict-merge 0 M C* (*b*, *zs*) = *lookup-conflict-merge 0 M C* (*False*, *zs*)›
for *M C zs*
unfolding *lookup-conflict-merge-def* **by** *auto*
have [*simp*]: ‹*out-learned M* (*Some* {#}) *outl*›
using *out* **by** (*cases outl*) (*auto simp*: *out-learned-def*)
have *T*: ‹((*False*, *n*, *xs*), *Some* {#}) ∈ *option-lookup-clause-rel* $\mathcal{A}$›
using *ocr* **unfolding** *option-lookup-clause-rel-def* **by** *auto*
have ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*N* ∝ *i*))›
using *literals-are-in-*$\mathcal{L}_{in}$*-mm-literals-are-in-*$\mathcal{L}_{in}$[*OF lits i*] **.**
then show *?thesis* **unfolding** *set-lookup-conflict-aa-def set-conflict-m-def*
using *lookup-conflict-merge'-spec*[*of False n xs* ‹{#}› $\mathcal{A}$ ‹*N*∝*i*› *0 - 0 outl*] *that dist T*
by (*auto simp*: *lookup-conflict-merge-normalise unat32-max-def merge-conflict-m-g-def*)
qed

have *H*: ‹*isa-set-lookup-conflict-aa M' arena i* (*b*, *n*, *xs*) *clvls outl*
≤ ⇓ (*option-lookup-clause-rel* $\mathcal{A}$ ×$_r$ *Id*)
(*set-conflict-m M N i None clvls outl*)›
if
i: ‹*i* ∈# *dom-m N*› **and**
ocr: ‹((*b*, *n*, *xs*), *None*) ∈ *option-lookup-clause-rel* $\mathcal{A}$› **and**
dist: ‹*distinct* (*N* ∝ *i*)› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* '# *ran-mf N*)› **and**
tauto: ‹¬*tautology* (*mset* (*N* ∝ *i*))› **and**
‹*clvls* = *0*› **and**
out: ‹*out-learned M None outl*› **and**
valid: ‹*valid-arena arena N vdom*› **and**
M'M: ‹(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
for *b n xs N i M clvls lbd outl arena vdom M'*
unfolding *isa-set-lookup-conflict-aa-def*
apply (*rule order.trans*)
apply (*rule isa-lookup-conflict-merge-lookup-conflict-merge-ext*[*OF valid i lits ocr M'M bounded*])
unfolding *lookup-conflict-merge-def*[*symmetric*] *set-lookup-conflict-aa-def*[*symmetric*]
by (*auto intro*: *H*[*OF that*(*1*−*7*,*10*)])
show *?thesis*
unfolding *lookup-conflict-merge-def uncurry-def*
by (*intro nres-relI frefI*) (*auto intro*!: *H*)
qed

definition *merge-conflict-m-pre* **where**

‹*merge-conflict-m-pre* $\mathcal{A}$ =
(λ(((((*M*, *N*), *i*), *xs*), *clvls*), *out*). *i* ∈# *dom-m N* ∧ *xs* ≠ *None* ∧ *distinct* (*N* ∝ *i*) ∧
¬*tautology* (*mset* (*N* ∝ *i*)) ∧
(∀ *L* ∈ *set* (*tl* (*N* ∝ *i*)). − *L* ∉# *the xs*) ∧
literals-are-in-$\mathcal{L}_{in}$ $\mathcal{A}$ (*the xs*) ∧ *clvls* = *card-max-lvl M* (*the xs*) ∧
out-learned M xs out ∧ *no-dup M* ∧
literals-are-in-$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf N*) ∧
isasat-input-bounded $\mathcal{A}$)›

definition *isa-resolve-merge-conflict-gt2* **where**
‹*isa-resolve-merge-conflict-gt2* = *isa-lookup-conflict-merge 1*›

lemma *isa-resolve-merge-conflict-gt2*:
‹(*uncurry5 isa-resolve-merge-conflict-gt2*, *uncurry5 merge-conflict-m*) ∈
[*merge-conflict-m-pre* $\mathcal{A}$]$_f$
trail-pol $\mathcal{A}$ ×$_f$ {(*arena*, *N*). *valid-arena arena N vdom*} ×$_f$ *nat-rel* ×$_f$ *option-lookup-clause-rel* $\mathcal{A}$
×$_f$ *nat-rel* ×$_f$ *Id* →
⟨*option-lookup-clause-rel* $\mathcal{A}$ ×$_r$ *nat-rel* ×$_r$ *Id*⟩*nres-rel*›
proof −
have *H1*: ‹*resolve-lookup-conflict-aa M N i* (*b*, *n*, *xs*) *clvls outl*
≤ ⇓ (*option-lookup-clause-rel* $\mathcal{A}$ ×$_r$ *Id*)
(*merge-conflict-m M N i C clvls outl*)›
if
i: ‹*i* ∈# *dom-m N*› **and**
ocr: ‹((*b*, *n*, *xs*), *C*) ∈ *option-lookup-clause-rel* $\mathcal{A}$› **and**
dist: ‹*distinct* (*N* ∝ *i*)› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf N*)› **and**
lits′: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*the C*)› **and**
tauto: ‹¬*tautology* (*mset* (*N* ∝ *i*))› **and**
out: ‹*out-learned M C outl*› **and**
not-neg: ‹⋀*L*. *L* ∈ *set* (*tl* (*N* ∝ *i*)) ⟹ − *L* ∉# *the C*› **and**
‹*clvls* = *card-max-lvl M* (*the C*)› **and**
C-None: ‹*C* ≠ *None*› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
for *b n xs N i M clvls outl C*
proof −
have *lookup-conflict-merge-normalise*:
‹*lookup-conflict-merge 1 M C* (*b*, *zs*) = *lookup-conflict-merge 1 M C* (*False*, *zs*)›
for *M C zs*
unfolding *lookup-conflict-merge-def* **by** *auto*
have ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*N* ∝ *i*))›
using *literals-are-in-*$\mathcal{L}_{in}$*-mm-literals-are-in-*$\mathcal{L}_{in}$[*OF lits i*] .
then show *?thesis* **unfolding** *resolve-lookup-conflict-aa-def merge-conflict-m-def*
using *lookup-conflict-merge′-spec*[*of b n xs* ‹*the C*› $\mathcal{A}$ ‹*N*∝*i*› *clvls M 1 outl*] *that dist*
not-neg ocr C-None lits′
by (*auto simp*: *lookup-conflict-merge-normalise unat32-max-def merge-conflict-m-g-def*
drop-Suc)
qed

have *H2*: ‹*isa-resolve-merge-conflict-gt2 M′ arena i* (*b*, *n*, *xs*) *clvls outl*
≤ ⇓ (*Id* ×$_r$ *Id*)
(*resolve-lookup-conflict-aa M N i* (*b*, *n*, *xs*) *clvls outl*)›
if
i: ‹*i* ∈# *dom-m N*› **and**
ocr: ‹((*b*, *n*, *xs*), *C*) ∈ *option-lookup-clause-rel* $\mathcal{A}$› **and**
dist: ‹*distinct* (*N* ∝ *i*)› **and**

lits: ‹*literals-are-in-$\mathcal{L}_{in}$-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf N*)› **and**
lits': ‹*literals-are-in-$\mathcal{L}_{in}$* $\mathcal{A}$ (*the C*)› **and**
tauto: ‹¬*tautology* (*mset* (*N* ∝ *i*))› **and**
out: ‹*out-learned M C outl*› **and**
not-neg: ‹⋀*L*. *L* ∈ *set* (*tl* (*N* ∝ *i*)) ⟹ − *L* ∉# *the C*› **and**
‹*clvls* = *card-max-lvl M* (*the C*)› **and**
C-None: ‹*C* ≠ *None*› **and**
valid: ‹*valid-arena arena N vdom*› **and**

i: ‹*i* ∈# *dom-m N*› **and**
dist: ‹*distinct* (*N* ∝ *i*)› **and**
lits: ‹*literals-are-in-$\mathcal{L}_{in}$-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf N*)› **and**
tauto: ‹¬*tautology* (*mset* (*N* ∝ *i*))› **and**
‹*clvls* = *card-max-lvl M* (*the C*)› **and**
out: ‹*out-learned M C outl*› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$› **and**
M'M: ‹(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$›
for *b n xs N i M clvls lbd outl arena vdom C M'*
unfolding *isa-resolve-merge-conflict-gt2-def*
apply (*rule order.trans*)
apply (*rule isa-lookup-conflict-merge-lookup-conflict-merge-ext*[*OF valid i lits ocr M'M*])
unfolding *resolve-lookup-conflict-aa-def*[*symmetric*] *set-lookup-conflict-aa-def*[*symmetric*]
using *bounded* **by** (*auto intro*: *H1*[*OF that*(*1*−*6*)])
show *?thesis*
unfolding *lookup-conflict-merge-def uncurry-def*
apply (*intro nres-relI frefI*)
apply *clarify*
subgoal
unfolding *merge-conflict-m-pre-def*
apply (*rule order-trans*)
apply (*rule H2*; *auto*; *auto*; *fail*)
by (*auto intro*!: *H1 simp*: *merge-conflict-m-pre-def*)
done
qed

During the conflict analysis, the literal of highest level is at the beginning. During the rest of the time the conflict is *None*.

definition *highest-lit* **where**
‹*highest-lit M C L* ⟷
(*L* = *None* ⟶ *C* = {#}) ∧
(*L* ≠ *None* ⟶ *get-level M* (*fst* (*the L*)) = *snd* (*the L*) ∧
snd (*the L*) = *get-maximum-level M C* ∧
fst (*the L*) ∈# *C*
)›

Conflict Minimisation **definition** *iterate-over-conflict-inv* **where**
‹*iterate-over-conflict-inv M D₀'* = (λ(*D*, *D'*). *D* ⊆# *D₀'* ∧ *D'* ⊆# *D*)›

definition *is-literal-redundant-spec* **where**
‹*is-literal-redundant-spec K NU UNE D L* = *SPEC*(λ*b*. *b* ⟶
NU + *UNE* ⊨*pm remove1-mset L* (*add-mset K D*))›

definition *iterate-over-conflict*
:: ‹'*v literal* ⇒ ('*v*, '*mark*) *ann-lits* ⇒ '*v clauses* ⇒ '*v clauses* ⇒ '*v clause* ⇒
'*v clause nres*›

where
  ‹*iterate-over-conflict K M NU UNE* D_0' = *do* {
    (*D*, -) ←
      WHILE$_T$*iterate-over-conflict-inv M* D_0'
      (λ(*D*, *D'*). *D'* ≠ {#})
      (λ(*D*, *D'*). *do*{
        x ← *SPEC* (λ*x*. *x* ∈# *D'*);
        red ← *is-literal-redundant-spec K NU UNE D x*;
        if ¬*red*
        then RETURN (*D*, *remove1-mset x D'*)
        else RETURN (*remove1-mset x D*, *remove1-mset x D'*)
       })
      (D_0', D_0');
    RETURN D
}›

definition *minimize-and-extract-highest-lookup-conflict-inv* **where**
  ‹*minimize-and-extract-highest-lookup-conflict-inv* = (λ(*D*, *i*, *s*, *outl*).
    length outl ≤ *unat32-max* ∧ *mset* (*tl outl*) = *D* ∧ *outl* ≠ [] ∧ *i* ≥ *1*)›

type-synonym *'v conflict-highest-conflict* = ‹(*'v literal* × *nat*) *option*›

definition (**in** −) *atm-in-conflict* **where**
  ‹*atm-in-conflict L D* ⟷ *L* ∈ *atms-of D*›

definition *atm-in-conflict-lookup* :: ‹*nat* ⇒ *lookup-clause-rel* ⇒ *bool*› **where**
  ‹*atm-in-conflict-lookup* = (λ*L* (-, *xs*). *xs* ! *L* ≠ *None*)›

definition *atm-in-conflict-lookup-pre* :: ‹*nat* ⇒ *lookup-clause-rel* ⇒ *bool*› **where**
‹*atm-in-conflict-lookup-pre L xs* ⟷ *L* < *length* (*snd xs*)›

lemma *atm-in-conflict-lookup-atm-in-conflict*:
  ‹(*uncurry* (*RETURN oo atm-in-conflict-lookup*), *uncurry* (*RETURN oo atm-in-conflict*)) ∈
    [λ(*L*, *xs*). *L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)]$_f$ *Id* ×$_f$ *lookup-clause-rel* $\mathcal{A}$ → ⟨*bool-rel*⟩*nres-rel*›
  apply (*intro frefI nres-relI*)
  subgoal for *x y*
    using *mset-as-position-in-iff-nth*[*of* ‹*snd* (*snd x*)› ‹*snd y*› ‹*Pos* (*fst x*)›]
      mset-as-position-in-iff-nth[*of* ‹*snd* (*snd x*)› ‹*snd y*› ‹*Neg* (*fst x*)›]
    by (*cases x*; *cases y*)
      (*auto simp*: *atm-in-conflict-lookup-def atm-in-conflict-def*
        lookup-clause-rel-def atm-iff-pos-or-neg-lit
        pos-lit-in-atms-of neg-lit-in-atms-of)
  done

lemma *atm-in-conflict-lookup-pre*:
  fixes *x1* :: ‹*nat*› **and** *x2* :: ‹*nat*›
  assumes
    ‹*x1n* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
    ‹(*x2f*, *x2a*) ∈ *lookup-clause-rel* $\mathcal{A}$›
  shows ‹*atm-in-conflict-lookup-pre* (*atm-of x1n*) *x2f*›
proof −
  show *?thesis*
    using *assms*
    by (*auto simp*: *lookup-clause-rel-def atm-in-conflict-lookup-pre-def atms-of-def*)

qed

definition *is-literal-redundant-lookup-spec* **where**
  ‹*is-literal-redundant-lookup-spec* $\mathcal{A}$ *M NU NUE D' L s* =
   SPEC(λ(*s'*, *b*). *b* ⟶ (∀ *D*. (*D'*, *D*) ∈ *lookup-clause-rel* $\mathcal{A}$ ⟶
     (*mset* ‘# *mset* (*tl NU*)) + *NUE* ⊨*pm remove1-mset L D*))›

type-synonym (**in** −) *conflict-min-cach-l* = ‹*minimize-status list* × *nat list*›

definition (**in** −) *conflict-min-cach-set-removable-l*
 :: ‹*conflict-min-cach-l* ⇒ *nat* ⇒ *conflict-min-cach-l nres*›
where
 ‹*conflict-min-cach-set-removable-l* = (λ(*cach*, *sup*) *L*. *do* {
    ASSERT(*L* < *length cach*);
    ASSERT(*length sup* ≤ 1 + *unat32-max div* 2);
    RETURN (*cach*[*L* := *SEEN-REMOVABLE*], *if cach* ! *L* = *SEEN-UNKNOWN then sup* @ [*L*] *else sup*)
  })›

definition (**in** −) *conflict-min-cach* :: ‹*nat conflict-min-cach* ⇒ *nat* ⇒ *minimize-status*› **where**
 [*simp*]: ‹*conflict-min-cach cach L* = *cach L*›

definition *lit-redundant-reason-stack2*
 :: ‹'*v literal* ⇒ '*v clauses-l* ⇒ *nat* ⇒ (*nat* × *nat* × *bool*)› **where**
‹*lit-redundant-reason-stack2 L NU C'* =
 (*if length* (*NU* ∝ *C'*) > 2 *then* (*C'*, 1, *False*)
 else if NU ∝ *C'* ! 0 = *L then* (*C'*, 1, *False*)
 else (*C'*, 0, *True*))›

definition *ana-lookup-rel*
 :: ‹*nat clauses-l* ⇒ ((*nat* × *nat* × *bool*) × (*nat* × *nat* × *nat* × *nat*)) *set*›
where
‹*ana-lookup-rel NU* = {((*C*, *i*, *b*), (*C'*, *k'*, *i'*, *len'*)).
 C = *C'* ∧ *k'* = (*if b then* 1 *else* 0) ∧ *i* = *i'* ∧
 len' = (*if b then* 1 *else length* (*NU* ∝ *C*))}›

lemma *ana-lookup-rel-alt-def*:
 ‹((*C*, *i*, *b*), (*C'*, *k'*, *i'*, *len'*)) ∈ *ana-lookup-rel NU* ⟷
 C = *C'* ∧ *k'* = (*if b then* 1 *else* 0) ∧ *i* = *i'* ∧
 len' = (*if b then* 1 *else length* (*NU* ∝ *C*))›
 unfolding *ana-lookup-rel-def*
 by *auto*

abbreviation *ana-lookups-rel* **where**
 ‹*ana-lookups-rel NU* ≡ ⟨*ana-lookup-rel NU*⟩*list-rel*›

definition *ana-lookup-conv* :: ‹*nat clauses-l* ⇒ (*nat* × *nat* × *bool*) ⇒ (*nat* × *nat* × *nat* × *nat*)› **where**
‹*ana-lookup-conv NU* = (λ(*C*, *i*, *b*). (*C*, (*if b then* 1 *else* 0), *i*, (*if b then* 1 *else length* (*NU* ∝ *C*))))›

definition *get-literal-and-remove-of-analyse-wl2*
  :: ‹'*v clause-l* ⇒ (*nat* × *nat* × *bool*) *list* ⇒ '*v literal* × (*nat* × *nat* × *bool*) *list*› **where**
 ‹*get-literal-and-remove-of-analyse-wl2 C analyse* =
   (*let* (*i*, *j*, *b*) = *last analyse in*
    (*C* ! *j*, *analyse*[*length analyse* − 1 := (*i*, *j* + 1, *b*)]))›

definition *lit-redundant-rec-wl-inv2* **where**
‹*lit-redundant-rec-wl-inv2 M NU D* =
(λ(*cach*, *analyse*, *b*). ∃ *analyse'*. (*analyse*, *analyse'*) ∈ *ana-lookups-rel NU* ∧
lit-redundant-rec-wl-inv M NU D (*cach*, *analyse'*, *b*))›

definition *mark-failed-lits-stack-inv2* **where**
‹*mark-failed-lits-stack-inv2 NU analyse* = (λ*cach*.
∃ *analyse'*. (*analyse*, *analyse'*) ∈ *ana-lookups-rel NU* ∧
mark-failed-lits-stack-inv NU analyse' cach)›

definition *lit-redundant-rec-wl-lookup*
:: ‹*nat multiset* ⇒ (*nat*,*nat*)*ann-lits* ⇒ *nat clauses-l* ⇒ *nat clause* ⇒
- ⇒ - ⇒ - ⇒ (- × - × *bool*) *nres*›
where
‹*lit-redundant-rec-wl-lookup* 𝒜 *M NU D cach analysis lbd* =
WHILE$_T^{\textit{lit-redundant-rec-wl-inv2 M NU D}}$
(λ(*cach*, *analyse*, *b*). *analyse* ≠ [])
(λ(*cach*, *analyse*, *b*). *do* {
ASSERT(*analyse* ≠ []);
ASSERT(*length analyse* ≤ *length M*);
let (*C*,*k*, *i*, *len*) = *ana-lookup-conv NU* (*last analyse*);
ASSERT(*C* ∈# *dom-m NU*);
ASSERT(*length* (*NU* ∝ *C*) > *k*); — >= 2 would work too
ASSERT (*NU* ∝ *C* ! *k* ∈ *lits-of-l M*);
ASSERT(*NU* ∝ *C* ! *k* ∈# $\mathcal{L}_{all}$ 𝒜);
ASSERT(*literals-are-in-*$\mathcal{L}_{in}$ 𝒜 (*mset* (*NU* ∝ *C*)));
ASSERT(*length* (*NU* ∝ *C*) ≤ *Suc* (*unat32-max div 2*));
ASSERT(*len* ≤ *length* (*NU* ∝ *C*)); — makes the refinement easier
let C = *NU* ∝ *C*;
if i ≥ *len*
then
RETURN(*cach* (*atm-of* (*C* ! *k*) := *SEEN-REMOVABLE*), *butlast analyse*, *True*)
else do {
let (*L*, *analyse*) = *get-literal-and-remove-of-analyse-wl2 C analyse*;
ASSERT(*L* ∈# $\mathcal{L}_{all}$ 𝒜);
let b = ¬*level-in-lbd* (*get-level M L*) *lbd*;
if (*get-level M L* = *0* ∨
conflict-min-cach cach (*atm-of L*) = *SEEN-REMOVABLE* ∨
atm-in-conflict (*atm-of L*) *D*)
then RETURN (*cach*, *analyse*, *False*)
else if b ∨ *conflict-min-cach cach* (*atm-of L*) = *SEEN-FAILED*
then do {
ASSERT(*mark-failed-lits-stack-inv2 NU analyse cach*);
cach ← *mark-failed-lits-wl NU analyse cach*;
RETURN (*cach*, [], *False*)
}
else do {
ASSERT(− *L* ∈ *lits-of-l M*);
C ← *get-propagation-reason M* (−*L*);
case C of
Some C ⇒ *do* {
ASSERT(*C* ∈# *dom-m NU*);
ASSERT(*length* (*NU* ∝ *C*) ≥ *2*);
ASSERT(*literals-are-in-*$\mathcal{L}_{in}$ 𝒜 (*mset* (*NU* ∝ *C*)));
ASSERT(*length* (*NU* ∝ *C*) ≤ *Suc* (*unat32-max div 2*));
RETURN (*cach*, *analyse* @ [*lit-redundant-reason-stack2* (−*L*) *NU C*], *False*)

```
    }
          | None ⇒ do {
              ASSERT(mark-failed-lits-stack-inv2 NU analyse cach);
              cach ← mark-failed-lits-wl NU analyse cach;
              RETURN (cach, [], False)
            }
        }
      }
    })
   (cach, analysis, False)⟩

lemma lit-redundant-rec-wl-ref-butlast:
  ⟨lit-redundant-rec-wl-ref NU x ⟹ lit-redundant-rec-wl-ref NU (butlast x)⟩
  by (cases x rule: rev-cases)
    (auto simp: lit-redundant-rec-wl-ref-def dest: in-set-butlastD)

lemma lit-redundant-rec-wl-lookup-mark-failed-lits-stack-inv:
  assumes
    ⟨(x, x') ∈ Id⟩ and
    ⟨case x of (cach, analyse, b) ⇒ analyse ≠ []⟩ and
    ⟨lit-redundant-rec-wl-inv M NU D x'⟩ and
    ⟨¬ snd (snd (snd (last x1a))) ≤ fst (snd (snd (last x1a)))⟩ and
    ⟨get-literal-and-remove-of-analyse-wl (NU ∝ fst (last x1c)) x1c = (x1e, x2e)⟩ and
    ⟨x2 = (x1a, x2a)⟩ and
    ⟨x' = (x1, x2)⟩ and
    ⟨x2b = (x1c, x2c)⟩ and
    ⟨x = (x1b, x2b)⟩
  shows ⟨mark-failed-lits-stack-inv NU x2e x1b⟩
proof −
  show ?thesis
    using assms
    unfolding mark-failed-lits-stack-inv-def lit-redundant-rec-wl-inv-def
      lit-redundant-rec-wl-ref-def get-literal-and-remove-of-analyse-wl-def
    by (cases ⟨x1a⟩ rule: rev-cases)
      (auto simp: elim!: in-set-upd-cases)
qed

context
  fixes M D 𝒜 NU analysis analysis'
  assumes
    M-D: ⟨M ⊨as CNot D⟩ and
    n-d: ⟨no-dup M⟩ and
    lits: ⟨literals-are-in-ℒ_in-trail 𝒜 M⟩ and
    ana: ⟨(analysis, analysis') ∈ ana-lookups-rel NU⟩ and
    lits-NU: ⟨literals-are-in-ℒ_in-mm 𝒜 ((mset ∘ fst) `# ran-m NU)⟩ and
    bounded: ⟨isasat-input-bounded 𝒜⟩
begin
lemma ccmin-rel:
  assumes ⟨lit-redundant-rec-wl-inv M NU D (cach, analysis', False)⟩
  shows ⟨((cach, analysis, False), cach, analysis', False)
        ∈ {((cach, ana, b), cach', ana', b').
          (ana, ana') ∈ ana-lookups-rel NU ∧
          b = b' ∧ cach = cach' ∧ lit-redundant-rec-wl-inv M NU D (cach, ana', b)}⟩
proof −
  show ?thesis using ana assms by auto
qed
```

context
  fixes x :: ‹$(nat \Rightarrow minimize\text{-}status) \times (nat \times nat \times bool)\ list \times bool$› **and**
  x' :: ‹$(nat \Rightarrow minimize\text{-}status) \times (nat \times nat \times nat \times nat)\ list \times bool$›
  assumes x-x': ‹$(x, x') \in \{((cach, ana, b), (cach', ana', b')).$
    $(ana, ana') \in ana\text{-}lookups\text{-}rel\ NU \wedge b = b' \wedge cach = cach' \wedge$
    $lit\text{-}redundant\text{-}rec\text{-}wl\text{-}inv\ M\ NU\ D\ (cach, ana', b)\}$›
begin

lemma *ccmin-lit-redundant-rec-wl-inv2*:
  assumes ‹$lit\text{-}redundant\text{-}rec\text{-}wl\text{-}inv\ M\ NU\ D\ x'$›
  shows ‹$lit\text{-}redundant\text{-}rec\text{-}wl\text{-}inv2\ M\ NU\ D\ x$›
  using x-x' **unfolding** *lit-redundant-rec-wl-inv2-def*
  by *auto*

context
  assumes
    ‹$lit\text{-}redundant\text{-}rec\text{-}wl\text{-}inv2\ M\ NU\ D\ x$› **and**
    ‹$lit\text{-}redundant\text{-}rec\text{-}wl\text{-}inv\ M\ NU\ D\ x'$›
begin

lemma *ccmin-cond*:
  fixes $x1$ :: ‹$nat \Rightarrow minimize\text{-}status$› **and**
    $x2$ :: ‹$(nat \times nat \times bool)\ list \times bool$› **and**
    $x1a$ :: ‹$(nat \times\ nat \times bool)\ list$› **and**
    $x2a$ :: ‹$bool$› **and** $x1b$ :: ‹$nat \Rightarrow minimize\text{-}status$› **and**
    $x2b$ :: ‹$(nat \times nat \times nat \times nat)\ list \times bool$› **and**
    $x1c$ :: ‹$(nat \times nat \times nat \times nat)\ list$› **and** $x2c$ :: ‹$bool$›
  assumes
    ‹$x2 = (x1a, x2a)$›
    ‹$x = (x1, x2)$›
    ‹$x2b = (x1c, x2c)$›
    ‹$x' = (x1b, x2b)$›
  shows ‹$(x1a \neq []) = (x1c \neq [])$›
  using *assms* x-x'
  by *auto*

end

context
  assumes
    ‹$case\ x\ of\ (cach, analyse, b) \Rightarrow analyse \neq []$› **and**
    ‹$case\ x'\ of\ (cach, analyse, b) \Rightarrow analyse \neq []$› **and**
    inv2: ‹$lit\text{-}redundant\text{-}rec\text{-}wl\text{-}inv2\ M\ NU\ D\ x$› **and**
    ‹$lit\text{-}redundant\text{-}rec\text{-}wl\text{-}inv\ M\ NU\ D\ x'$›
begin

context
  fixes $x1$ :: ‹$nat \Rightarrow minimize\text{-}status$› **and**
  $x2$ :: ‹$(nat \times nat \times nat \times nat)\ list \times bool$› **and**
  $x1a$ :: ‹$(nat \times nat \times nat \times nat)\ list$› **and** $x2a$ :: ‹$bool$› **and**
  $x1b$ :: ‹$nat \Rightarrow minimize\text{-}status$› **and**
  $x2b$ :: ‹$(nat \times nat \times bool)\ list \times bool$› **and**
  $x1c$ :: ‹$(nat \times nat \times bool)\ list$› **and**

x2c :: ‹*bool*›
assumes *st*:
‹*x2* = (*x1a*, *x2a*)›
‹*x'* = (*x1*, *x2*)›
‹*x2b* = (*x1c*, *x2c*)›
‹*x* = (*x1b*, *x2b*)› **and**
x1a: ‹*x1a* ≠ []›
begin

private lemma *st*:
‹*x2* = (*x1a*, *x2a*)›
‹*x'* = (*x1*, *x1a*, *x2a*)›
‹*x2b* = (*x1c*, *x2a*)›
‹*x* = (*x1*, *x1c*, *x2a*)›
‹*x1b* = *x1*›
‹*x2c* = *x2a*› **and**
x1c: ‹*x1c* ≠ []›
using *st x-x' x1a* **by** *auto*

lemma *ccmin-nempty*:
shows ‹*x1c* ≠ []›
using *x-x' x1a*
by (*auto simp*: *st*)

context
notes -[*simp*] = *st*
fixes *x1d* :: ‹*nat*› **and** *x2d* :: ‹*nat* × *nat* × *nat*› **and**
x1e :: ‹*nat*› **and** *x2e* :: ‹*nat* × *nat*› **and**
x1f :: ‹*nat*› **and**
x2f :: ‹*nat*› **and** *x1g* :: ‹*nat*› **and**
x2g :: ‹*nat* × *nat* × *nat*› **and**
x1h :: ‹*nat*› **and**
x2h :: ‹*nat* × *nat*› **and**
x1i :: ‹*nat*› **and**
x2i :: ‹*nat*›
assumes
ana-lookup-conv: ‹*ana-lookup-conv NU* (*last x1c*) = (*x1g*, *x2g*)› **and**
last: ‹*last x1a* = (*x1d*, *x2d*)› **and**
dom: ‹*x1d* ∈# *dom-m NU*› **and**
le: ‹*x1e* < *length* (*NU* ∝ *x1d*)› **and**
in-lits: ‹*NU* ∝ *x1d* ! *x1e* ∈ *lits-of-l M*› **and**
st2:
‹*x2g* = (*x1h*, *x2h*)›
‹*x2e* = (*x1f*, *x2f*)›
‹*x2d* = (*x1e*, *x2e*)›
‹*x2h* = (*x1i*, *x2i*)›
begin

private lemma *x1g-x1d*:
‹*x1g* = *x1d*›
‹*x1h* = *x1e*›
‹*x1i* = *x1f*›
using *st2 last ana-lookup-conv x-x' x1a last*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *ana-lookup-conv-def ana-lookup-rel-def*
list-rel-append-single-iff; *fail*)+

private definition *j* **where**
‹*j* = *fst* (*snd* (*last x1c*))›

private definition *b* **where**
‹*b* = *snd* (*snd* (*last x1c*))›

private lemma *last-x1c*[*simp*]:
‹*last x1c* = (*x1d*, *x1f*, *b*)›
using *inv2 x1a last x-x'* **unfolding** *x1g-x1d st j-def b-def st2*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def)

private lemma
ana: ‹(*x1d*, (*if b then 1 else 0*), *x1f*, (*if b then 1 else length* (*NU* ∝ *x1d*))) = (*x1d*, *x1e*, *x1f*, *x2i*)› **and**
st3:
‹*x1e* = (*if b then 1 else 0*)›
‹*x1f* = *j*›
‹*x2f* = (*if b then 1 else length* (*NU* ∝ *x1d*))›
‹*x2d* = (*if b then 1 else 0*, *j*, *if b then 1 else length* (*NU* ∝ *x1d*))› **and**
‹*j* ≤ (*if b then 1 else length* (*NU* ∝ *x1d*))› **and**
‹*x1d* ∈# *dom-m NU*› **and**
‹0 < *x1d*› **and**
‹(*if b then 1 else length* (*NU* ∝ *x1d*)) ≤ *length* (*NU* ∝ *x1d*)› **and**
‹(*if b then 1 else 0*) < *length* (*NU* ∝ *x1d*)› **and**
dist: ‹*distinct* (*NU* ∝ *x1d*)› **and**
tauto: ‹¬ *tautology* (*mset* (*NU* ∝ *x1d*))›
subgoal
using *inv2 x1a last x-x' x1c ana-lookup-conv*
unfolding *x1g-x1d st j-def b-def st2*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def ana-lookup-conv-def
simp del: *x1c*)
subgoal
using *inv2 x1a last x-x' x1c* **unfolding** *x1g-x1d st j-def b-def st2*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)
subgoal
using *inv2 x1a last x-x' x1c* **unfolding** *x1g-x1d st j-def b-def st2*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def
lit-redundant-rec-wl-ref-def
simp del: *x1c*)
subgoal
using *inv2 x1a last x-x' x1c* **unfolding** *x1g-x1d st j-def b-def st2*
by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*;
auto simp: *lit-redundant-rec-wl-inv2-def list-rel-append-single-iff*
lit-redundant-rec-wl-inv-def ana-lookup-rel-def

```
      lit-redundant-rec-wl-ref-def
    simp del: x1c)
subgoal
  using inv2 x1a last x-x' x1c unfolding x1g-x1d st j-def b-def st2
  by (cases x1a rule: rev-cases; cases x1c rule: rev-cases;
   auto simp: lit-redundant-rec-wl-inv2-def list-rel-append-single-iff
      lit-redundant-rec-wl-inv-def ana-lookup-rel-def
      lit-redundant-rec-wl-ref-def
    simp del: x1c)
subgoal
  using inv2 x1a last x-x' x1c unfolding x1g-x1d st j-def b-def st2
  by (cases x1a rule: rev-cases; cases x1c rule: rev-cases;
   auto simp: lit-redundant-rec-wl-inv2-def list-rel-append-single-iff
      lit-redundant-rec-wl-inv-def ana-lookup-rel-def
      lit-redundant-rec-wl-ref-def
    simp del: x1c)
subgoal
  using inv2 x1a last x-x' x1c unfolding x1g-x1d st j-def b-def
  by (cases x1a rule: rev-cases; cases x1c rule: rev-cases;
   auto simp: lit-redundant-rec-wl-inv2-def list-rel-append-single-iff
      lit-redundant-rec-wl-inv-def ana-lookup-rel-def
      lit-redundant-rec-wl-ref-def
    simp del: x1c)
subgoal
  using inv2 x1a last x-x' x1c unfolding x1g-x1d st j-def b-def
  by (cases x1a rule: rev-cases; cases x1c rule: rev-cases;
   auto simp: lit-redundant-rec-wl-inv2-def list-rel-append-single-iff
      lit-redundant-rec-wl-inv-def ana-lookup-rel-def
      lit-redundant-rec-wl-ref-def
    simp del: x1c)
subgoal
  using inv2 x1a last x-x' x1c unfolding x1g-x1d st j-def b-def
  by (cases x1a rule: rev-cases; cases x1c rule: rev-cases;
   auto simp: lit-redundant-rec-wl-inv2-def list-rel-append-single-iff
      lit-redundant-rec-wl-inv-def ana-lookup-rel-def
      lit-redundant-rec-wl-ref-def
    simp del: x1c)
subgoal
  using inv2 x1a last x-x' x1c unfolding x1g-x1d st j-def b-def
  by (cases x1a rule: rev-cases; cases x1c rule: rev-cases;
   auto simp: lit-redundant-rec-wl-inv2-def list-rel-append-single-iff
      lit-redundant-rec-wl-inv-def ana-lookup-rel-def
      lit-redundant-rec-wl-ref-def
    simp del: x1c)
subgoal
  using inv2 x1a last x-x' x1c unfolding x1g-x1d st j-def b-def
  by (cases x1a rule: rev-cases; cases x1c rule: rev-cases;
   auto simp: lit-redundant-rec-wl-inv2-def list-rel-append-single-iff
      lit-redundant-rec-wl-inv-def ana-lookup-rel-def
      lit-redundant-rec-wl-ref-def
    simp del: x1c)
subgoal
  using inv2 x1a last x-x' x1c unfolding x1g-x1d st j-def b-def
  by (cases x1a rule: rev-cases; cases x1c rule: rev-cases;
   auto simp: lit-redundant-rec-wl-inv2-def list-rel-append-single-iff
      lit-redundant-rec-wl-inv-def ana-lookup-rel-def
```

lit-redundant-rec-wl-ref-def
simp del: *x1c*)
done

lemma *ccmin-in-dom*:
shows *x1g-dom*: ‹*x1g* ∈# *dom-m NU*›
using *dom* **unfolding** *x1g-x1d* **.**

lemma *ccmin-in-dom-le-length*:
shows ‹*x1h* < *length* (*NU* ∝ *x1g*)›
using *le* **unfolding** *x1g-x1d* **.**

lemma *ccmin-in-trail*:
shows ‹*NU* ∝ *x1g* ! *x1h* ∈ *lits-of-l M*›
using *in-lits* **unfolding** *x1g-x1d* **.**

lemma *ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-x1g*:
shows ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*NU* ∝ *x1g*))›
using *lits-NU multi-member-split*[*OF x1g-dom*]
by (*auto simp*: *ran-m-def literals-are-in-*$\mathcal{L}_{in}$*-mm-add-mset*)

lemma *ccmin-le-unat32-max*:
‹*length* (*NU* ∝ *x1g*) ≤ *Suc* (*unat32-max div 2*)›
using *simple-clss-size-upper-div2*[*OF bounded ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-x1g*]
dist tauto **unfolding** *x1g-x1d*
by *auto*

lemma *ccmin-in-all-lits*:
shows ‹*NU* ∝ *x1g* ! *x1h* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
using *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$[*OF ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-x1g*, *of x1h*]
le **unfolding** *x1g-x1d* **by** *auto*

lemma *ccmin-less-length*:
shows ‹*x2i* ≤ *length* (*NU* ∝ *x1g*)›
using *le ana* **unfolding** *x1g-x1d st3* **by** (*simp split*: *if-splits*)

lemma *ccmin-same-cond*:
shows ‹(*x2i* ≤ *x1i*) = (*x2f* ≤ *x1f*)›
using *le ana* **unfolding** *x1g-x1d st3* **by** (*simp split*: *if-splits*)

lemma *ccmin-set-removable*:
assumes
‹*x2i* ≤ *x1i*› **and**
‹*x2f* ≤ *x1f*› **and** ‹*lit-redundant-rec-wl-inv2 M NU D x*›
shows ‹((*x1b*(*atm-of* (*NU* ∝ *x1g* ! *x1h*) := *SEEN-REMOVABLE*), *butlast x1c*, *True*),
x1(*atm-of* (*NU* ∝ *x1d* ! *x1e*) := *SEEN-REMOVABLE*), *butlast x1a*, *True*)
∈ {((*cach*, *ana*, *b*), *cach'*, *ana'*, *b'*).
(*ana*, *ana'*) ∈ *ana-lookups-rel NU* ∧
b = *b'* ∧ *cach* = *cach'* ∧ *lit-redundant-rec-wl-inv M NU D* (*cach*, *ana'*, *b*)}›
using *x-x'* **by** (*auto simp*: *x1g-x1d lit-redundant-rec-wl-ref-butlast lit-redundant-rec-wl-inv-def*
dest: *list-rel-butlast*)

context
assumes
le: ‹¬ *x2i* ≤ *x1i*› ‹¬ *x2f* ≤ *x1f*›
begin

context
notes -[*simp*]= *x1g-x1d st2 last*
fixes *x1j* :: ‹*nat literal*› **and** *x2j* :: ‹(*nat* × *nat* × *nat* × *nat*) *list*› **and**
x1k :: ‹*nat literal*› **and** *x2k* :: ‹(*nat* × *nat* × *bool*) *list*›
assumes
rem: ‹*get-literal-and-remove-of-analyse-wl* (*NU* ∝ *x1d*) *x1a* = (*x1j*, *x2j*)› **and**
rem2:‹*get-literal-and-remove-of-analyse-wl2* (*NU* ∝ *x1g*) *x1c* = (*x1k*, *x2k*)› **and**
‹*fst* (*snd* (*snd* (*last x2j*))) ≠ *0*› **and**
ux1j-M: ‹− *x1j* ∈ *lits-of-l M*›
begin

private lemma *confl-min-last*: ‹(*last x1c*, *last x1a*) ∈ *ana-lookup-rel NU*›
using *x1a x1c x-x′ rem rem2 last ana-lookup-conv* **unfolding** *x1g-x1d st2 b-def st*
by (*cases x1c rule*: *rev-cases*; *cases x1a rule*: *rev-cases*)
(*auto simp*: *list-rel-append-single-iff*
get-literal-and-remove-of-analyse-wl-def
get-literal-and-remove-of-analyse-wl2-def)

private lemma *rel*: ‹(*x1c*[*length x1c* − *Suc 0* := (*x1d*, *Suc x1f*, *b*)], *x1a*
[*length x1a* − *Suc 0* := (*x1d*, *x1e*, *Suc x1f*, *x2f*)])
∈ *ana-lookups-rel NU*›
using *x1a x1c x-x′ rem rem2 confl-min-last* **unfolding** *x1g-x1d st2 last b-def st*
by (*cases x1c rule*: *rev-cases*; *cases x1a rule*: *rev-cases*)
(*auto simp*: *list-rel-append-single-iff*
ana-lookup-rel-alt-def get-literal-and-remove-of-analyse-wl-def
get-literal-and-remove-of-analyse-wl2-def)

private lemma *x1k-x1j*: ‹*x1k* = *x1j*› ‹*x1j* = *NU* ∝ *x1d* ! *x1f*› **and**
x2k-x2j: ‹(*x2k*, *x2j*) ∈ *ana-lookups-rel NU*›
subgoal
using *x1a x1c x-x′ rem rem2 confl-min-last* **unfolding** *x1g-x1d st2 last b-def st*
by (*cases x1c rule*: *rev-cases*; *cases x1a rule*: *rev-cases*)
(*auto simp*: *list-rel-append-single-iff*
ana-lookup-rel-alt-def get-literal-and-remove-of-analyse-wl-def
get-literal-and-remove-of-analyse-wl2-def)
subgoal
using *x1a x1c x-x′ rem rem2 confl-min-last* **unfolding** *x1g-x1d st2 last b-def st*
by (*cases x1c rule*: *rev-cases*; *cases x1a rule*: *rev-cases*)
(*auto simp*: *list-rel-append-single-iff*
ana-lookup-rel-alt-def get-literal-and-remove-of-analyse-wl-def
get-literal-and-remove-of-analyse-wl2-def)
subgoal
using *x1a x1c x-x′ rem rem2 confl-min-last* **unfolding** *x1g-x1d st2 last b-def st*
by (*cases x1c rule*: *rev-cases*; *cases x1a rule*: *rev-cases*)
(*auto simp*: *list-rel-append-single-iff*
ana-lookup-rel-alt-def get-literal-and-remove-of-analyse-wl-def
get-literal-and-remove-of-analyse-wl2-def)
done

lemma *ccmin-x1k-all*:
shows ‹*x1k* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
unfolding *x1k-x1j*
using *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$[*OF ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-x1g*, *of x1f*]
literals-are-in-$\mathcal{L}_{in}$*-trail-in-lits-of-l*[*OF lits* ‹− *x1j* ∈ *lits-of-l M*›]
le st3 **unfolding** *x1g-x1d* **by** (*auto split*: *if-splits simp*: *x1k-x1j uminus-*$\mathcal{A}_{in}$*-iff*)

context
  notes -[*simp*]= *x1k-x1j*
  fixes *b* :: ‹*bool*› **and** *lbd*
  assumes *b*: ‹(¬ *level-in-lbd* (*get-level M x1k*) *lbd*, *b*) ∈ *bool-rel*›
begin

private lemma *in-conflict-atm-in*:
  ‹− *x1e′* ∈ *lits-of-l M* ⟹ *atm-in-conflict* (*atm-of x1e′*) *D* ⟷ *x1e′* ∈# *D*› **for** *x1e′*
  using *M-D n-d*
  by (*auto simp*: *atm-in-conflict-def true-annots-true-cls-def-iff-negation-in-model*
      atms-of-def atm-of-eq-atm-of dest!: *multi-member-split no-dup-consistentD*)

lemma *ccmin-already-seen*:
  shows ‹(*get-level M x1k* = *0* ∨
        conflict-min-cach x1b (*atm-of x1k*) = *SEEN-REMOVABLE* ∨
        atm-in-conflict (*atm-of x1k*) *D*) =
       (*get-level M x1j* = *0* ∨ *x1* (*atm-of x1j*) = *SEEN-REMOVABLE* ∨ *x1j* ∈# *D*)›
  using *in-lits ana ux1j-M*
  by (*auto simp add*: *in-conflict-atm-in*)

private lemma *ccmin-lit-redundant-rec-wl-inv*: ‹*lit-redundant-rec-wl-inv M NU D*
    (*x1*, *x2j*, *False*)›
  using *x-x′ last ana-lookup-conv rem rem2 x1a x1c le*
  by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*)
    (*auto simp add*: *lit-redundant-rec-wl-inv-def lit-redundant-rec-wl-ref-def*
    lit-redundant-reason-stack-def get-literal-and-remove-of-analyse-wl-def
    list-rel-append-single-iff get-literal-and-remove-of-analyse-wl2-def)

lemma *ccmin-already-seen-rel*:
  assumes
    ‹*get-level M x1k* = *0* ∨
     conflict-min-cach x1b (*atm-of x1k*) = *SEEN-REMOVABLE* ∨
     atm-in-conflict (*atm-of x1k*) *D*› **and**
    ‹*get-level M x1j* = *0* ∨ *x1* (*atm-of x1j*) = *SEEN-REMOVABLE* ∨ *x1j* ∈# *D*›
  shows ‹((*x1b*, *x2k*, *False*), *x1*, *x2j*, *False*)
       ∈ {((*cach*, *ana*, *b*), *cach′*, *ana′*, *b′*).
        (*ana*, *ana′*) ∈ *ana-lookups-rel NU* ∧
        b = *b′* ∧ *cach* = *cach′* ∧ *lit-redundant-rec-wl-inv M NU D* (*cach*, *ana′*, *b*)}›
  using *x2k-x2j ccmin-lit-redundant-rec-wl-inv* **by** *auto*

context
  assumes
    ‹¬ (*get-level M x1k* = *0* ∨
       conflict-min-cach x1b (*atm-of x1k*) = *SEEN-REMOVABLE* ∨
       atm-in-conflict (*atm-of x1k*) *D*)› **and**
    ‹¬ (*get-level M x1j* = *0* ∨ *x1* (*atm-of x1j*) = *SEEN-REMOVABLE* ∨ *x1j* ∈# *D*)›
begin
lemma *ccmin-already-failed*:
  shows ‹(¬ *level-in-lbd* (*get-level M x1k*) *lbd* ∨
        conflict-min-cach x1b (*atm-of x1k*) = *SEEN-FAILED*) =
       (*b* ∨ *x1* (*atm-of x1j*) = *SEEN-FAILED*)›
  using *b* **by** *auto*

context
  assumes
    ‹¬ *level-in-lbd* (*get-level M x1k*) *lbd* ∨
     conflict-min-cach x1b (*atm-of x1k*) = *SEEN-FAILED*› **and**
    ‹*b* ∨ *x1* (*atm-of x1j*) = *SEEN-FAILED*›
begin

lemma *ccmin-mark-failed-lits-stack-inv2-lbd*:
  shows ‹*mark-failed-lits-stack-inv2 NU x2k x1b*›
  using *x1a x1c x2k-x2j rem rem2 x-x′ le last*
  unfolding *mark-failed-lits-stack-inv-def lit-redundant-rec-wl-inv-def*
    lit-redundant-rec-wl-ref-def get-literal-and-remove-of-analyse-wl-def
  unfolding *mark-failed-lits-stack-inv2-def*
  apply −
  apply (*rule exI*[*of* - *x2j*])
  apply (*cases* ‹*x1a*› *rule*: *rev-cases*; *cases* ‹*x1c*› *rule*: *rev-cases*)
  by (*auto simp*: *mark-failed-lits-stack-inv-def elim*!: *in-set-upd-cases*)

lemma *ccmin-mark-failed-lits-wl-lbd*:
  shows ‹*mark-failed-lits-wl NU x2k x1b*
        ≤ ⇓ *Id*
          (*mark-failed-lits-wl NU x2j x1*)›
  by (*auto simp*: *mark-failed-lits-wl-def*)

lemma *ccmin-rel-lbd*:
  fixes *cach* :: ‹*nat* ⇒ *minimize-status*› **and** *cacha* :: ‹*nat* ⇒ *minimize-status*›
  assumes ‹(*cach*, *cacha*) ∈ *Id*›
  shows ‹((*cach*, [], *False*), *cacha*, [], *False*) ∈ {((*cach*, *ana*, *b*), *cach′*, *ana′*, *b′*).
      (*ana*, *ana′*) ∈ *ana-lookups-rel NU* ∧
      b = *b′* ∧ *cach* = *cach′* ∧ *lit-redundant-rec-wl-inv M NU D* (*cach*, *ana′*, *b*)}›
  using *x-x′ assms* **by** (*auto simp*: *lit-redundant-rec-wl-inv-def lit-redundant-rec-wl-ref-def*)

end

context
  assumes
    ‹¬ (¬ *level-in-lbd* (*get-level M x1k*) *lbd* ∨
        conflict-min-cach x1b (*atm-of x1k*) = *SEEN-FAILED*)› **and**
    ‹¬ (*b* ∨ *x1* (*atm-of x1j*) = *SEEN-FAILED*)›
begin

lemma *ccmin-lit-in-trail*:
  ‹− *x1k* ∈ *lits-of-l M*›
  using ‹− *x1j* ∈ *lits-of-l M*› *x1k-x1j*(*1*) **by** *blast*

lemma *ccmin-lit-eq*:
  ‹− *x1k* = − *x1j*›
  by *auto*

context
  fixes *xa* :: ‹*nat option*› **and** *x′a* :: ‹*nat option*›
  assumes *xa-x′a*: ‹(*xa*, *x′a*) ∈ ⟨*nat-rel*⟩*option-rel*›

begin

lemma *ccmin-lit-eq2*:
  ‹(*xa*, *x'a*) ∈ *Id*›
  using *xa-x'a* **by** *auto*

context
  assumes
    [*simp*]: ‹*xa* = *None*› ‹*x'a* = *None*›
begin

lemma *ccmin-mark-failed-lits-stack-inv2-dec*:
  ‹*mark-failed-lits-stack-inv2 NU x2k x1b*›
  using *x1a x1c x2k-x2j rem rem2 x-x' le last*
  unfolding *mark-failed-lits-stack-inv-def lit-redundant-rec-wl-inv-def*
    lit-redundant-rec-wl-ref-def get-literal-and-remove-of-analyse-wl-def
  unfolding *mark-failed-lits-stack-inv2-def*
  apply −
  apply (*rule exI*[*of* - *x2j*])
  apply (*cases* ‹*x1a*› *rule*: *rev-cases*; *cases* ‹*x1c*› *rule*: *rev-cases*)
  by (*auto simp*: *mark-failed-lits-stack-inv-def elim*!: *in-set-upd-cases*)

lemma *ccmin-mark-failed-lits-stack-wl-dec*:
  shows ‹*mark-failed-lits-wl NU x2k x1b*
      ≤ ⇓ *Id*
        (*mark-failed-lits-wl NU x2j x1*)›
  by (*auto simp*: *mark-failed-lits-wl-def*)

lemma *ccmin-rel-dec*:
  fixes *cach* :: ‹*nat* ⇒ *minimize-status*› **and** *cacha* :: ‹*nat* ⇒ *minimize-status*›
  assumes ‹(*cach*, *cacha*) ∈ *Id*›
  shows ‹((*cach*, [], *False*), *cacha*, [], *False*)
      ∈ {((*cach*, *ana*, *b*), *cach'*, *ana'*, *b'*).
     (*ana*, *ana'*) ∈ *ana-lookups-rel NU* ∧
     b = *b'* ∧ *cach* = *cach'* ∧ *lit-redundant-rec-wl-inv M NU D* (*cach*, *ana'*, *b*)}›
  using *assms* **by** (*auto simp*: *lit-redundant-rec-wl-ref-def lit-redundant-rec-wl-inv-def*)

end

context
  fixes *xb* :: ‹*nat*› **and** *x'b* :: ‹*nat*›
  assumes *H*:
    ‹*xa* = *Some xb*›
    ‹*x'a* = *Some x'b*›
    ‹(*xb*, *x'b*) ∈ *nat-rel*›
    ‹*x'b* ∈# *dom-m NU*›
    ‹2 ≤ *length* (*NU* ∝ *x'b*)›
    ‹*x'b* > 0›
    ‹*distinct* (*NU* ∝ *x'b*) ∧ ¬ *tautology* (*mset* (*NU* ∝ *x'b*))›
begin

lemma *ccmin-stack-pre*:
  shows ‹*xb* ∈# *dom-m NU*› ‹2 ≤ *length* (*NU* ∝ *xb*)›
  using *H* **by** *auto*

lemma *ccmin-literals-are-in-$\mathcal{L}_{in}$-NU-xb*:
  shows ‹*literals-are-in-$\mathcal{L}_{in}$* $\mathcal{A}$ (*mset* (*NU* $\propto$ *xb*))›
  using *lits-NU multi-member-split*[*of xb* ‹*dom-m NU*›] *H*
  by (*auto simp*: *ran-m-def literals-are-in-$\mathcal{L}_{in}$-mm-add-mset*)

lemma *ccmin-le-unat32-max-xb*:
  ‹*length* (*NU* $\propto$ *xb*) $\leq$ *Suc* (*unat32-max div 2*)›
  using *simple-clss-size-upper-div2*[*OF bounded ccmin-literals-are-in-$\mathcal{L}_{in}$-NU-xb*]
    H **unfolding** *x1g-x1d*
  by *auto*

private lemma *ccmin-lit-redundant-rec-wl-inv3*: ‹*lit-redundant-rec-wl-inv M NU D*
    (*x1*, *x2j* @ [*lit-redundant-reason-stack* (− *NU* $\propto$ *x1d* ! *x1f*) *NU x'b*], *False*)›
  using *ccmin-stack-pre H x-x' last ana-lookup-conv rem rem2 x1a x1c le*
  by (*cases x1a rule*: *rev-cases*; *cases x1c rule*: *rev-cases*)
    (*auto simp add*: *lit-redundant-rec-wl-inv-def lit-redundant-rec-wl-ref-def*
    lit-redundant-reason-stack-def get-literal-and-remove-of-analyse-wl-def
    list-rel-append-single-iff get-literal-and-remove-of-analyse-wl2-def)

lemma *ccmin-stack-rel*:
  shows ‹((*x1b*, *x2k* @ [*lit-redundant-reason-stack2* (− *x1k*) *NU xb*], *False*), *x1*,
        x2j @ [*lit-redundant-reason-stack* (− *x1j*) *NU x'b*], *False*)
       ∈ {((*cach*, *ana*, *b*), *cach'*, *ana'*, *b'*).
      (*ana*, *ana'*) ∈ *ana-lookups-rel NU* ∧
      b = *b'* ∧ *cach* = *cach'* ∧ *lit-redundant-rec-wl-inv M NU D* (*cach*, *ana'*, *b*)}›
  using *x2k-x2j H ccmin-lit-redundant-rec-wl-inv3*
  by (*auto simp*: *list-rel-append-single-iff ana-lookup-rel-alt-def*
    lit-redundant-reason-stack2-def lit-redundant-reason-stack-def)

end
end
end
end
end
end
end
end
end
end
end
end
end

lemma *lit-redundant-rec-wl-lookup-lit-redundant-rec-wl*:
  assumes
    M-D: ‹*M* ⊨*as CNot D*› **and**
    n-d: ‹*no-dup M*› **and**
    lits: ‹*literals-are-in-$\mathcal{L}_{in}$-trail* $\mathcal{A}$ *M*› **and**
    ‹(*analysis*, *analysis'*) ∈ *ana-lookups-rel NU*› **and**
    ‹*literals-are-in-$\mathcal{L}_{in}$-mm* $\mathcal{A}$ ((*mset* ∘ *fst*) ‘# *ran-m NU*)› **and**
    ‹*isasat-input-bounded* $\mathcal{A}$›
  shows
   ‹*lit-redundant-rec-wl-lookup* $\mathcal{A}$ *M NU D cach analysis lbd* ≤
     ⇓ (*Id* ×$_r$ (*ana-lookups-rel NU*) ×$_r$ *bool-rel*) (*lit-redundant-rec-wl M NU D cach analysis' lbd*)›
proof −

have *M*: ‹∀ *a* ∈ *lits-of-l M*. *a* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
using *literals-are-in-*$\mathcal{L}_{in}$*-trail-in-lits-of-l lits* **by** *blast*
have [*simp*]: ‹− *x1e* ∈ *lits-of-l M* ⟹ *atm-in-conflict* (*atm-of x1e*) *D* ⟷ *x1e* ∈# *D*› **for** *x1e*
using *M-D n-d*
by (*auto simp*: *atm-in-conflict-def true-annots-true-cls-def-iff-negation-in-model*
atms-of-def atm-of-eq-atm-of dest!: *multi-member-split no-dup-consistentD*)
have [*simp*, *intro*]: ‹− *x1e* ∈ *lits-of-l M* ⟹ *atm-of x1e* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
‹*x1e* ∈ *lits-of-l M* ⟹ *x1e* ∈# ($\mathcal{L}_{all}$ $\mathcal{A}$)›
‹− *x1e* ∈ *lits-of-l M* ⟹ *x1e* ∈# ($\mathcal{L}_{all}$ $\mathcal{A}$)› **for** *x1e*
using *lits atm-of-notin-atms-of-iff literals-are-in-*$\mathcal{L}_{in}$*-trail-in-lits-of-l* **apply** *blast*
using *M uminus-*$\mathcal{A}_{in}$*-iff* **by** *auto*
have [*refine-vcg*]: ‹(*a*, *b*) ∈ *Id* ⟹ (*a*, *b*) ∈ ⟨*Id*⟩*option-rel*› **for** *a b* **by** *auto*
have [*refine-vcg*]: ‹*get-propagation-reason M x*
≤ ⇓ (⟨*nat-rel*⟩*option-rel*) (*get-propagation-reason M y*)› **if** ‹*x* = *y*› **for** *x y*
by (*use that* **in** *auto*)
have [*refine-vcg*]:‹*RETURN* (¬ *level-in-lbd* (*get-level M L*) *lbd*) ≤ ⇓ *Id* (*RES UNIV*)› **for** *L*
by *auto*
have [*refine-vcg*]: ‹*mark-failed-lits-wl NU a b*
≤ ⇓ *Id*
(*mark-failed-lits-wl NU a′ b′*)› **if** ‹*a* = *a′*› **and** ‹*b* = *b′*› **for** *a a′ b b′*
unfolding *that* **by** *auto*

have *H*: ‹*lit-redundant-rec-wl-lookup* $\mathcal{A}$ *M NU D cach analysis lbd* ≤
⇓ {((*cach*, *ana*, *b*), *cach′*, *ana′*, *b′*).
(*ana*, *ana′*) ∈ *ana-lookups-rel NU* ∧
b = *b′* ∧ *cach* = *cach′* ∧ *lit-redundant-rec-wl-inv M NU D* (*cach*, *ana′*, *b*)}
(*lit-redundant-rec-wl M NU D cach analysis′ lbd*)›
using *assms* **apply** −
unfolding *lit-redundant-rec-wl-lookup-def lit-redundant-rec-wl-def WHILET-def*
apply (*refine-vcg*)
subgoal by (*rule ccmin-rel*)
subgoal by (*rule ccmin-lit-redundant-rec-wl-inv2*)
subgoal by (*rule ccmin-cond*)
subgoal by (*rule ccmin-nempty*)
subgoal by (*auto simp*: *list-rel-imp-same-length*)
subgoal by (*rule ccmin-in-dom*)
subgoal by (*rule ccmin-in-dom-le-length*)
subgoal by (*rule ccmin-in-trail*)
subgoal by (*rule ccmin-in-all-lits*)
subgoal by (*rule ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-x1g*)
subgoal by (*rule ccmin-le-unat32-max*)
subgoal by (*rule ccmin-less-length*)
subgoal by (*rule ccmin-same-cond*)
subgoal by (*rule ccmin-set-removable*)
subgoal by (*rule ccmin-x1k-all*)
subgoal by (*rule ccmin-already-seen*)
subgoal by (*rule ccmin-already-seen-rel*)
subgoal by (*rule ccmin-already-failed*)
subgoal by (*rule ccmin-mark-failed-lits-stack-inv2-lbd*)
apply (*rule ccmin-mark-failed-lits-wl-lbd*; *assumption*)
subgoal by (*rule ccmin-rel-lbd*)
subgoal by (*rule ccmin-lit-in-trail*)
subgoal by (*rule ccmin-lit-eq*)
subgoal by (*rule ccmin-lit-eq2*)
subgoal by (*rule ccmin-mark-failed-lits-stack-inv2-dec*)
apply (*rule ccmin-mark-failed-lits-stack-wl-dec*; *assumption*)

subgoal by (*rule ccmin-rel-dec*)
subgoal by (*rule ccmin-stack-pre*)
subgoal by (*rule ccmin-stack-pre*)
subgoal by (*rule ccmin-literals-are-in-$\mathcal{L}_{in}$-NU-xb*)
subgoal by (*rule ccmin-le-unat32-max-xb*)
subgoal by (*rule ccmin-stack-rel*)
done
show *?thesis*
by (*rule H*[*THEN order-trans*], *rule conc-fun-R-mono*)
auto
qed

definition *literal-redundant-wl-lookup* **where**
‹*literal-redundant-wl-lookup* $\mathcal{A}$ *M NU D cach L lbd* = *do* {
ASSERT(*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$);
if get-level M L = *0* ∨ *cach* (*atm-of L*) = *SEEN-REMOVABLE*
then RETURN (*cach*, [], *True*)
else if cach (*atm-of L*) = *SEEN-FAILED*
then RETURN (*cach*, [], *False*)
else do {
ASSERT(−*L* ∈ *lits-of-l M*);
C ← *get-propagation-reason M* (−*L*);
case C of
Some C ⇒ *do* {
ASSERT(*C* ∈# *dom-m NU*);
ASSERT(*length* (*NU* ∝ *C*) ≥ *2*);
ASSERT(*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*NU* ∝ *C*)));
ASSERT(*distinct* (*NU* ∝ *C*) ∧ ¬*tautology* (*mset* (*NU* ∝ *C*)));
ASSERT(*length* (*NU* ∝ *C*) ≤ *Suc* (*unat32-max div 2*));
lit-redundant-rec-wl-lookup $\mathcal{A}$ *M NU D cach* [*lit-redundant-reason-stack2* (−*L*) *NU C*] *lbd*
}
| *None* ⇒ *do* {
RETURN (*cach*, [], *False*)
}
}
}›

lemma *literal-redundant-wl-lookup-literal-redundant-wl*:
assumes ‹*M* ⊨*as CNot D*› ‹*no-dup M*› ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *M*›
‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ ((*mset* ∘ *fst*) ‘# *ran-m NU*)› **and**
‹*isasat-input-bounded* $\mathcal{A}$›
shows
‹*literal-redundant-wl-lookup* $\mathcal{A}$ *M NU D cach L lbd* ≤
⇓ (*Id* ×$_f$ (*ana-lookups-rel NU* ×$_f$ *bool-rel*)) (*literal-redundant-wl M NU D cach L lbd*)›
proof −
have *M*: ‹∀ *a* ∈ *lits-of-l M*. *a* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
using *literals-are-in-*$\mathcal{L}_{in}$*-trail-in-lits-of-l assms* **by** *blast*
have [*simp*, *intro*!]: ‹− *x1e* ∈ *lits-of-l M* ⟹ *atm-of x1e* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
‹− *x1e* ∈ *lits-of-l M* ⟹ *x1e* ∈# ($\mathcal{L}_{all}$ $\mathcal{A}$)› **for** *x1e*
using *assms atm-of-notin-atms-of-iff literals-are-in-*$\mathcal{L}_{in}$*-trail-in-lits-of-l* **apply** *blast*
using *M uminus-*$\mathcal{A}_{in}$*-iff* **by** *auto*
have [*refine*]: ‹(*x*, *x*′) ∈ *Id* ⟹ (*x*, *x*′) ∈ ⟨*Id*⟩*option-rel*› **for** *x x*′
by *auto*
have [*refine-vcg*]: ‹*get-propagation-reason M x*
≤ ⇓ ({(*C*, *C*′). (*C*, *C*′) ∈ ⟨*nat-rel*⟩*option-rel*})

(*get-propagation-reason M y*)› **if** ‹*x = y*› **and** ‹*y ∈ lits-of-l M*› **for** *x y*
by (*use that* **in** ‹*auto simp*: *get-propagation-reason-def intro*: *RES-refine*›)
show *?thesis*
unfolding *literal-redundant-wl-lookup-def literal-redundant-wl-def*
apply (*refine-vcg lit-redundant-rec-wl-lookup-lit-redundant-rec-wl*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal
using *assms* **by** (*auto dest*!: *multi-member-split simp*: *ran-m-def literals-are-in-*$\mathcal{L}_{in}$*-mm-add-mset*)
subgoal by *auto*
subgoal by *auto*
subgoal using *assms simple-clss-size-upper-div2*[*of* $\mathcal{A}$ ‹*mset* (*NU* ∝ -)›] **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal by (*auto simp*: *lit-redundant-reason-stack2-def lit-redundant-reason-stack-def*
ana-lookup-rel-def)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
done
qed

definition (**in** −) *lookup-conflict-nth* **where**
[*simp*]: ‹*lookup-conflict-nth* = (λ(-, *xs*) *i*. *xs* ! *i*)›

definition (**in** −) *lookup-conflict-size* **where**
[*simp*]: ‹*lookup-conflict-size* = (λ(*n*, *xs*). *n*)›

definition (**in** −) *lookup-conflict-upd-None* **where**
[*simp*]: ‹*lookup-conflict-upd-None* = (λ(*n*, *xs*) *i*. (*n*−*1*, *xs* [*i* :=*None*]))›

definition *minimize-and-extract-highest-lookup-conflict*
:: ‹*nat multiset* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *nat clauses-l* ⇒ *nat clause* ⇒ (*nat* ⇒ *minimize-status*) ⇒ *lbd*
⇒
out-learned ⇒ (*nat clause* × (*nat* ⇒ *minimize-status*) × *out-learned*) *nres*›
where
‹*minimize-and-extract-highest-lookup-conflict* $\mathcal{A}$ = (λ*M NU nxs s lbd outl*. *do* {
(*D*, -, *s*, *outl*) ←
WHILE$_T$*minimize-and-extract-highest-lookup-conflict-inv*
(λ(*nxs*, *i*, *s*, *outl*). *i* < *length outl*)
(λ(*nxs*, *x*, *s*, *outl*). *do* {
ASSERT(*x* < *length outl*);
let L = *outl* ! *x*;
ASSERT(*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$);
(*s'*, -, *red*) ← *literal-redundant-wl-lookup* $\mathcal{A}$ *M NU nxs s L lbd*;
if ¬*red*
then RETURN (*nxs*, *x*+*1*, *s'*, *outl*)
else do {
ASSERT (*delete-from-lookup-conflict-pre* $\mathcal{A}$ (*L*, *nxs*));

```
            RETURN (remove1-mset L nxs, x, s', delete-index-and-swap outl x)
          }
        })
        (nxs, 1, s, outl);
    RETURN (D, s, outl)
  })›

lemma entails-uminus-filter-to-poslev-can-remove:
  assumes NU-uL-E: ‹NU ⊨p add-mset (− L) (filter-to-poslev M' L E)› and
    NU-E: ‹NU ⊨p E› and L-E: ‹L ∈# E›
  shows ‹NU ⊨p remove1-mset L E›
proof −
  have ‹filter-to-poslev M' L E ⊆# remove1-mset L E›
    by (induction E)
      (auto simp add: filter-to-poslev-add-mset remove1-mset-add-mset-If subset-mset-trans-add-mset
       intro: diff-subset-eq-self subset-mset.dual-order.trans)
  then have ‹NU ⊨p add-mset (− L) (remove1-mset L E)›
    using NU-uL-E
    by (meson conflict-minimize-intermediate-step mset-subset-eqD)
  moreover have ‹NU ⊨p add-mset L (remove1-mset L E)›
    using NU-E L-E by auto
  ultimately show ?thesis
    using true-clss-cls-or-true-clss-cls-or-not-true-clss-cls-or[of NU L ‹remove1-mset L E›
      ‹remove1-mset L E›]
    by (auto simp: true-clss-cls-add-self)
qed

lemma minimize-and-extract-highest-lookup-conflict-iterate-over-conflict:
  fixes D :: ‹nat clause› and S' :: ‹nat twl-st-l› and NU :: ‹nat clauses-l› and S :: ‹nat twl-st-wl›
    and S'' :: ‹nat twl-st›
  defines
    ‹S''' ≡ state_W-of S''›
  defines
    ‹M ≡ get-trail-wl S› and
    NU: ‹NU ≡ get-clauses-wl S› and
    NU'-def: ‹NU' ≡ mset `# ran-mf NU› and
    NUE: ‹NUE ≡ get-unit-learned-clss-wl S + get-unit-init-clss-wl S› and
    NUS: ‹NUS ≡ get-subsumed-learned-clauses-wl S + get-subsumed-init-clauses-wl S› and
    N0S: ‹N0S ≡ get-learned-clauses0-wl S + get-init-clauses0-wl S› and
    M': ‹M' ≡ trail S'''›
  assumes
    S-S': ‹(S, S') ∈ state-wl-l None› and
    S'-S'': ‹(S', S'') ∈ twl-st-l None› and
    D'-D: ‹mset (tl outl) = D› and
    M-D: ‹M ⊨as CNot D› and
    dist-D: ‹distinct-mset D› and
    tauto: ‹¬tautology D› and
    lits: ‹literals-are-in-ℒ_in-trail 𝒜 M› and
    struct-invs: ‹twl-struct-invs S''› and
    add-inv: ‹twl-list-invs S'› and
    cach-init: ‹conflict-min-analysis-inv M' s' (NU' + NUE + NUS + N0S) D› and
    NU-P-D: ‹NU' + NUE + NUS + N0S ⊨pm add-mset K D› and
    lits-D: ‹literals-are-in-ℒ_in 𝒜 D› and
    lits-NU: ‹literals-are-in-ℒ_in-mm 𝒜 (mset `# ran-mf NU)› and
    K: ‹K = outl ! 0› and
    outl-nempty: ‹outl ≠ []› and
```

bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
shows
‹*minimize-and-extract-highest-lookup-conflict* $\mathcal{A}$ *M NU D s' lbd outl* ≤
⇓ ({((*E*, *s*, *outl*), *E'*). *E* = *E'* ∧ *mset* (*tl outl*) = *E* ∧ *outl* ! 0 = *K* ∧
E' ⊆# *D* ∧ *outl* ≠ []})
(*iterate-over-conflict K M NU'* (*NUE* + *NUS* + *N0S*) *D*)›
(**is** ‹- ≤ ⇓ *?R* -›)
proof −
let *?UE* = ‹*get-unit-learned-clss-wl S*›
let *?NE* = ‹*get-unit-init-clss-wl S*›
let *?US* = ‹*get-subsumed-learned-clauses-wl S*›
let *?NS* = ‹*get-subsumed-init-clauses-wl S*›
let *?U0* = ‹*get-learned-clauses0-wl S*›
let *?N0* = ‹*get-init-clauses0-wl S*›
define *N U* **where**
‹*N* ≡ *mset* '# *init-clss-lf NU*› **and**
‹*U* ≡ *mset* '# *learned-clss-lf NU*›
obtain *E* **where**
S''': ‹*S'''* = (*M'*, *N* + *?NE* + *?NS* + *?N0*, *U* + *?UE* + *?US* + *?U0*, *E*)›
using *M' S-S' S'-S''* **unfolding** *S'''-def N-def U-def NU*
by (*cases S*) (*auto simp*: *state-wl-l-def twl-st-l-def*
mset-take-mset-drop-mset')
then have *NU-N-U*: ‹*mset* '# *ran-mf NU* = *N* + *U*›
using *NU S-S' S'-S''* **unfolding** *S'''-def N-def U-def*
apply (*subst all-clss-l-ran-m*[*symmetric*])
apply (*subst image-mset-union*[*symmetric*])
apply (*subst image-mset-union*[*symmetric*])
by (*auto simp*: *mset-take-mset-drop-mset'*)
let *?NU* = ‹*N* + *?NE* + *?NS* + *?N0* + *U* + *?UE* + *?US* + *?U0*›
have *NU'-N-U*: ‹*NU'* = *N* + *U*›
unfolding *NU'-def N-def U-def mset-append*[*symmetric*] *image-mset-union*[*symmetric*]
by *auto*
have *NU'-NUE*: ‹*NU'* + *NUE* = *N* + *get-unit-init-clss-wl S* + *U* + *get-unit-learned-clss-wl S*›
unfolding *NUE NU'-N-U* **by** (*auto simp*: *ac-simps*)
have *struct-inv-S'''*: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*M'*, *N* + (*?NE* + *?NS* + *?N0*),
U + (*?UE* + *?US* + *?U0*), *E*)›
using *struct-invs* **unfolding** *twl-struct-invs-def S'''-def*[*symmetric*] *S''' add.assoc*
pcdcl-all-struct-invs-def state$_W$*-of-def*[*symmetric*]
by *fast*
then have *n-d*: ‹*no-dup M'*›
unfolding *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-M-level-inv-def*
trail.simps **by** *fast*
then have *n-d*: ‹*no-dup M*›
using *S-S' S'-S''* **unfolding** *M-def M' S'''-def* **by** (*auto simp*: *twl-st-wl twl-st-l twl-st*)

define *R* **where**
‹*R* = {((*D'*:: *nat clause*, *i*, *cach* :: *nat* ⇒ *minimize-status*, *outl'* :: *out-learned*),
(*F* :: *nat clause*, *E* :: *nat clause*)).
i ≤ *length outl'* ∧
F ⊆# *D* ∧
E ⊆# *F* ∧
mset (*drop i outl'*) = *E* ∧
mset (*tl outl'*) = *F* ∧
conflict-min-analysis-inv M' cach (*?NU*) *F* ∧
?NU ⊨*pm add-mset K F* ∧
mset (*tl outl'*) = *D'* ∧

```
        i > 0 ∧ outl' ≠ [] ∧
        outl' ! 0 = K
    }›
have [simp]: ‹add-mset K (mset (tl outl)) = mset outl›
  using D'-D K
  by (cases outl) (auto simp: drop-Suc outl-nempty)
have ‹Suc 0 < length outl ⟹
  highest-lit M (mset (take (Suc 0) (tl outl)))
   (Some (outl ! Suc 0, get-level M (outl ! Suc 0)))›
  using outl-nempty
  by (cases outl; cases ‹tl outl›)  (auto simp: highest-lit-def get-maximum-level-add-mset)
 then have init-args-ref: ‹((D, 1, s', outl), D, D) ∈ R›
  using D'-D cach-init NU-P-D dist-D tauto K
  unfolding R-def NUE NU'-def NU-N-U NUS N0S
  by (auto simp: ac-simps drop-Suc outl-nempty ac-simps)

 have init-lo-inv: ‹minimize-and-extract-highest-lookup-conflict-inv s'›
  if
    ‹(s', s) ∈ R› and
    ‹iterate-over-conflict-inv M D s›
  for s' s
 proof −
   have [dest!]: ‹mset b ⊆# D ⟹ length b ≤ size D› for b
     using size-mset-mono by fastforce
  show ?thesis
    using that simple-clss-size-upper-div2[OF bounded lits-D dist-D tauto]
    unfolding minimize-and-extract-highest-lookup-conflict-inv-def
    by (auto simp: R-def unat32-max-def)
qed
have cond: ‹(m < length outl') = (D' ≠ {#})›
  if
    st'-st: ‹(st', st) ∈ R› and
    ‹minimize-and-extract-highest-lookup-conflict-inv st'› and
    ‹iterate-over-conflict-inv M D st› and
    st:
      ‹x2b = (j, outl')›
      ‹x2a = (m, x2b)›
      ‹st' = (nxs, x2a)›
      ‹st = (E, D')›
  for st' st nxs x2a m x2b j x2c D' E st2 st3 outl'
proof −
  show ?thesis
    using st'-st unfolding st R-def
    by auto
qed

have redundant: ‹literal-redundant-wl-lookup 𝒜 M NU nxs cach
       (outl' ! x1d) lbd
   ≤ ⇓ {((s', a', b'), b). b = b' ∧
        (b ⟶ ?NU ⊨pm remove1-mset L (add-mset K E) ∧
           conflict-min-analysis-inv M' s' ?NU (remove1-mset L E)) ∧
        (¬b ⟶ ?NU ⊨pm add-mset K E ∧ conflict-min-analysis-inv M' s' ?NU E)}
       (is-literal-redundant-spec K NU' (NUE+NUS+N0S) E L)›
  (is ‹- ≤ ⇓ ?red -›)
  if
    R: ‹(x, x') ∈ R› and
```

‹case x′ of (D, D′) ⇒ D′ ≠ {#}› **and**
‹minimize-and-extract-highest-lookup-conflict-inv x› **and**
‹iterate-over-conflict-inv M D x′› **and**
st:
‹x′ = (E, x1a)›
‹x2d = (cach, outl′)›
‹x2c = (x1d, x2d)›
‹x = (nxs, x2c)› **and**
L: ‹(outl′!x1d, L) ∈ Id›
‹x1d < length outl′›
for *x x′ E x2 x1a x2a nxs x2c x1d x2d x1e x2e cach highest L outl′ st3*
proof −
let *?L* = ‹(outl′ ! x1d)›
have
‹x1d < length outl′› **and**
‹x1d ≤ length outl′› **and**
‹mset (tl outl′) ⊆# D› **and**
‹E = mset (tl outl′)› **and**
cach: ‹conflict-min-analysis-inv M′ cach ?NU E› **and**
NU-P-E: ‹?NU ⊨pm add-mset K (mset (tl outl′))› **and**
‹nxs = mset (tl outl′)› **and**
‹0 < x1d› **and**
[*simp*]: ‹L = outl′!x1d› **and**
‹E ⊆# D›
‹E = mset (tl outl′)› **and**
‹E = nxs›
using *R L* **unfolding** *R-def st*
by *auto*

have *M-x1*: ‹M ⊨as CNot E›
by (*metis CNot-plus M-D* ‹E ⊆# D› *subset-mset.le-iff-add true-annots-union*)
then have *M′-x1*: ‹M′ ⊨as CNot E›
using *S-S′ S′-S″* **unfolding** *M′ M-def S‴-def* **by** (*auto simp*: *twl-st twl-st-wl twl-st-l*)
have ‹outl′ ! x1d ∈# E›
using ‹E = mset (tl outl′)› ‹x1d < length outl′› ‹0 < x1d›
by (*auto simp*: *nth-in-set-tl*)

have *1*:
‹literal-redundant-wl-lookup 𝒜 M NU nxs cach ?L lbd ≤ ⇓ (Id ×f (ana-lookups-rel NU ×f bool-rel))
(literal-redundant-wl M NU nxs cach ?L lbd)›
by (*rule literal-redundant-wl-lookup-literal-redundant-wl*)
(*use lits-NU n-d lits M-x1 struct-invs bounded add-inv* ‹outl′ ! x1d ∈# E› ‹E = nxs› **in** *auto*)

have *2*:
‹literal-redundant-wl M NU nxs cach ?L lbd ≤ ⇓
(Id ×r {(analyse, analyse′). analyse′ = convert-analysis-list NU analyse ∧
lit-redundant-rec-wl-ref NU analyse} ×r bool-rel)
(literal-redundant M′ NU′ nxs cach ?L)›
by (*rule literal-redundant-wl-literal-redundant*[*of S S′ S″*,
unfolded M-def[*symmetric*] *NU*[*symmetric*] *M′*[*symmetric*] *S‴-def*[*symmetric*]
NU′-def[*symmetric*], *THEN order-trans*])
(*use bounded S-S′ S′-S″ M-x1 struct-invs add-inv* ‹outl′ ! x1d ∈# E› ‹E = nxs› **in**
‹*auto simp*: *NU*›)

have *NU-alt-def*: ‹?NU = N + (?NE + ?NS + ?N0) + U + (?UE + ?US + ?U0)›
by (*auto simp*: *ac-simps*)

```
  have 3:
    ‹literal-redundant M' (N + U) nxs cach ?L ≤
      literal-redundant-spec M' (N + U + (?NE + ?NS + ?N0) + (?UE + ?US + ?U0)) nxs ?L›
    unfolding ‹E = nxs›[symmetric]
    apply (rule literal-redundant-spec)
      apply (rule struct-inv-S''')
    apply (rule cach[unfolded NU-alt-def])
     apply (rule ‹outl' ! x1d ∈# E›)
    apply (rule M'-x1)
    done

  then have 3:
    ‹literal-redundant M' (NU') nxs cach ?L ≤ literal-redundant-spec M' ?NU nxs ?L›
    by (auto simp: ac-simps NU'-N-U)

  have ent: ‹?NU ⊨pm add-mset (− L) (filter-to-poslev M' L (add-mset K E))›
    if ‹?NU ⊨pm add-mset (− L) (filter-to-poslev M' L E)›
    using that by (auto simp: filter-to-poslev-add-mset add-mset-commute)
  show ?thesis
    apply (rule order.trans)
     apply (rule 1)
    apply (rule order.trans)
    apply (rule ref-two-step')
     apply (rule 2)
     apply (subst conc-fun-chain)
    apply (rule order.trans)
     apply (rule ref-two-step'[OF 3])
    unfolding literal-redundant-spec-def is-literal-redundant-spec-def
        conc-fun-SPEC NU'-NUE[symmetric]
    apply (rule SPEC-rule)
    apply clarify
    using NU-P-E ent ‹E = nxs› ‹E = mset (tl outl')›[symmetric] ‹outl' ! x1d ∈# E› NU'-NUE
    by (auto intro!: entails-uminus-filter-to-poslev-can-remove[of - - M']
      filter-to-poslev-conflict-min-analysis-inv
      simp: NUE NUS ac-simps N0S ac-simps simp del: diff-union-swap2)
qed

have
  outl'-F: ‹outl' ! i ∈# F› (is ?out) and
  outl'-L_all: ‹outl' ! i ∈# L_all A› (is ?out-L)
  if
    R: ‹(S, T) ∈ R› and
    ‹case S of (nxs, i, s, outl) ⇒ i < length outl› and
    ‹case T of (D, D') ⇒ D' ≠ {#}› and
    ‹minimize-and-extract-highest-lookup-conflict-inv S› and
    ‹iterate-over-conflict-inv M D T› and
    st:
      ‹T = (F', F)›
      ‹S2 = (cach, outl')›
      ‹S1 = (i, S2)›
      ‹S = (D', S1)›
    ‹i < length outl'›
  for S T F' T1 F highest' D' S1 i S2 cach S3 highest outl'
proof −
  have ?out and ‹F ⊆# D›
    using R ‹i < length outl'› unfolding R-def st
```

by (*auto simp*: *set-drop-conv*)
show *?out*
using ‹*?out*› **.**
then have ‹*outl′* ! *i* ∈# *D*›
using ‹*F* ⊆# *D*› **by** *auto*
then show *?out-L*
using *lits-D* **by** (*auto dest*!: *multi-member-split simp*: *literals-are-in-*$\mathcal{L}_{in}$*-add-mset*)
qed

have
not-red: ‹¬ *red* ⟹ ((*D′*, *i* + *1*, *cachr*, *outl′*), *F′*,
remove1-mset L F) ∈ *R*› (**is** ‹- ⟹ *?not-red*›) **and**
red: ‹¬ ¬ *red* ⟹
((*remove1-mset* (*outl′* ! *i*) *D′*, *i*, *cachr*, *delete-index-and-swap outl′ i*),
remove1-mset L F′, *remove1-mset L F*) ∈ *R*› (**is** ‹- ⟹ *?red*›) **and**
del: ‹*delete-from-lookup-conflict-pre* $\mathcal{A}$ (*outl′* ! *i*, *D′*)› (**is** *?del*)
if
R: ‹(*S*, *T*) ∈ *R*› **and**
‹*case S of* (*nxs*, *i*, *s*, *outl*) ⇒ *i* < *length outl*› **and**
‹*case T of* (*D*, *D′*) ⇒ *D′* ≠ {#}› **and**
‹*minimize-and-extract-highest-lookup-conflict-inv S*› **and**
‹*iterate-over-conflict-inv M D T*› **and**
st:
‹*T* = (*F′*, *F*)›
‹*S2* = (*cach*, *outl′*)›
‹*S1* = (*i*, *S2*)›
‹*S* = (*D′*, *S1*)›
‹*cachred1* = (*stack*, *red*)›
‹*cachred* = (*cachr*, *cachred1*)› **and**
‹*i* < *length outl′*› **and**
L: ‹(*outl′* ! *i*, *L*) ∈ *Id*› **and**
‹*outl′* ! *i* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
cach: ‹(*cachred*, *red′*) ∈ (*?red F′ L*)›
for *S T F′ T1 F D′ S1 i S2 cach S3 highest outl′ L cachred red′ cachr*
cachred1 stack red
proof −
have ‹*L* = *outl′* ! *i*› **and**
‹*i* ≤ *length outl′*› **and**
‹*mset* (*tl outl′*) ⊆# *D*› **and**
‹*mset* (*drop i outl′*) ⊆# *mset* (*tl outl′*)› **and**
F: ‹*F* = *mset* (*drop i outl′*)› **and**
F′: ‹*F′* = *mset* (*tl outl′*)› **and**
‹*conflict-min-analysis-inv M′ cach ?NU* (*mset* (*tl outl′*))› **and**
‹*?NU* ⊨*pm add-mset K* (*mset* (*tl outl′*))› **and**
‹*D′* = *mset* (*tl outl′*)› **and**
‹*0* < *i*› **and**
[*simp*]: ‹*D′* = *F′*› **and**
F′-D: ‹*F′* ⊆# *D*› **and**
F′-F: ‹*F* ⊆# *F′*› **and**
‹*outl′* ≠ []› ‹*outl′* ! *0* = *K*›
using *R L* **unfolding** *R-def st*
by *clarify+*

have [*simp*]: ‹*L* = *outl′* ! *i*›
using *L* **by** *fast*
have *L-F*: ‹*mset* (*drop* (*Suc i*) *outl′*) = *remove1-mset L F*›

unfolding *F*
apply (*subst* (*2*) *Cons-nth-drop-Suc*[*symmetric*])
using ‹*i* < *length outl'*› *F'-D*
by (*auto*)
have ‹*remove1-mset* (*outl'* ! *i*) *F* ⊆# *F'*›
using ‹*F* ⊆# *F'*›
by *auto*
have ‹*red'* = *red*› **and**
red: ‹*red* ⟶ *?NU* ⊨*pm remove1-mset L* (*add-mset K F'*) ∧
conflict-min-analysis-inv M' cachr ?NU (*remove1-mset L F'*)› **and**
not-red: ‹¬ *red* ⟶ *?NU* ⊨*pm add-mset K F'* ∧ *conflict-min-analysis-inv M' cachr ?NU F'*›
using *cach*
unfolding *st*
by *auto*
have [*simp*]: ‹*mset* (*drop* (*Suc i*) (*swap outl'* (*Suc 0*) *i*)) = *mset* (*drop* (*Suc i*) *outl'*)›
by (*subst drop-swap-irrelevant*) (*use* ‹*0* < *i*› **in** *auto*)
have [*simp*]: ‹*mset* (*tl* (*swap outl'* (*Suc 0*) *i*)) = *mset* (*tl outl'*)›
apply (*cases outl'*; *cases i*)
using ‹*i* > *0*› ‹*outl'* ≠ []› ‹*i* < *length outl'*›
apply (*auto simp*: *WB-More-Refinement-List.swap-def*)
unfolding *WB-More-Refinement-List.swap-def*[*symmetric*]
by (*auto simp*:)
have [*simp*]: ‹*mset* (*take* (*Suc i*) (*tl* (*swap outl'* (*Suc 0*) *i*))) = *mset* (*take* (*Suc i*) (*tl outl'*))›
using ‹*i* > *0*› ‹*outl'* ≠ []› ‹*i* < *length outl'*›
by (*auto simp*: *take-tl take-swap-relevant tl-swap-relevant*)
have [*simp*]: ‹*mset* (*take i* (*tl* (*swap outl'* (*Suc 0*) *i*))) = *mset* (*take i* (*tl outl'*))›
using ‹*i* > *0*› ‹*outl'* ≠ []› ‹*i* < *length outl'*›
by (*auto simp*: *take-tl take-swap-relevant tl-swap-relevant*)

have [*simp*]: ‹¬ *Suc 0* < *a* ⟷ *a* = *0* ∨ *a* = *1*› **for** *a* :: *nat*
by *auto*
show *?not-red* **if** ‹¬*red*›
using ‹*i* < *length outl'*› *F'-D L-F* ‹*remove1-mset* (*outl'* ! *i*) *F* ⊆# *F'*› *not-red that*
‹*i* > *0*› ‹*outl'* ! *0* = *K*›
by (*auto simp*: *R-def F*[*symmetric*] *F'*[*symmetric*] *drop-swap-irrelevant*)

have [*simp*]: ‹*length* (*delete-index-and-swap outl' i*) = *length outl'* − *1*›
by *auto*
have *last*: ‹¬ *length outl'* ≤ *Suc i* ⟹*last outl'* ∈ *set* (*drop* (*Suc i*) *outl'*)›
by (*metis List.last-in-set drop-eq-Nil last-drop not-le-imp-less*)
then have *H*: ‹*mset* (*drop i* (*delete-index-and-swap outl' i*)) = *mset* (*drop* (*Suc i*) *outl'*)›
using ‹*i* < *length outl'*›
by (*cases* ‹*drop* (*Suc i*) *outl'* = []›)
(*auto simp*: *butlast-list-update mset-butlast-remove1-mset*)
have *H'*: ‹*mset* (*tl* (*delete-index-and-swap outl' i*)) = *remove1-mset* (*outl'* ! *i*) (*mset* (*tl outl'*))›
apply (*rule mset-tl-delete-index-and-swap*)
using ‹*i* < *length outl'*› ‹*i* > *0*› **by** *fast+*
have [*simp*]: ‹*Suc 0* < *i* ⟹ *delete-index-and-swap outl' i* ! *Suc 0* = *outl'* ! *Suc 0*›
using ‹*i* < *length outl'*› ‹*i* > *0*›
by (*auto simp*: *nth-butlast*)
have ‹*remove1-mset* (*outl'* ! *i*) *F* ⊆# *remove1-mset* (*outl'* ! *i*) *F'*›
using ‹*F* ⊆# *F'*›
using *mset-le-subtract* **by** *blast*
have [*simp*]: ‹*delete-index-and-swap outl' i* ≠ []›
using ‹*outl'* ≠ []› ‹*i* > *0*› ‹*i* < *length outl'*›
by (*cases outl'*) (*auto simp*: *butlast-update'*[*symmetric*] *split*: *nat.splits*)

```
    have [simp]: ‹delete-index-and-swap outl' i ! 0 = outl' ! 0›
      using ‹outl' ! 0 = K› ‹i < length outl'› ‹i > 0›
      by (auto simp: butlast-update'[symmetric] nth-butlast)
    have ‹(outl' ! i) ∈# F'›
      using ‹i < length outl'› ‹i > 0› unfolding F' by (auto simp: nth-in-set-tl)
    then show ?red if ‹¬¬red›
      using ‹i < length outl'› F'-D L-F ‹remove1-mset (outl' ! i) F ⊆# remove1-mset (outl' ! i) F'›
        red that ‹i > 0› ‹outl' ! 0 = K› unfolding R-def
      by (auto simp: R-def F[symmetric] F'[symmetric] H H' drop-swap-irrelevant
          simp del: delete-index-and-swap.simps)

    have ‹outl' ! i ∈# ℒ_all 𝒜› ‹outl' ! i ∈# D›
      using ‹(outl' ! i) ∈# F'› F'-D lits-D
      by (force simp: literals-are-in-ℒ_in-add-mset
          dest!: multi-member-split[of ‹outl' ! i› D])+
    then show ?del
      using ‹(outl' ! i) ∈# F'› lits-D F'-D tauto
      by (auto simp: delete-from-lookup-conflict-pre-def
          literals-are-in-ℒ_in-add-mset)
  qed
  show ?thesis
    unfolding minimize-and-extract-highest-lookup-conflict-def iterate-over-conflict-def
    apply (refine-vcg WHILEIT-refine[where R = R])
    subgoal by (rule init-args-ref)
    subgoal for s' s by (rule init-lo-inv)
    subgoal by (rule cond)
    subgoal by auto
    subgoal by (rule outl'-F)
    subgoal by (rule outl'-ℒ_all)
    apply (rule redundant; assumption)
    subgoal by auto
    subgoal by (rule not-red)
    subgoal by (rule del)
    subgoal
      by (rule red)
    subgoal for x x' x1 x2 x1a x2a x1b x2b x1c x2c
      unfolding R-def by (cases x1b) auto
    done
qed

definition cach-refinement-list
  :: ‹nat multiset ⇒ (minimize-status list × (nat conflict-min-cach)) set›
where
  ‹cach-refinement-list 𝒜_in = ⟨Id⟩map-fun-rel {(a, a'). a = a' ∧ a ∈# 𝒜_in}›

definition cach-refinement-nonull
  :: ‹nat multiset ⇒ ((minimize-status list × nat list) × minimize-status list) set›
where
  ‹cach-refinement-nonull 𝒜 = {((cach, support), cach'). cach = cach' ∧
      (∀ L < length cach. cach ! L ≠ SEEN-UNKNOWN ⟷ L ∈ set support) ∧
      (∀ L ∈ set support. L < length cach) ∧
      distinct support ∧ set support ⊆ set-mset 𝒜}›


definition cach-refinement
  :: ‹nat multiset ⇒ ((minimize-status list × nat list) × (nat conflict-min-cach)) set›
```

where
‹*cach-refinement* $\mathcal{A}_{in}$ = *cach-refinement-nonull* $\mathcal{A}_{in}$ *O cach-refinement-list* $\mathcal{A}_{in}$›

lemma *cach-refinement-alt-def*:
‹*cach-refinement* $\mathcal{A}_{in}$ = {((*cach*, *support*), *cach'*).
(∀ *L* < *length cach*. *cach* ! *L* ≠ *SEEN-UNKNOWN* ⟷ *L* ∈ *set support*) ∧
(∀ *L* ∈ *set support*. *L* < *length cach*) ∧
(∀ *L* ∈# $\mathcal{A}_{in}$. *L* < *length cach* ∧ *cach* ! *L* = *cach'* *L*) ∧
distinct support ∧ *set support* ⊆ *set-mset* $\mathcal{A}_{in}$}›
unfolding *cach-refinement-def cach-refinement-nonull-def cach-refinement-list-def*
apply (*rule*; *rule*)
apply (*simp add*: *map-fun-rel-def split*: *prod.splits*)
apply *blast*
apply (*simp add*: *map-fun-rel-def split*: *prod.splits*)
apply (*rule-tac b*=*x1a* **in** *relcomp.relcompI*)
apply *blast*
apply *blast*
done

lemma *in-cach-refinement-alt-def*:
‹((*cach*, *support*), *cach'*) ∈ *cach-refinement* $\mathcal{A}_{in}$ ⟷
(*cach*, *cach'*) ∈ *cach-refinement-list* $\mathcal{A}_{in}$ ∧
(∀ *L*<*length cach*. *cach* ! *L* ≠ *SEEN-UNKNOWN* ⟷ *L* ∈ *set support*) ∧
(∀ *L* ∈ *set support*. *L* < *length cach*) ∧
distinct support ∧ *set support* ⊆ *set-mset* $\mathcal{A}_{in}$›
by (*auto simp*: *cach-refinement-def cach-refinement-nonull-def cach-refinement-list-def*)

definition (**in** −) *conflict-min-cach-l* :: ‹*conflict-min-cach-l* ⇒ *nat* ⇒ *minimize-status*› **where**
‹*conflict-min-cach-l* = (λ(*cach*, *sup*) *L*.
(*cach* ! *L*)
)›

definition *conflict-min-cach-l-pre* **where**
‹*conflict-min-cach-l-pre* = (λ((*cach*, *sup*), *L*). *L* < *length cach*)›

lemma *conflict-min-cach-l-pre*:
fixes *x1* :: ‹*nat*› **and** *x2* :: ‹*nat*›
assumes
‹*x1n* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
‹(*x1l*, *x1j*) ∈ *cach-refinement* $\mathcal{A}$›
shows ‹*conflict-min-cach-l-pre* (*x1l*, *atm-of x1n*)›
proof −
show *?thesis*
using *assms* **by** (*auto simp*: *cach-refinement-alt-def in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *conflict-min-cach-l-pre-def*)
qed

lemma *nth-conflict-min-cach*:
‹(*uncurry* (*RETURN oo conflict-min-cach-l*), *uncurry* (*RETURN oo conflict-min-cach*)) ∈
[λ(*cach*, *L*). *L* ∈# $\mathcal{A}_{in}$]$_f$ *cach-refinement* $\mathcal{A}_{in}$ ×$_r$ *nat-rel* → ⟨*Id*⟩*nres-rel*›
by (*intro frefI nres-relI*) (*auto simp*: *map-fun-rel-def*
in-cach-refinement-alt-def cach-refinement-list-def conflict-min-cach-l-def)

definition (**in** −) *conflict-min-cach-set-failed*
:: ‹*nat conflict-min-cach* ⇒ *nat* ⇒ *nat conflict-min-cach*›
where

[*simp*]: ‹*conflict-min-cach-set-failed cach L = cach*(*L* := *SEEN-FAILED*)›

definition (**in** −) *conflict-min-cach-set-failed-l*
:: ‹*conflict-min-cach-l* ⇒ *nat* ⇒ *conflict-min-cach-l nres*›
where
‹*conflict-min-cach-set-failed-l* = (λ(*cach*, *sup*) *L*. *do* {
ASSERT(*L* < *length cach*);
ASSERT(*length sup* ≤ *1* + *unat32-max div 2*);
RETURN (*cach*[*L* := *SEEN-FAILED*], *if cach* ! *L* = *SEEN-UNKNOWN then sup* @ [*L*] *else sup*)
})›

lemma *bounded-included-le*:
assumes *bounded*: ‹*isasat-input-bounded* $\mathcal{A}$› **and** ‹*distinct n*› **and** ‹*set n* ⊆ *set-mset* $\mathcal{A}$›
shows ‹*length n* ≤ *Suc* (*unat32-max div 2*)›
proof −
have *lits*: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*Pos* ‘# *mset n*)› **and**
dist: ‹*distinct n*›
using *assms*
by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-alt-def inj-on-def atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$)
have *dist*: ‹*distinct-mset* (*Pos* ‘# *mset n*)›
by (*subst distinct-image-mset-inj*)
(*use dist* **in** ‹*auto simp*: *inj-on-def*›)
have *tauto*: ‹¬ *tautology* (*poss* (*mset n*))›
by (*auto simp*: *tautology-decomp*)

show *?thesis*
using *simple-clss-size-upper-div2*[*OF bounded lits dist tauto*]
by (*auto simp*: *unat32-max-def*)
qed

lemma *conflict-min-cach-set-failed*:
‹(*uncurry conflict-min-cach-set-failed-l*, *uncurry* (*RETURN oo conflict-min-cach-set-failed*)) ∈
[λ(*cach*, *L*). *L* ∈# $\mathcal{A}_{in}$ ∧ *isasat-input-bounded* $\mathcal{A}_{in}$]$_f$ *cach-refinement* $\mathcal{A}_{in}$ ×$_r$ *nat-rel* → ⟨*cach-refinement* $\mathcal{A}_{in}$⟩*nres-rel*›
supply *isasat-input-bounded-def*[*simp del*]
apply (*intro frefI nres-relI*)
apply (*auto simp*: *in-cach-refinement-alt-def map-fun-rel-def cach-refinement-list-def*
conflict-min-cach-set-failed-l-def cach-refinement-nonull-def
all-conj-distrib intro!: *ASSERT-leI bounded-included-le*[*of* $\mathcal{A}_{in}$]
dest!: *multi-member-split dest*: *set-mset-mono*
dest: *subset-add-mset-notin-subset-mset*)
by (*fastforce dest*: *subset-add-mset-notin-subset-mset*)+

definition (**in** −) *conflict-min-cach-set-removable*
:: ‹*nat conflict-min-cach* ⇒ *nat* ⇒ *nat conflict-min-cach*›
where
[*simp*]: ‹*conflict-min-cach-set-removable cach L = cach*(*L*:= *SEEN-REMOVABLE*)›

lemma *conflict-min-cach-set-removable*:
‹(*uncurry conflict-min-cach-set-removable-l*,
uncurry (*RETURN oo conflict-min-cach-set-removable*)) ∈
[λ(*cach*, *L*). *L* ∈# $\mathcal{A}_{in}$ ∧ *isasat-input-bounded* $\mathcal{A}_{in}$]$_f$ *cach-refinement* $\mathcal{A}_{in}$ ×$_r$ *nat-rel* → ⟨*cach-refinement* $\mathcal{A}_{in}$⟩*nres-rel*›
supply *isasat-input-bounded-def*[*simp del*]
by (*intro frefI nres-relI*)
(*auto 5 5 simp*: *in-cach-refinement-alt-def map-fun-rel-def cach-refinement-list-def*

conflict-min-cach-set-removable-l-def cach-refinement-nonull-def
all-conj-distrib intro!: *ASSERT-leI bounded-included-le*[*of* $\mathcal{A}_{in}$]
dest!: *multi-member-split dest*: *set-mset-mono*
dest: *subset-add-mset-notin-subset-mset*)

definition *isa-mark-failed-lits-stack* **where**
‹*isa-mark-failed-lits-stack NU analyse cach* = *do* {
let l = *length analyse*;
ASSERT(*length analyse* ≤ 1 + *unat32-max div 2*);
(-, *cach*) ← $WHILE_T^{\lambda(\text{-}, cach).\ True}$
(λ(*i*, *cach*). *i* < *l*)
(λ(*i*, *cach*). *do* {
ASSERT(*i* < *length analyse*);
let (*cls-idx*, *idx*, -) = (*analyse* ! *i*);
ASSERT(*cls-idx* + *idx* ≥ 1);
ASSERT(*cls-idx* + *idx* − 1 < *length NU*);
ASSERT(*arena-lit-pre NU* (*cls-idx* + *idx* − 1));
cach ← *conflict-min-cach-set-failed-l cach* (*atm-of* (*arena-lit NU* (*cls-idx* + *idx* − 1)));
RETURN (*i*+1, *cach*)
})
(0, *cach*);
RETURN cach
}›

context
begin
lemma *mark-failed-lits-stack-inv-helper1*: ‹*mark-failed-lits-stack-inv a ba a2'* ⟹
a1' < *length ba* ⟹
(*a1'a*, *a2'a*) = *ba* ! *a1'* ⟹
a1'a ∈# *dom-m a*›
using *nth-mem*[*of a1' ba*] **unfolding** *mark-failed-lits-stack-inv-def*
by (*auto simp del*: *nth-mem*)

lemma *mark-failed-lits-stack-inv-helper2*: ‹*mark-failed-lits-stack-inv a ba a2'* ⟹
a1' < *length ba* ⟹
(*a1'a*, *xx*, *a2'a*, *yy*) = *ba* ! *a1'* ⟹
a2'a − *Suc* 0 < *length* (*a* ∝ *a1'a*)›
using *nth-mem*[*of a1' ba*] **unfolding** *mark-failed-lits-stack-inv-def*
by (*auto simp del*: *nth-mem*)

lemma *isa-mark-failed-lits-stack-isa-mark-failed-lits-stack*:
assumes ‹*isasat-input-bounded* $\mathcal{A}_{in}$›
shows ‹(*uncurry2 isa-mark-failed-lits-stack*, *uncurry2* (*mark-failed-lits-stack* $\mathcal{A}_{in}$)) ∈
[λ((*N*, *ana*), *cach*). *length ana* ≤ 1 + *unat32-max div 2*]$_f$
{(*arena*, *N*). *valid-arena arena N vdom*} ×$_f$ *ana-lookups-rel NU* ×$_f$ *cach-refinement* $\mathcal{A}_{in}$ →
⟨*cach-refinement* $\mathcal{A}_{in}$⟩*nres-rel*›
proof −
have *subset-mset-add-new*: ‹*a* ∉# *A* ⟹ *a* ∈# *B* ⟹ *add-mset a A* ⊆# *B* ⟷ *A* ⊆# *B*› **for** *a A B*
by (*metis insert-DiffM insert-subset-eq-iff subset-add-mset-notin-subset*)
have [*refine0*]: ‹((0, *x2c*), 0, *x2a*) ∈ *nat-rel* ×$_f$ *cach-refinement* $\mathcal{A}_{in}$›
if ‹(*x2c*, *x2a*) ∈ *cach-refinement* $\mathcal{A}_{in}$› **for** *x2c x2a*
using *that* **by** *auto*
have *le-length-arena*: ‹*x1g* + *x2g* − 1 < *length x1c*› (**is** *?le*) **and**

is-lit: ‹*arena-lit-pre x1c* (*x1g* + *x2g* − *1*)› (**is** *?lit*) **and**
isA: ‹*atm-of* (*arena-lit x1c* (*x1g* + *x2g* − *1*)) ∈# $\mathcal{A}_{in}$› (**is** *?A*) **and**
final: ‹*conflict-min-cach-set-failed-l x2e*
(*atm-of* (*arena-lit x1c* (*x1g* + *x2g* − *1*)))
≤ *SPEC*
(λ*cach*.
RETURN (*x1e* + *1*, *cach*)
≤ *SPEC*
(λ*c*. (*c*, *x1d* + *1*, *x2d*
(*atm-of* (*x1a* ∝ *x1f* ! (*x2f* − *1*)) := *SEEN-FAILED*))
∈ *nat-rel* ×$_f$ *cach-refinement* $\mathcal{A}_{in}$))› (**is** *?final*) **and**
ge1: ‹*x1g* + *x2g* ≥ *1*›
if
‹*case y of* (*x*, *xa*) ⇒ (*case x of* (*N*, *ana*) ⇒ λ*cach*. *length ana* ≤ *1* + *unat32-max div 2*) *xa*› **and**
xy: ‹(*x*, *y*) ∈ {(*arena*, *N*). *valid-arena arena N vdom*} ×$_f$ *ana-lookups-rel NU*
×$_f$ *cach-refinement* $\mathcal{A}_{in}$› **and**
st:
‹*x1* = (*x1a*, *x2*)›
‹*y* = (*x1*, *x2a*)›
‹*x1b* = (*x1c*, *x2b*)›
‹*x* = (*x1b*, *x2c*)›
‹*x′* = (*x1d*, *x2d*)›
‹*xa* = (*x1e*, *x2e*)›
‹*x2f2* = (*x2f*, *x2f3*)›
‹*x2f0* = (*x2f1*, *x2f2*)›
‹*x2* ! *x1d* = (*x1f*, *x2f0*)›
‹*x2g0* = (*x2g*, *x2g2*)›
‹*x2b* ! *x1e* = (*x1g*, *x2g0*)› **and**
xax′: ‹(*xa*, *x′*) ∈ *nat-rel* ×$_f$ *cach-refinement* $\mathcal{A}_{in}$› **and**
cond: ‹*case xa of* (*i*, *cach*) ⇒ *i* < *length x2b*› **and**
cond′: ‹*case x′ of* (*i*, *cach*) ⇒ *i* < *length x2*› **and**
inv: ‹*case x′ of* (-, *x*) ⇒ *mark-failed-lits-stack-inv x1a x2 x*› **and**
le: ‹*x1d* < *length x2*› ‹*x1e* < *length x2b*› **and**
atm: ‹*atm-of* (*x1a* ∝ *x1f* ! (*x2f* − *1*)) ∈# $\mathcal{A}_{in}$›
for *x y x1 x1a x2 x2a x1b x1c x2b x2c xa x′ x1d x2d x1e x2e x1f x2f x1g x2g*
x2f0 x2f1 x2f2 x2f3 x2g0 x2g1 x2g2 x2g3
proof −
obtain *i cach* **where** *x′*: ‹*x′* = (*i*, *cach*)› **by** (*cases x′*)
have [*simp*]:
‹*x1* = (*x1a*, *x2*)›
‹*y* = ((*x1a*, *x2*), *x2a*)›
‹*x1b* = (*x1c*, *x2b*)›
‹*x* = ((*x1c*, *x2b*), *x2c*)›
‹*x′* = (*x1d*, *x2d*)›
‹*xa* = (*x1d*, *x2e*)›
‹*x1f* = *x1g*›
‹*x1e* = *x1d*›
‹*x2f0* = (*x2f1*, *x2f*, *x2f3*)›
‹*x2g* = *x2f*›
‹*x2g0* = (*x2g*, *x2g2*)› **and**
st′: ‹*x2* ! *x1d* = (*x1g*, *x2f0*)› **and**
cach:‹(*x2e*, *x2d*) ∈ *cach-refinement* $\mathcal{A}_{in}$› **and**
‹(*x2c*, *x2a*) ∈ *cach-refinement* $\mathcal{A}_{in}$› **and**
x2f0-x2g0: ‹((*x1g*, *x2g*, *x2g2*), (*x1f*, *x2f1*, *x2f*, *x2f3*)) ∈ *ana-lookup-rel NU*›
using *xy st xax′ param-nth*[*of x1e x2 x1d x2b* ‹*ana-lookup-rel NU*›] *le*
by (*auto intro*: *simp*: *ana-lookup-rel-alt-def*)

```
  have arena: ‹valid-arena x1c x1a vdom›
    using xy unfolding st by auto
  have ‹x2 ! x1e ∈ set x2›
    using le
    by auto
  then have ‹x2 ! x1d ∈ set x2› and
    x2f: ‹x2f ≤ length (x1a ∝ x1f)› and
    x1f: ‹x1g ∈# dom-m x1a› and
    x2g: ‹x2g > 0› and
    x2g-u1-le: ‹x2g − 1 < length (x1a ∝ x1f)›
    using inv le x2f0-x2g0 nth-mem[of x1d x2]
    unfolding mark-failed-lits-stack-inv-def x′ prod.case st st′
    by (auto simp del: nth-mem simp: st′ ana-lookup-rel-alt-def split: if-splits
      dest!: bspec[of ‹set x2› - ‹(-, -, -, -)›])

  have ‹is-Lit (x1c ! (x1g + (x2g − 1)))›
    by (rule arena-lifting[OF arena x1f]) (use x2f x2g x2g-u1-le in auto)
  then show ?le and ?A
    using arena-lifting[OF arena x1f] le x2f x1f x2g atm x2g-u1-le
    by (auto simp: arena-lit-def)
  show ?lit
    unfolding arena-lit-pre-def arena-is-valid-clause-idx-and-access-def
    by (rule bex-leI[of - x1f])
      (use arena x1f x2f x2g x2g-u1-le in ‹auto intro!: exI[of - x1a] exI[of - vdom]›)
  show ‹x1g + x2g ≥ 1›
    using x2g by auto
  have [simp]: ‹arena-lit x1c (x1g + x2g − Suc 0) = x1a ∝ x1g ! (x2g − Suc 0)›
     using that x1f x2f x2g x2g-u1-le by (auto simp: arena-lifting[OF arena])
  have ‹atm-of (arena-lit x1c (x1g + x2g − Suc 0)) < length (fst x2e)›
    using ‹?A› cach by (auto simp: cach-refinement-alt-def dest: multi-member-split)

  then show ?final
    using ‹?le› ‹?A› cach x1f x2g-u1-le x2g assms
   apply −
   apply (rule conflict-min-cach-set-failed[of 𝒜_in, THEN fref-to-Down-curry, THEN order-trans])
    by (cases x2e)
      (auto simp:  cach-refinement-alt-def RETURN-def conc-fun-RES
      arena-lifting[OF arena] subset-mset-add-new)
qed

show ?thesis
  unfolding isa-mark-failed-lits-stack-def mark-failed-lits-stack-def uncurry-def
  apply (rewrite at ‹let - = length - in -› Let-def)
  apply (intro frefI nres-relI)
  apply refine-vcg
  subgoal by (auto simp: list-rel-imp-same-length)
  subgoal by auto
  subgoal by auto
  subgoal for x y x1 x1a x2 x2a x1b x1c x2b x2c xa x′ x1d x2d x1e x2e
    by (auto simp: list-rel-imp-same-length)
  subgoal by auto
  subgoal by (rule ge1)
  subgoal by (rule le-length-arena)
  subgoal
    by (rule is-lit)
```

subgoal
by (*rule final*)
subgoal by *auto*
done
qed

definition *isa-get-literal-and-remove-of-analyse-wl*
:: ‹*arena* ⇒ (*nat* × *nat* × *bool*) *list* ⇒ *nat literal* × (*nat* × *nat* × *bool*) *list*› **where**
‹*isa-get-literal-and-remove-of-analyse-wl C analyse* =
(*let* (*i*, *j*, *b*) = (*last analyse*) *in*
(*arena-lit C* (*i* + *j*), *analyse*[*length analyse* − *1* := (*i*, *j* + *1*, *b*)]))›

definition *isa-get-literal-and-remove-of-analyse-wl-pre*
:: ‹*arena* ⇒ (*nat* × *nat* × *bool*) *list* ⇒ *bool*› **where**
‹*isa-get-literal-and-remove-of-analyse-wl-pre arena analyse* ⟷
(*let* (*i*, *j*, *b*) = *last analyse in*
analyse ≠ [] ∧ *arena-lit-pre arena* (*i*+*j*) ∧ *j* < *unat32-max*)›

lemma *arena-lit-pre-le*: ‹*length a* ≤ *unat64-max* ⟹
arena-lit-pre a i ⟹ *i* ≤ *unat64-max*›
using *arena-lifting*(*7*)[*of a* - -] **unfolding** *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
by *fastforce*

lemma *arena-lit-pre-le2*: ‹*length a* ≤ *unat64-max* ⟹
arena-lit-pre a i ⟹ *i* < *unat64-max*›
using *arena-lifting*(*7*)[*of a* - -] **unfolding** *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
by *fastforce*

definition *lit-redundant-reason-stack-wl-lookup-pre* :: ‹*nat literal* ⇒ *arena-el list* ⇒ *nat* ⇒ *bool*› **where**
‹*lit-redundant-reason-stack-wl-lookup-pre L NU C* ⟷
arena-lit-pre NU C ∧
arena-is-valid-clause-idx NU C›

definition *lit-redundant-reason-stack-wl-lookup*
:: ‹*nat literal* ⇒ *arena-el list* ⇒ *nat* ⇒ *nat* × *nat* × *bool*›
where
‹*lit-redundant-reason-stack-wl-lookup L NU C* =
(*if arena-length NU C* > *2 then* (*C*, *1*, *False*)
else if arena-lit NU C = *L*
then (*C*, *1*, *False*)
else (*C*, *0*, *True*))›

definition *ana-lookup-conv-lookup* :: ‹*arena* ⇒ (*nat* × *nat* × *bool*) ⇒ (*nat* × *nat* × *nat* × *nat*)› **where**
‹*ana-lookup-conv-lookup NU* = (λ(*C*, *i*, *b*).
(*C*, (*if b then 1 else 0*), *i*, (*if b then 1 else arena-length NU C*)))›

definition *ana-lookup-conv-lookup-pre* :: ‹*arena* ⇒ (*nat* × *nat* × *bool*) ⇒ *bool*› **where**
‹*ana-lookup-conv-lookup-pre NU* = (λ(*C*, *i*, *b*). *arena-is-valid-clause-idx NU C*)›

definition *isa-lit-redundant-rec-wl-lookup*
:: ‹*trail-pol* ⇒ *arena* ⇒ *lookup-clause-rel* ⇒
- ⇒ - ⇒ - ⇒ (- × - × *bool*) *nres*›
where
‹*isa-lit-redundant-rec-wl-lookup M NU D cach analysis lbd* =
$WHILE_T^{\lambda\text{-}.\ True}$

```
    (λ(cach, analyse, b). analyse ≠ [])
    (λ(cach, analyse, b). do {
        ASSERT(analyse ≠ []);
        ASSERT(length analyse ≤ 1 +  unat32-max div 2);
        ASSERT(arena-is-valid-clause-idx NU (fst (last analyse)));
  ASSERT(ana-lookup-conv-lookup-pre NU ((last analyse)));
  let (C, k, i, len) = ana-lookup-conv-lookup NU ((last analyse));
        ASSERT(C < length NU);
        ASSERT(arena-is-valid-clause-idx NU C);
        ASSERT(arena-lit-pre NU (C + k));
        if i ≥ len
        then do {
    cach ← conflict-min-cach-set-removable-l cach (atm-of (arena-lit NU (C + k)));
          RETURN(cach, butlast analyse, True)
  }
        else do {
    ASSERT (isa-get-literal-and-remove-of-analyse-wl-pre NU analyse);
    let (L, analyse) = isa-get-literal-and-remove-of-analyse-wl NU analyse;
          ASSERT(length analyse ≤ 1 +  unat32-max div 2);
    ASSERT(get-level-pol-pre (M, L));
    let b = ¬level-in-lbd (get-level-pol M L) lbd;
    ASSERT(atm-in-conflict-lookup-pre (atm-of L) D);
    ASSERT(conflict-min-cach-l-pre (cach, atm-of L));
    if (get-level-pol M L = 0 ∨
 conflict-min-cach-l cach (atm-of L) = SEEN-REMOVABLE ∨
 atm-in-conflict-lookup (atm-of L) D)
    then RETURN (cach, analyse, False)
    else if b ∨ conflict-min-cach-l cach (atm-of L) = SEEN-FAILED
    then do {
  cach ← isa-mark-failed-lits-stack NU analyse cach;
  RETURN (cach, take 0 analyse, False)
    }
    else do {
  C ← get-propagation-reason-pol M (−L);
  case C of
    Some C ⇒ do {
    ASSERT(lit-redundant-reason-stack-wl-lookup-pre (−L) NU C);
    RETURN (cach, analyse @ [lit-redundant-reason-stack-wl-lookup (−L) NU C], False)
    }
  | None ⇒ do {
    cach ← isa-mark-failed-lits-stack NU analyse cach;
    RETURN (cach, take 0 analyse, False)
     }
       }
      }
     })
    (cach, analysis, False)›

lemma isa-lit-redundant-rec-wl-lookup-alt-def:
  ‹isa-lit-redundant-rec-wl-lookup M NU D cach analysis lbd =
   WHILE_T^λ-. True
    (λ(cach, analyse, b). analyse ≠ [])
    (λ(cach, analyse, b). do {
        ASSERT(analyse ≠ []);
        ASSERT(length analyse ≤ 1 +  unat32-max div 2);
  let (C, i, b) = last analyse;
```

```
        ASSERT(arena-is-valid-clause-idx NU (fst (last analyse)));
    ASSERT(ana-lookup-conv-lookup-pre NU (last analyse));
    let (C, k, i, len) = ana-lookup-conv-lookup NU ((C, i, b));
        ASSERT(C < length NU);
        let - = map xarena-lit
            ((Misc.slice
              C
              (C + arena-length NU C))
              NU);
        ASSERT(arena-is-valid-clause-idx NU C);
        ASSERT(arena-lit-pre NU (C + k));
        if i ≥ len
        then do {
      cach ← conflict-min-cach-set-removable-l cach (atm-of (arena-lit NU (C + k)));
          RETURN(cach, butlast analyse, True)
        }
        else do {
            ASSERT (isa-get-literal-and-remove-of-analyse-wl-pre NU analyse);
            let (L, analyse) = isa-get-literal-and-remove-of-analyse-wl NU analyse;
            ASSERT(length analyse ≤ 1+ unat32-max div 2);
            ASSERT(get-level-pol-pre (M, L));
            let b = ¬level-in-lbd (get-level-pol M L) lbd;
            ASSERT(atm-in-conflict-lookup-pre (atm-of L) D);
       ASSERT(conflict-min-cach-l-pre (cach, atm-of L));
            if (get-level-pol M L = 0 ∨
                conflict-min-cach-l cach (atm-of L) = SEEN-REMOVABLE ∨
                atm-in-conflict-lookup (atm-of L) D)
            then RETURN (cach, analyse, False)
            else if b ∨ conflict-min-cach-l cach (atm-of L) = SEEN-FAILED
            then do {
              cach ← isa-mark-failed-lits-stack NU analyse cach;
              RETURN (cach, [], False)
            }
            else do {
              C ← get-propagation-reason-pol M (−L);
              case C of
                Some C ⇒ do {
      ASSERT(lit-redundant-reason-stack-wl-lookup-pre (−L) NU C);
      RETURN (cach, analyse @ [lit-redundant-reason-stack-wl-lookup (−L) NU C], False)
     }
              | None ⇒ do {
                  cach ← isa-mark-failed-lits-stack NU analyse cach;
                  RETURN (cach, [], False)
              }
          }
        }
      })
      (cach, analysis, False)›
  unfolding isa-lit-redundant-rec-wl-lookup-def Let-def take-0
  by (auto simp: Let-def)

lemma lit-redundant-rec-wl-lookup-alt-def:
  ‹lit-redundant-rec-wl-lookup A M NU D cach analysis lbd =
      WHILE_T^lit-redundant-rec-wl-inv2 M NU D
        (λ(cach, analyse, b). analyse ≠ [])
        (λ(cach, analyse, b). do {
```

```
        ASSERT(analyse ≠ []);
        ASSERT(length analyse ≤ length M);
    let (C, k, i, len) = ana-lookup-conv NU (last analyse);
        ASSERT(C ∈# dom-m NU);
        ASSERT(length (NU ∝ C) > k); — >= 2 would work too
        ASSERT (NU ∝ C ! k ∈ lits-of-l M);
        ASSERT(NU ∝ C ! k ∈# L_all A);
    ASSERT(literals-are-in-L_in A (mset (NU ∝ C)));
    ASSERT(length (NU ∝ C) ≤ Suc (unat32-max div 2));
    ASSERT(len ≤ length (NU ∝ C)); — makes the refinement easier
    let (C,k, i, len) = (C,k,i,len);
        let C = NU ∝ C;
        if i ≥ len
        then
          RETURN(cach (atm-of (C ! k) := SEEN-REMOVABLE), butlast analyse, True)
        else do {
          let (L, analyse) = get-literal-and-remove-of-analyse-wl2 C analyse;
          ASSERT(L ∈# L_all A);
          let b = ¬level-in-lbd (get-level M L) lbd;
          if (get-level M L = 0 ∨
              conflict-min-cach cach (atm-of L) = SEEN-REMOVABLE ∨
              atm-in-conflict (atm-of L) D)
          then RETURN (cach, analyse, False)
          else if b ∨ conflict-min-cach cach (atm-of L) = SEEN-FAILED
          then do {
            ASSERT(mark-failed-lits-stack-inv2 NU analyse cach);
            cach ← mark-failed-lits-wl NU analyse cach;
            RETURN (cach, [], False)
          }
          else do {
       ASSERT(- L ∈ lits-of-l M);
            C ← get-propagation-reason M (-L);
            case C of
              Some C ⇒ do {
    ASSERT(C ∈# dom-m NU);
    ASSERT(length (NU ∝ C) ≥ 2);
    ASSERT(literals-are-in-L_in A (mset (NU ∝ C)));
                ASSERT(length (NU ∝ C) ≤ Suc (unat32-max div 2));
    RETURN (cach, analyse @ [lit-redundant-reason-stack2 (-L) NU C], False)
  }
            | None ⇒ do {
                ASSERT(mark-failed-lits-stack-inv2 NU analyse cach);
                cach ← mark-failed-lits-wl NU analyse cach;
                RETURN (cach, [], False)
              }
          }
        }
      })
     (cach, analysis, False)›
  unfolding lit-redundant-rec-wl-lookup-def Let-def by auto

lemma valid-arena-nempty:
  ‹valid-arena arena N vdom ⟹ i ∈# dom-m N ⟹ N ∝ i ≠ []›
  using arena-lifting(19)[of arena N vdom i]
  arena-lifting(4)[of arena N vdom i]
  by auto
```

lemma *isa-lit-redundant-rec-wl-lookup-lit-redundant-rec-wl-lookup*:
assumes ‹*isasat-input-bounded* $\mathcal{A}$›
shows ‹(*uncurry5 isa-lit-redundant-rec-wl-lookup*, *uncurry5* (*lit-redundant-rec-wl-lookup* $\mathcal{A}$)) ∈
[λ(((((-, *N*), -), -), -), -). *literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ ((*mset* ∘ *fst*) ‘# *ran-m N*)]$_f$
trail-pol $\mathcal{A}$ ×$_f$ {(*arena*, *N*). *valid-arena arena N vdom*} ×$_f$ *lookup-clause-rel* $\mathcal{A}$ ×$_f$
cach-refinement $\mathcal{A}$ ×$_f$ *Id* ×$_f$ *Id* →
⟨*cach-refinement* $\mathcal{A}$ ×$_r$ *Id* ×$_r$ *bool-rel*⟩*nres-rel*›
proof −
have *isa-mark-failed-lits-stack*: ‹*isa-mark-failed-lits-stack x2e x2z x1l*
≤ ⇓ (*cach-refinement* $\mathcal{A}$)
(*mark-failed-lits-wl x2 x2y x1j*)›
if
‹*case y of*
(*x*, *xa*) ⇒
(*case x of*
(*x*, *xa*) ⇒
(*case x of*
(*x*, *xa*) ⇒
(*case x of*
(*x*, *xa*) ⇒
(*case x of*
(*uu*-, *N*) ⇒
λ- - - -.
literals-are-in-$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ ((*mset* ∘ *fst*) ‘# *ran-m N*)) *xa*)
xa)
xa)
xa› **and**
‹(*x*, *y*)
∈ *trail-pol* $\mathcal{A}$ ×$_f$ {(*arena*, *N*). *valid-arena arena N vdom*} ×$_f$
lookup-clause-rel $\mathcal{A}$ ×$_f$ *cach-refinement* $\mathcal{A}$ ×$_f$ *Id* ×$_f$ *Id*› **and**
‹*x1c* = (*x1d*, *x2*)› **and**
‹*x1b* = (*x1c*, *x2a*)› **and**
‹*x1a* = (*x1b*, *x2b*)› **and**
‹*x1* = (*x1a*, *x2c*)› **and**
‹*y* = (*x1*, *x2d*)› **and**
‹*x1h* = (*x1i*, *x2e*)› **and**
‹*x1g* = (*x1h*, *x2f*)› **and**
‹*x1f* = (*x1g*, *x2g*)› **and**
‹*x1e* = (*x1f*, *x2h*)› **and**
‹*x* = (*x1e*, *x2i*)› **and**
‹(*xa*, *x′*) ∈ *cach-refinement* $\mathcal{A}$ ×$_f$ (*Id* ×$_f$ *bool-rel*)› **and**
‹*case xa of* (*cach*, *analyse*, *b*) ⇒ *analyse* ≠ []› **and**
‹*case x′ of* (*cach*, *analyse*, *b*) ⇒ *analyse* ≠ []› **and**
‹*lit-redundant-rec-wl-inv2 x1d x2 x2a x′*› **and**
‹*x2j* = (*x1k*, *x2k*)› **and**
‹*x′* = (*x1j*, *x2j*)› **and**
‹*x2l* = (*x1m*, *x2m*)› **and**
‹*xa* = (*x1l*, *x2l*)› **and**
‹*x1k* ≠ []› **and**
‹*x1m* ≠ []› **and**
‹*x2o* = (*x1p*, *x2p*)› **and**
‹*x2n* = (*x1o*, *x2o*)› **and**
‹*ana-lookup-conv x2* (*last x1k*) = (*x1n*, *x2n*)› **and**
‹*x2q* = (*x1r*, *x2r*)› **and**
‹*last x1m* = (*x1q*, *x2q*)› **and**

‹x1n ∈# dom-m x2› **and**
‹x1o < length (x2 ∝ x1n)› **and**
‹x2 ∝ x1n ! x1o ∈ lits-of-l x1d› **and**
‹x2 ∝ x1n ! x1o ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
‹literals-are-in-$\mathcal{L}_{in}$ $\mathcal{A}$ (mset (x2 ∝ x1n))› **and**
‹length (x2 ∝ x1n) ≤ Suc (unat32-max div 2)› **and**
‹x2p ≤ length (x2 ∝ x1n)› **and**
‹arena-is-valid-clause-idx x2e (fst (last x1m))› **and**
‹x2t = (x1u, x2u)› **and**
‹x2s = (x1t, x2t)› **and**
‹(x1n, x1o, x1p, x2p) = (x1s, x2s)› **and**
‹x2w = (x1x, x2x)› **and**
‹x2v = (x1w, x2w)› **and**
‹ana-lookup-conv-lookup x2e (x1q, x1r, x2r) = (x1v, x2v)› **and**
‹x1v < length x2e› **and**
‹arena-is-valid-clause-idx x2e x1v› **and**
‹arena-lit-pre x2e (x1v + x1w)› **and**
‹¬ x2x ≤ x1x› **and**
‹¬ x2u ≤ x1u› **and**
‹isa-get-literal-and-remove-of-analyse-wl-pre x2e x1m› **and**
‹get-literal-and-remove-of-analyse-wl2 (x2 ∝ x1s) x1k = (x1y, x2y)› **and**
‹isa-get-literal-and-remove-of-analyse-wl x2e x1m = (x1z, x2z)› **and**
‹x1y ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and** ‹get-level-pol-pre (x1i, x1z)› **and**
‹atm-in-conflict-lookup-pre (atm-of x1z) x2f› **and**
‹conflict-min-cach-l-pre (x1l, atm-of x1z)› **and**
‹¬ (get-level-pol x1i x1z = 0 ∨
conflict-min-cach-l x1l (atm-of x1z) = SEEN-REMOVABLE ∨
atm-in-conflict-lookup (atm-of x1z) x2f)› **and**
‹¬ (get-level x1d x1y = 0 ∨
conflict-min-cach x1j (atm-of x1y) = SEEN-REMOVABLE ∨
atm-in-conflict (atm-of x1y) x2a)› **and**
‹¬ level-in-lbd (get-level-pol x1i x1z) x2i ∨
conflict-min-cach-l x1l (atm-of x1z) = SEEN-FAILED› **and**
‹¬ level-in-lbd (get-level x1d x1y) x2d ∨
conflict-min-cach x1j (atm-of x1y) = SEEN-FAILED› **and**
inv2: ‹mark-failed-lits-stack-inv2 x2 x2y x1j› **and**
‹length x1m ≤ 1+ unat32-max div 2›
for x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g
x2h x2i xa x' x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q
x2q x1r x2r x1s x2s x1t x2t x1u x2u x1v x2v x1w x2w x1x x2x x1y x2y x1z
x2z
proof −
have [simp]: ‹x2z = x2y›
using that
by (auto simp: isa-get-literal-and-remove-of-analyse-wl-def
get-literal-and-remove-of-analyse-wl2-def)

obtain x2y0 **where**
x2z: ‹(x2y, x2y0) ∈ ana-lookups-rel x2› **and**
inv: ‹mark-failed-lits-stack-inv x2 x2y0 x1j›
using inv2 **unfolding** mark-failed-lits-stack-inv2-def
by blast
have 1: ‹mark-failed-lits-wl x2 x2y x1j = mark-failed-lits-wl x2 x2y0 x1j›
unfolding mark-failed-lits-wl-def **by** auto
show ?thesis
unfolding 1

apply (*rule isa-mark-failed-lits-stack-isa-mark-failed-lits-stack*[*THEN*
fref-to-Down-curry2, *of* $\mathcal{A}$ *x2 x2y0 x1j x2e x2z x1l vdom x2*, *THEN order-trans*])
subgoal using *assms* **by** *fast*
subgoal using *that x2z* **by** (*auto simp*: *list-rel-imp-same-length*[*symmetric*]
isa-get-literal-and-remove-of-analyse-wl-def
get-literal-and-remove-of-analyse-wl2-def)
subgoal using *that x2z inv* **by** *auto*
apply (*rule order-trans*)
apply (*rule ref-two-step'*)
apply (*rule mark-failed-lits-stack-mark-failed-lits-wl*[*THEN*
fref-to-Down-curry2, *of* $\mathcal{A}$ *x2 x2y0 x1j*])
subgoal using *inv x2z that* **by** *auto*
subgoal using *that* **by** *auto*
subgoal by *auto*
done
qed
have *isa-mark-failed-lits-stack2*: ‹*isa-mark-failed-lits-stack x2e x2z x1l*
≤ ⇓ (*cach-refinement* $\mathcal{A}$) (*mark-failed-lits-wl x2 x2y x1j*)›
if
‹*case y of*
(*x*, *xa*) ⇒
(*case x of*
(*x*, *xa*) ⇒
(*case x of*
(*x*, *xa*) ⇒
(*case x of*
(*x*, *xa*) ⇒
(*case x of*
(*uu-*, *N*) ⇒
λ- - - -.
literals-are-in-$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ ((*mset* ∘ *fst*) ‘# *ran-m N*)) *xa*)
xa)
xa)
xa› **and**
‹(*x*, *y*)
∈ *trail-pol* $\mathcal{A}$ ×$_f$ {(*arena*, *N*). *valid-arena arena N vdom*} ×$_f$ *lookup-clause-rel* $\mathcal{A}$ ×$_f$
cach-refinement $\mathcal{A}$ ×$_f$ *Id* ×$_f$
Id› **and**
‹*ana-lookup-conv-lookup x2e* (*x1q*, *x1r*, *x2r*) = (*x1v*, *x2v*)› **and**
‹*x1v* < *length x2e*› **and**
‹*arena-is-valid-clause-idx x2e x1v*› **and**
‹*arena-lit-pre x2e* (*x1v* + *x1w*)› **and**
‹¬ *x2x* ≤ *x1x*› **and**
‹¬ *x2u* ≤ *x1u*› **and**
‹*isa-get-literal-and-remove-of-analyse-wl-pre x2e x1m*› **and**
‹*get-literal-and-remove-of-analyse-wl2* (*x2* ∝ *x1s*) *x1k* = (*x1y*, *x2y*)› **and**
‹*isa-get-literal-and-remove-of-analyse-wl x2e x1m* = (*x1z*, *x2z*)› **and**
‹*x1y* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and** ‹*get-level-pol-pre* (*x1i*, *x1z*)› **and**
‹*atm-in-conflict-lookup-pre* (*atm-of x1z*) *x2f*› **and**
‹*conflict-min-cach-l-pre* (*x1l*, *atm-of x1z*)› **and**
‹¬ (*get-level-pol x1i x1z* = *0* ∨
conflict-min-cach-l x1l (*atm-of x1z*) = *SEEN-REMOVABLE* ∨
atm-in-conflict-lookup (*atm-of x1z*) *x2f*)› **and**
‹¬ (*get-level x1d x1y* = *0* ∨
conflict-min-cach x1j (*atm-of x1y*) = *SEEN-REMOVABLE* ∨
atm-in-conflict (*atm-of x1y*) *x2a*)› **and**

‹¬ (¬ level-in-lbd (get-level-pol x1i x1z) x2i ∨
conflict-min-cach-l x1l (atm-of x1z) = SEEN-FAILED)› **and**
‹¬ (¬ level-in-lbd (get-level x1d x1y) x2d ∨
conflict-min-cach x1j (atm-of x1y) = SEEN-FAILED)› **and**
‹− x1y ∈ lits-of-l x1d› **and**
‹(xb, x′a) ∈ ⟨nat-rel⟩option-rel› **and**
‹xb = None› **and**
‹x′a = None› **and**
inv2: ‹mark-failed-lits-stack-inv2 x2 x2y x1j› **and**
‹(xa, x′) ∈ cach-refinement 𝒜 ×$_f$ (Id ×$_f$ bool-rel)› **and** ‹case xa of (cach, analyse, b) ⇒ analyse ≠ []› **and**
‹case x′ of (cach, analyse, b) ⇒ analyse ≠ []› **and**
‹lit-redundant-rec-wl-inv2 x1d x2 x2a x′› **and**
‹x2j = (x1k, x2k)› **and**
‹x′ = (x1j, x2j)› **and**
‹x2l = (x1m, x2m)› **and**
‹xa = (x1l, x2l)› **and**
‹x1k ≠ []› **and**
‹x1m ≠ []› **and**
‹x2o = (x1p, x2p)› **and**
‹x2n = (x1o, x2o)› **and**
‹ana-lookup-conv x2 (last x1k) = (x1n, x2n)› **and**
‹x2q = (x1r, x2r)› **and**
‹last x1m = (x1q, x2q)› **and**
‹x1n ∈# dom-m x2› **and**
‹x1o < length (x2 ∝ x1n)› **and**
‹x2 ∝ x1n ! x1o ∈ lits-of-l x1d› **and**
‹x2 ∝ x1n ! x1o ∈# $\mathcal{L}_{all}$ 𝒜› **and**
‹literals-are-in-$\mathcal{L}_{in}$ 𝒜 (mset (x2 ∝ x1n))› **and**
‹length (x2 ∝ x1n) ≤ Suc (unat32-max div 2)› **and**
‹x2p ≤ length (x2 ∝ x1n)› **and**
‹arena-is-valid-clause-idx x2e (fst (last x1m))› **and**
‹x2t = (x1u, x2u)› **and**
‹x2s = (x1t, x2t)› **and**
‹(x1n, x1o, x1p, x2p) = (x1s, x2s)› **and**
‹x2w = (x1x, x2x)› **and**
‹x2v = (x1w, x2w)› **and**
‹x1c = (x1d, x2)› **and**
‹x1b = (x1c, x2a)› **and**
‹x1a = (x1b, x2b)› **and**
‹x1 = (x1a, x2c)› **and**
‹y = (x1, x2d)› **and**
‹x1h = (x1i, x2e)› **and**
‹x1g = (x1h, x2f)› **and**
‹x1f = (x1g, x2g)› **and**
‹x1e = (x1f, x2h)› **and**
‹x = (x1e, x2i)› **and**
‹length x1m ≤ 1 + unat32-max div 2›
for x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g
x2h x2i xa x′ x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q
x2q x1r x2r x1s x2s x1t x2t x1u x2u x1v x2v x1w x2w x1x x2x x1y x2y x1z
x2z xb x′a
proof −
have [simp]: ‹x2z = x2y›
using that
by (auto simp: isa-get-literal-and-remove-of-analyse-wl-def

get-literal-and-remove-of-analyse-wl2-def)

obtain *x2y0* **where**
x2z: ‹(*x2y*, *x2y0*) ∈ *ana-lookups-rel x2*› **and**
inv: ‹*mark-failed-lits-stack-inv x2 x2y0 x1j*›
using *inv2* **unfolding** *mark-failed-lits-stack-inv2-def*
by *blast*
have *1*: ‹*mark-failed-lits-wl x2 x2y x1j* = *mark-failed-lits-wl x2 x2y0 x1j*›
unfolding *mark-failed-lits-wl-def* **by** *auto*
show *?thesis*
unfolding *1*
apply (*rule isa-mark-failed-lits-stack-isa-mark-failed-lits-stack*[*THEN*
fref-to-Down-curry2, *of* $\mathcal{A}$ *x2 x2y0 x1j x2e x2z x1l vdom x2*, *THEN order-trans*])
subgoal using *assms* **by** *fast*
subgoal using *that x2z* **by** (*auto simp*: *list-rel-imp-same-length*[*symmetric*]
isa-get-literal-and-remove-of-analyse-wl-def
get-literal-and-remove-of-analyse-wl2-def)
subgoal using *that x2z inv* **by** *auto*
apply (*rule order-trans*)
apply (*rule ref-two-step'*)
apply (*rule mark-failed-lits-stack-mark-failed-lits-wl*[*THEN*
fref-to-Down-curry2, *of* $\mathcal{A}$ *x2 x2y0 x1j*])
subgoal using *inv x2z that* **by** *auto*
subgoal using *that* **by** *auto*
subgoal by *auto*
done
qed
have [*refine0*]: ‹*get-propagation-reason-pol M' L'*
≤ ⇓ (⟨*Id*⟩*option-rel*)
(*get-propagation-reason M L*)›
if ‹(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$› **and** ‹(*L'*, *L*) ∈ *Id*› **and** ‹*L* ∈ *lits-of-l M*›
for *M M' L L'*
using *get-propagation-reason-pol*[*of* $\mathcal{A}$, *THEN fref-to-Down-curry*, *of M L M' L'*] *that* **by** *auto*
note [*simp*]=*get-literal-and-remove-of-analyse-wl-def isa-get-literal-and-remove-of-analyse-wl-def*
arena-lifting **and** [*split*] = *prod.splits*

show *?thesis*
supply [[*goals-limit=1*]] *ana-lookup-conv-def*[*simp*] *ana-lookup-conv-lookup-def*[*simp*]
supply *RETURN-as-SPEC-refine*[*refine2 add*]
unfolding *isa-lit-redundant-rec-wl-lookup-alt-def lit-redundant-rec-wl-lookup-alt-def uncurry-def*
apply (*intro frefI nres-relI*)
apply (*refine-rcg*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g*
x2h x2i xa x' x1j x2j x1k x2k x1l x2l x1m x2m
by (*auto simp*: *arena-lifting*)
subgoal by (*auto simp*: *trail-pol-alt-def*)
subgoal by (*auto simp*: *arena-is-valid-clause-idx-def*
lit-redundant-rec-wl-inv2-def)
subgoal by (*auto simp*: *ana-lookup-conv-lookup-pre-def*)
subgoal by (*auto simp*: *arena-is-valid-clause-idx-def*)
subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g*
x2h x2i xa x' x1j x2j x1k x2k x1l x2l x1m x2m
by (*auto simp*: *arena-lifting arena-is-valid-clause-idx-def*)

subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g*
x2h x2i xa x′ x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q
x2q x1r x2r x1s x2s x1t x2t x1u x2u x1v x2v x1w x2w x1x x2x
apply (*auto simp*: *arena-is-valid-clause-idx-def lit-redundant-rec-wl-inv-def*
isa-get-literal-and-remove-of-analyse-wl-pre-def arena-lit-pre-def
arena-is-valid-clause-idx-and-access-def lit-redundant-rec-wl-ref-def)
by (*rule-tac x* = ‹*x1s*› **in** *exI*; *auto simp*: *valid-arena-nempty*)+
subgoal by (*auto simp*: *arena-lifting arena-is-valid-clause-idx-def*
lit-redundant-rec-wl-inv-def split: *if-splits*)
subgoal using *assms*
by (*auto simp*: *arena-lifting arena-is-valid-clause-idx-def bind-rule-complete-RES conc-fun-RETURN*
in-$\mathcal{L}_{all}$-atm-of-$\mathcal{A}_{in}$ lit-redundant-rec-wl-inv-def lit-redundant-rec-wl-ref-def
intro!: *conflict-min-cach-set-removable*[*of* $\mathcal{A}$,*THEN fref-to-Down-curry*, *THEN order-trans*]
dest: *List.last-in-set*)

subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g*
x2h x2i xa x′ x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q
x2q x1r x2r x1s x2s x1t x2t x1u x2u x1v x2v x1w x2w x1x x2x
by (*auto simp*: *arena-is-valid-clause-idx-def lit-redundant-rec-wl-inv-def*
isa-get-literal-and-remove-of-analyse-wl-pre-def arena-lit-pre-def
unat32-max-def
arena-is-valid-clause-idx-and-access-def lit-redundant-rec-wl-ref-def)
(*rule-tac x* = *x1s* **in** *exI*; *auto simp*: *unat32-max-def*; *fail*)+
subgoal by (*auto simp*: *list-rel-imp-same-length*)
subgoal by (*auto intro*!: *get-level-pol-pre*
simp: *get-literal-and-remove-of-analyse-wl2-def*)
subgoal by (*auto intro*!: *atm-in-conflict-lookup-pre*
simp: *get-literal-and-remove-of-analyse-wl2-def*)
subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g*
x2h x2i xa x′ x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o
by (*auto intro*!: *conflict-min-cach-l-pre*
simp: *get-literal-and-remove-of-analyse-wl2-def*)
subgoal
by (*auto simp*: *atm-in-conflict-lookup-atm-in-conflict*[*THEN fref-to-Down-unRET-uncurry-Id*]
nth-conflict-min-cach[*THEN fref-to-Down-unRET-uncurry-Id*] *in-$\mathcal{L}_{all}$-atm-of-$\mathcal{A}_{in}$*
get-level-get-level-pol atms-of-def
get-literal-and-remove-of-analyse-wl2-def
split: *prod.splits*)
(*subst* (*asm*) *atm-in-conflict-lookup-atm-in-conflict*[*THEN fref-to-Down-unRET-uncurry-Id*];
auto simp: *in-$\mathcal{L}_{all}$-atm-of-$\mathcal{A}_{in}$ atms-of-def*; *fail*)+
subgoal by (*auto simp*: *get-literal-and-remove-of-analyse-wl2-def*
split: *prod.splits*)
subgoal by (*auto simp*: *atm-in-conflict-lookup-atm-in-conflict*[*THEN fref-to-Down-unRET-uncurry-Id*]
nth-conflict-min-cach[*THEN fref-to-Down-unRET-uncurry-Id*] *in-$\mathcal{L}_{all}$-atm-of-$\mathcal{A}_{in}$*
get-level-get-level-pol atms-of-def
simp: *get-literal-and-remove-of-analyse-wl2-def*
split: *prod.splits*)
apply (*rule isa-mark-failed-lits-stack*; *assumption*)
subgoal by (*auto simp*: *split*: *prod.splits*)
subgoal by (*auto simp*: *split*: *prod.splits*)
subgoal by (*auto simp*: *get-literal-and-remove-of-analyse-wl2-def*
split: *prod.splits*)
apply *assumption*
apply (*rule isa-mark-failed-lits-stack2*; *assumption*)
subgoal by *auto*
subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g*

x2h x2i xa x′ x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q
x2q x1r x2r x1s x2s x1t x2t x1u x2u x1v x2v x1w x2w x1x x2x x1y x2y x1z
x2z xb x′a xc x′b
unfolding *lit-redundant-reason-stack-wl-lookup-pre-def*
by (*auto simp*: *lit-redundant-reason-stack-wl-lookup-pre-def arena-lit-pre-def*
arena-is-valid-clause-idx-and-access-def arena-is-valid-clause-idx-def
simp: *valid-arena-nempty get-literal-and-remove-of-analyse-wl2-def*
lit-redundant-reason-stack-wl-lookup-def
lit-redundant-reason-stack2-def
intro!: *exI*[*of* - *x′b*] *bex-leI*[*of* - *x′b*])
subgoal premises *p* **for** *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g*
x2h x2i xa x′ x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q
x2q x1r x2r x1s x2s x1t x2t x1u x2u xb x′a xc x′b
using *p*
by (*auto simp add*: *lit-redundant-reason-stack-wl-lookup-def*
lit-redundant-reason-stack-def lit-redundant-reason-stack-wl-lookup-pre-def
lit-redundant-reason-stack2-def get-literal-and-remove-of-analyse-wl2-def
arena-lifting[*of x2e x2 vdom*]) — I have no idea why ⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *header-size* (*?N* ∝ *?i*) ≤ *?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *?i* < *length ?arena*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *is-Size* (*?arena* ! (*?i* − *SIZE-SHIFT*))
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *length* (*?N* ∝ *?i*) = *arena-length ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*; *?j* < *length* (*?N* ∝ *?i*)⟧ ⟹ *?N* ∝ *?i* ! *?j* = *arena-lit ?arena* (*?i* + *?j*)
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*; *?j* < *length* (*?N* ∝ *?i*)⟧ ⟹ *is-Lit* (*?arena* ! (*?i* + *?j*))
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*; *?j* < *length* (*?N* ∝ *?i*)⟧ ⟹ *?i* + *?j* < *length ?arena*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *?N* ∝ *?i* ! *0* = *arena-lit ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *is-Lit* (*?arena* ! *?i*)
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *?i* + *length* (*?N* ∝ *?i*) ≤ *length ?arena*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*; *is-long-clause* (*?N* ∝ *?i*)⟧ ⟹ *is-Pos* (*?arena* ! (*?i* − *POS-SHIFT*))
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*; *is-long-clause* (*?N* ∝ *?i*)⟧ ⟹ *arena-pos ?arena ?i* ≤ *arena-length ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *True*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *is-Status* (*?arena* ! (*?i* − *LBD-SHIFT*))
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *SIZE-SHIFT* ≤ *?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *LBD-SHIFT* ≤ *?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *True*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *2* ≤ *arena-length ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *Suc 0* ≤ *arena-length ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *0* ≤ *arena-length ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *Suc 0* < *arena-length ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *0* < *arena-length ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ (*arena-status ?arena ?i* = *LEARNED*) = (¬ *irred ?N ?i*)
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ (*arena-status ?arena ?i* = *IRRED*) = *irred ?N ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *arena-status ?arena ?i* ≠ *DELETED*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *Misc.slice ?i* (*?i* + *arena-length ?arena ?i*) *?arena* = *map ALit* (*?N* ∝ *?i*) requires to be instantiated.
done
qed

lemma *iterate-over-conflict-spec*:

```
  fixes D :: ‹'v clause›
  assumes ‹NU + NUE ⊨pm add-mset K D› and dist: ‹distinct-mset D›
  shows
    ‹iterate-over-conflict K M NU NUE D ≤ ⇓ Id (SPEC(λD'. D' ⊆# D ∧
       NU + NUE ⊨pm add-mset K D'))›
proof −
  define I' where
    ‹I' = (λ(E:: 'v clause, f :: 'v clause).
          E ⊆# D ∧ NU + NUE ⊨pm add-mset K E ∧ distinct-mset E ∧ distinct-mset f)›

  have init-I': ‹I' (D, D)›
    using ‹NU + NUE ⊨pm add-mset K D› dist unfolding I'-def highest-lit-def by auto

  have red: ‹is-literal-redundant-spec K NU NUE a x
     ≤ SPEC (λred. (if ¬ red then RETURN (a, remove1-mset x aa)
              else RETURN (remove1-mset x a, remove1-mset x aa))
            ≤ SPEC (λs'. iterate-over-conflict-inv M D s' ∧ I' s' ∧
              (s', s) ∈ measure (λ(D, D'). size D')))›
    if
      ‹iterate-over-conflict-inv M D s› and
      ‹I' s› and
      ‹case s of (D, D') ⇒ D' ≠ {#}› and
      ‹s = (a, aa)› and
      ‹x ∈# aa›
    for s a b aa x
  proof −
    have ‹x ∈# a› ‹distinct-mset aa›
      using that
      by (auto simp: I'-def highest-lit-def
          eq-commute[of ‹get-level - -›] iterate-over-conflict-inv-def
          get-maximum-level-add-mset add-mset-eq-add-mset
          dest!:  split: option.splits if-splits)
    then show ?thesis
      using that
      by (auto simp: is-literal-redundant-spec-def iterate-over-conflict-inv-def
          I'-def size-mset-remove1-mset-le-iff remove1-mset-add-mset-If
          intro: mset-le-subtract)
  qed

  show ?thesis
    unfolding iterate-over-conflict-def
    apply (refine-vcg WHILEIT-rule-stronger-inv[where
       R = ‹measure (λ(D :: 'v clause, D':: 'v clause).
            size D')› and
         I' = I'])
    subgoal by auto
    subgoal by (auto simp: iterate-over-conflict-inv-def highest-lit-def)
    subgoal by (rule init-I')
    subgoal by (rule red)
    subgoal unfolding I'-def iterate-over-conflict-inv-def by auto
    subgoal unfolding I'-def iterate-over-conflict-inv-def by auto
    done
qed

end
```

lemma
  fixes D :: ‹*nat clause*› **and** s **and** s' **and** NU :: ‹*nat clauses-l*› **and**
    S :: ‹*nat twl-st-wl*› **and** S' :: ‹*nat twl-st-l*› **and** S'' :: ‹*nat twl-st*›
  defines
    ‹$S''' \equiv state_W$*-of* S''›
  defines
    ‹$M \equiv$ *get-trail-wl* S› **and**
    NU: ‹$NU \equiv$ *get-clauses-wl* S› **and**
    NU'*-def*: ‹$NU' \equiv$ *mset* ‘# *ran-mf* NU› **and**
    NUE: ‹$NUE \equiv$ *get-unit-learned-clss-wl* S + *get-unit-init-clss-wl* S› **and**
    NUE: ‹$NUS \equiv$ *get-subsumed-learned-clauses-wl* S + *get-subsumed-init-clauses-wl* S› **and**
    $N0S$: ‹$N0S \equiv$ *get-learned-clauses0-wl* S + *get-init-clauses0-wl* S› **and**
    M': ‹$M' \equiv$ *trail* S'''›
  assumes
    S-S': ‹$(S, S') \in$ *state-wl-l None*› **and**
    S'-S'': ‹$(S', S'') \in$ *twl-st-l None*› **and**
    D'-D: ‹*mset* (*tl outl*) = D› **and**
    M-D: ‹$M \models as$ *CNot* D› **and**
    dist-D: ‹*distinct-mset* D› **and**
    tauto: ‹¬*tautology* D› **and**
    lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ M› **and**
    struct-invs: ‹*twl-struct-invs* S''› **and**
    add-inv: ‹*twl-list-invs* S'› **and**
    cach-init: ‹*conflict-min-analysis-inv* M' s' (NU' + NUE + NUS + $N0S$) D› **and**
    NU-P-D: ‹NU' + NUE + NUS + $N0S \models pm$ *add-mset* K D› **and**
    lits-D: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ D› **and**
    lits-NU: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf* NU)› **and**
    K: ‹K = *outl* ! 0› **and**
    outl-nempty: ‹*outl* ≠ []› **and**
    ‹*isasat-input-bounded* $\mathcal{A}$›
  shows
    ‹*minimize-and-extract-highest-lookup-conflict* $\mathcal{A}$ M NU D s' *lbd outl* ≤
      ⇓ ({((*E*, *s*, *outl*), *E'*). *E* = *E'* ∧ *mset* (*tl outl*) = *E* ∧ *outl*!0 = K ∧
          E' ⊆# D})
        (*SPEC* (λD'. $D' \subseteq\# D \wedge NU' + NUE + NUS + N0S \models pm$ *add-mset* K D'))›
proof −
  show *?thesis*
    apply (*rule order.trans*)
    apply (*rule minimize-and-extract-highest-lookup-conflict-iterate-over-conflict*[*OF*
        assms(9−24)[*unfolded assms*(1−10)],
        unfolded assms(1−10)[*symmetric*]])
    apply (*rule order.trans*)
    unfolding *add.assoc*
    apply (*rule ref-two-step'*[*OF iterate-over-conflict-spec*[*OF NU-P-D*[*unfolded add.assoc*] *dist-D*]])
    by (*auto simp*: *conc-fun-RES ac-simps*)
qed

lemma (**in** −) *lookup-conflict-upd-None-RETURN-def*:
  ‹*RETURN oo lookup-conflict-upd-None* = (λ(*n*, *xs*) *i*. *RETURN* (*n*− 1, *xs* [*i* := *NOTIN*]))›
  by (*auto intro*!: *ext*)

definition *isa-literal-redundant-wl-lookup* ::
    trail-pol ⇒ *arena* ⇒ *lookup-clause-rel* ⇒ *conflict-min-cach-l*
      ⇒ *nat literal* ⇒ *lbd* ⇒ (*conflict-min-cach-l* × (*nat* × *nat* × *bool*) *list* × *bool*) *nres*
where
  ‹*isa-literal-redundant-wl-lookup* M NU D *cach* L *lbd* = *do* {

```
    ASSERT(get-level-pol-pre (M, L));
    ASSERT(conflict-min-cach-l-pre (cach, atm-of L));
    if get-level-pol M L = 0 ∨ conflict-min-cach-l cach (atm-of L) = SEEN-REMOVABLE
    then RETURN (cach, [], True)
    else if conflict-min-cach-l cach (atm-of L) = SEEN-FAILED
    then RETURN (cach, [], False)
    else do {
      C ← get-propagation-reason-pol M (−L);
      case C of
        Some C ⇒ do {
          ASSERT(lit-redundant-reason-stack-wl-lookup-pre (−L) NU C);
          isa-lit-redundant-rec-wl-lookup M NU D cach
     [lit-redundant-reason-stack-wl-lookup (−L) NU C] lbd}
      | None ⇒ do {
          RETURN (cach, [], False)
      }
    }
  }›

lemma in-$\mathcal{L}_{all}$-atm-of-$\mathcal{A}_{in}$D[intro]: ‹L ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹ atm-of L ∈# $\mathcal{A}$›
  using in-$\mathcal{L}_{all}$-atm-of-$\mathcal{A}_{in}$ by blast

lemma isa-literal-redundant-wl-lookup-literal-redundant-wl-lookup:
  assumes ‹isasat-input-bounded $\mathcal{A}$›
  shows ‹(uncurry5 isa-literal-redundant-wl-lookup, uncurry5 (literal-redundant-wl-lookup $\mathcal{A}$)) ∈
    [λ(((((-, N), -), -), -), -). literals-are-in-$\mathcal{L}_{in}$-mm $\mathcal{A}$ ((mset ∘ fst) `# ran-m N)]$_f$
     trail-pol $\mathcal{A}$ ×$_f$ {(arena, N). valid-arena arena N vdom} ×$_f$ lookup-clause-rel $\mathcal{A}$ ×$_f$ cach-refinement
$\mathcal{A}$
        ×$_f$ Id ×$_f$ Id →
      ⟨cach-refinement $\mathcal{A}$ ×$_r$ Id ×$_r$ bool-rel⟩nres-rel›
proof −
  have [intro!]: ‹(x2g, x′) ∈ cach-refinement $\mathcal{A}$ ⟹
    (x2g, x′) ∈ cach-refinement (fold-mset (+) $\mathcal{A}$ {#})› for x2g x′
    by auto
  have [refine0]: ‹get-propagation-reason-pol M (− L)
    ≤ ⇓ (⟨Id⟩option-rel)
      (get-propagation-reason M′ (− L′))›
    if ‹(M, M′) ∈ trail-pol $\mathcal{A}$› and ‹(L, L′) ∈ Id› and ‹−L ∈ lits-of-l M′›
    for M M′ L L′
     using that get-propagation-reason-pol[of $\mathcal{A}$, THEN fref-to-Down-curry, of M′ ‹−L′› M ‹−L›] by
auto

  show ?thesis
    unfolding isa-literal-redundant-wl-lookup-def literal-redundant-wl-lookup-def uncurry-def
    apply (intro frefI nres-relI)
    apply (refine-vcg
     isa-lit-redundant-rec-wl-lookup-lit-redundant-rec-wl-lookup[of $\mathcal{A}$ vdom, THEN fref-to-Down-curry5])
    subgoal
      by (rule get-level-pol-pre) auto
    subgoal by (rule conflict-min-cach-l-pre) auto
    subgoal
      by (auto simp: get-level-get-level-pol in-$\mathcal{L}_{all}$-atm-of-$\mathcal{A}_{in}$D
           nth-conflict-min-cach[THEN fref-to-Down-unRET-uncurry-Id])
 (subst (asm) nth-conflict-min-cach[THEN fref-to-Down-unRET-uncurry-Id]; auto)+
    subgoal by auto
    subgoal for x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g
```

x2h x2i
by (*subst nth-conflict-min-cach*[*THEN fref-to-Down-unRET-uncurry-Id*];
auto simp del: *conflict-min-cach-def*)
(*auto simp*: *get-level-get-level-pol in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *D*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
apply *assumption*
subgoal by *auto*
subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g x2h x2i xa x' xb x'a*
unfolding *lit-redundant-reason-stack-wl-lookup-pre-def*
by (*auto simp*: *lit-redundant-reason-stack-wl-lookup-pre-def arena-lit-pre-def arena-is-valid-clause-idx-and-access-def arena-is-valid-clause-idx-def*
simp: *valid-arena-nempty*
intro!: *exI*[*of - xb*])
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal for *x y x1 x1a x1b x1c x1d x2 x2a x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g x2h x2i xa x' xb x'a*
by (*simp add*: *lit-redundant-reason-stack-wl-lookup-def*
lit-redundant-reason-stack-def lit-redundant-reason-stack-wl-lookup-pre-def
lit-redundant-reason-stack2-def
arena-lifting[*of x2e x2 vdom*]) — I have no idea why ⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *header-size* (*?N* ∝ *?i*) ≤ *?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *?i* < *length ?arena*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *is-Size* (*?arena* ! (*?i* − *SIZE-SHIFT*))
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *length* (*?N* ∝ *?i*) = *arena-length ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*; *?j* < *length* (*?N* ∝ *?i*)⟧ ⟹ *?N* ∝ *?i* ! *?j* = *arena-lit ?arena* (*?i* + *?j*)
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*; *?j* < *length* (*?N* ∝ *?i*)⟧ ⟹ *is-Lit* (*?arena* ! (*?i* + *?j*))
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*; *?j* < *length* (*?N* ∝ *?i*)⟧ ⟹ *?i* + *?j* < *length ?arena*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *?N* ∝ *?i* ! *0* = *arena-lit ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *is-Lit* (*?arena* ! *?i*)
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *?i* + *length* (*?N* ∝ *?i*) ≤ *length ?arena*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*; *is-long-clause* (*?N* ∝ *?i*)⟧ ⟹ *is-Pos* (*?arena* ! (*?i* − *POS-SHIFT*))
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*; *is-long-clause* (*?N* ∝ *?i*)⟧ ⟹ *arena-pos ?arena ?i* ≤ *arena-length ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *True*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *is-Status* (*?arena* ! (*?i* − *LBD-SHIFT*))
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *SIZE-SHIFT* ≤ *?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *LBD-SHIFT* ≤ *?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *True*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *2* ≤ *arena-length ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *Suc 0* ≤ *arena-length ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *0* ≤ *arena-length ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *Suc 0* < *arena-length ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *0* < *arena-length ?arena ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ (*arena-status ?arena ?i* = *LEARNED*) = (¬ *irred ?N ?i*)
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ (*arena-status ?arena ?i* = *IRRED*) = *irred ?N ?i*
⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *arena-status ?arena ?i* ≠ *DELETED*

⟦*valid-arena ?arena ?N ?vdom*; *?i* ∈# *dom-m ?N*⟧ ⟹ *Misc.slice ?i* (*?i* + *arena-length ?arena ?i*) *?arena* = *map ALit* (*?N* ∝ *?i*) requires to be instantiated.
done
qed

definition (**in** −) *lookup-conflict-remove1* :: ‹*nat literal* ⇒ *lookup-clause-rel* ⇒ *lookup-clause-rel*› **where**
‹*lookup-conflict-remove1* =
(λ*L* (*n*,*xs*). (*n*−*1*, *xs* [*atm-of L* := *NOTIN*]))›

lemma *lookup-conflict-remove1*:
‹(*uncurry* (*RETURN oo lookup-conflict-remove1*), *uncurry* (*RETURN oo remove1-mset*))
∈ [λ(*L*,*C*). *L* ∈# *C* ∧ −*L* ∉# *C* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$
Id ×$_f$ *lookup-clause-rel* $\mathcal{A}$ → ⟨*lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*›
apply (*intro frefI nres-relI*)
apply (*case-tac y*; *case-tac x*)
subgoal for *x y a b aa ab c*
using *mset-as-position-remove*[*of c b* ‹*atm-of aa*›]
by (*cases* ‹*aa*›)
(*auto simp*: *lookup-clause-rel-def lookup-conflict-remove1-def lookup-clause-rel-atm-in-iff*
minus-notin-trivial2 size-remove1-mset-If in-$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff minus-notin-trivial*
mset-as-position-in-iff-nth)
done

definition (**in** −) *lookup-conflict-remove1-pre* :: ‹*nat literal* × *nat* × *bool option list* ⇒ *bool*› **where**
‹*lookup-conflict-remove1-pre* = (λ(*L*,(*n*,*xs*)). *n* > *0* ∧ *atm-of L* < *length xs*)›

definition *isa-minimize-and-extract-highest-lookup-conflict*
:: ‹*trail-pol* ⇒ *arena* ⇒ *lookup-clause-rel* ⇒ *conflict-min-cach-l* ⇒ *lbd* ⇒
out-learned ⇒ (*lookup-clause-rel* × *conflict-min-cach-l* × *out-learned*) *nres*›
where
‹*isa-minimize-and-extract-highest-lookup-conflict* = (λ*M NU nxs s lbd outl*. *do* {
- ← *RETURN* (*IsaSAT-Profile.start-minimization*);
(*D*, -, *s*, *outl*) ←
WHILE$_T$$^{\lambda(nxs,\ i,\ s,\ outl).\ length\ outl \le unat32\text{-}max}$
(λ(*nxs*, *i*, *s*, *outl*). *i* < *length outl*)
(λ(*nxs*, *x*, *s*, *outl*). *do* {
ASSERT(*x* < *length outl*);
let L = *outl* ! *x*;
(*s'*, -, *red*) ← *isa-literal-redundant-wl-lookup M NU nxs s L lbd*;
if ¬*red*
then RETURN (*nxs*, *x*+*1*, *s'*, *outl*)
else do {
ASSERT(*lookup-conflict-remove1-pre* (*L*, *nxs*));
RETURN (*lookup-conflict-remove1 L nxs*, *x*, *s'*, *delete-index-and-swap outl x*)
}
})
(*nxs*, *1*, *s*, *outl*);
- ← *RETURN* (*IsaSAT-Profile.stop-minimization*);
RETURN (*D*, *s*, *outl*)
})›

lemma *isa-minimize-and-extract-highest-lookup-conflict-alt-def*:
‹*isa-minimize-and-extract-highest-lookup-conflict* = (λ*M NU nxs s lbd outl*. *do* {
(*D*, -, *s*, *outl*) ←
WHILE$_T$$^{\lambda(nxs,\ i,\ s,\ outl).\ length\ outl \le unat32\text{-}max}$

```
  (λ(nxs, i, s, outl). i < length outl)
  (λ(nxs, x, s, outl). do {
  ASSERT(x < length outl);
 let L = outl ! x;
  (s', -, red) ← isa-literal-redundant-wl-lookup M NU nxs s L lbd;
  if ¬red
 then RETURN (nxs, x+1, s', outl)
  else do {
  ASSERT(lookup-conflict-remove1-pre (L, nxs));
  RETURN (lookup-conflict-remove1 L nxs, x, s',  delete-index-and-swap outl x)
  }
  })
  (nxs, 1, s, outl);
  RETURN (D, s, outl)
  })›
  unfolding isa-minimize-and-extract-highest-lookup-conflict-def
    IsaSAT-Profile.start-def IsaSAT-Profile.stop-def nres-monad1
  ..

lemma isa-minimize-and-extract-highest-lookup-conflict-minimize-and-extract-highest-lookup-conflict:
  assumes ‹isasat-input-bounded A›
  shows ‹(uncurry5 isa-minimize-and-extract-highest-lookup-conflict,
    uncurry5 (minimize-and-extract-highest-lookup-conflict A)) ∈
    [λ(((((-, N), D), -), -), -). literals-are-in-ℒ_in-mm A ((mset ∘ fst) `# ran-m N) ∧
      ¬tautology D]_f
     trail-pol A ×_f {(arena, N). valid-arena arena N vdom} ×_f lookup-clause-rel A ×_f
        cach-refinement A ×_f Id  ×_f Id →
      ⟨lookup-clause-rel A ×_r cach-refinement A ×_r Id⟩nres-rel›
proof −
  have init: ‹((x2f, 1, x2g, x2i), x2a::nat literal multiset, 1, x2b, x2d)
       ∈ lookup-clause-rel A ×_r Id ×_r cach-refinement A ×_r Id ›
    if
      ‹(x, y)
      ∈ trail-pol A ×_f {(arena, N). valid-arena arena N vdom} ×_f lookup-clause-rel A ×_f
        cach-refinement A ×_f Id ×_f Id› and
      ‹x1c = (x1d, x2)› and
      ‹x1b = (x1c, x2a)› and
      ‹x1a = (x1b, x2b)› and
      ‹x1 = (x1a, x2c)› and
      ‹y = (x1, x2d)› and
      ‹x1h = (x1i, x2e)› and
      ‹x1g = (x1h, x2f)› and
      ‹x1f = (x1g, x2g)› and
      ‹x1e = (x1f, x2h)› and
      ‹x = (x1e, x2i)›
    for x y x1 x1a x1b x1c x1d x2 x2b x2c x2d x1e x1f x1g x1h x1i x2e x2f x2g
       x2h x2i and
       x2a
  proof −
    show ?thesis
      using that by auto
  qed

  show ?thesis
    unfolding isa-minimize-and-extract-highest-lookup-conflict-alt-def uncurry-def
      minimize-and-extract-highest-lookup-conflict-def
```

apply (*intro frefI nres-relI*)
apply (*refine-vcg*
isa-literal-redundant-wl-lookup-literal-redundant-wl-lookup[*of* $\mathcal{A}$ *vdom*, *THEN fref-to-Down-curry5*])
apply (*rule init*; *assumption*)
subgoal by (*auto simp*: *minimize-and-extract-highest-lookup-conflict-inv-def*)
subgoal by *auto*
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal
by (*auto simp*: *lookup-conflict-remove1-pre-def lookup-clause-rel-def atms-of-def*
minimize-and-extract-highest-lookup-conflict-inv-def)
subgoal
by (*auto simp*: *minimize-and-extract-highest-lookup-conflict-inv-def*
intro!: *lookup-conflict-remove1*[*THEN fref-to-Down-unRET-uncurry*]
simp: *nth-in-set-tl delete-from-lookup-conflict-pre-def*
dest!: *in-set-takeD*)
subgoal by *auto*
done
qed

definition *set-empty-conflict-to-none* **where**
‹*set-empty-conflict-to-none D* = *None*›

definition *set-lookup-empty-conflict-to-none* **where**
‹*set-lookup-empty-conflict-to-none* = (λ(*n*, *xs*). (*True*, *n*, *xs*))›

lemma *set-empty-conflict-to-none-hnr*:
‹(*RETURN o set-lookup-empty-conflict-to-none*, *RETURN o set-empty-conflict-to-none*) ∈
[λ*D*. *D* = {#}]$_f$ *lookup-clause-rel* $\mathcal{A}$ → ⟨*option-lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def*
set-empty-conflict-to-none-def set-lookup-empty-conflict-to-none-def)

definition *lookup-merge-eq2*
:: ‹*nat literal* ⇒ (*nat*,*nat*) *ann-lits* ⇒ *nat clause-l* ⇒ *conflict-option-rel* ⇒ *nat* ⇒
out-learned ⇒ (*conflict-option-rel* × *nat* × *out-learned*) *nres*› **where**
‹*lookup-merge-eq2 L M N* = (λ(-, *zs*) *clvls outl*. *do* {
ASSERT(*length N* = *2*);
let L' = (*if N* ! *0* = *L then N* ! *1 else N* ! *0*);
ASSERT(*get-level M L'* ≤ *Suc* (*unat32-max div 2*));
ASSERT(*atm-of L'* < *length* (*snd zs*));
ASSERT(*length outl* < *unat32-max*);
let outl = *outlearned-add M L' zs outl*;
ASSERT(*clvls* < *unat32-max*);
ASSERT(*fst zs* < *unat32-max*);
let clvls = *clvls-add M L' zs clvls*;
let zs = *add-to-lookup-conflict L' zs*;
RETURN((*False*, *zs*), *clvls*, *outl*)
})›

definition *merge-conflict-m-eq2*
:: ‹*nat literal* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *nat clause-l* ⇒ *nat clause option* ⇒
(*nat clause option* × *nat* × *out-learned*) *nres*›
where
‹*merge-conflict-m-eq2 L M Ni D* =
SPEC (λ(*C*, *n*, *outl*). *C* = *Some* (*remove1-mset L* (*mset Ni*) ∪# *the D*) ∧
n = *card-max-lvl M* (*remove1-mset L* (*mset Ni*) ∪# *the D*) ∧
out-learned M C outl)›

lemma *lookup-merge-eq2-spec*:
assumes
o: ‹((*b*, *n*, *xs*), *Some C*) ∈ *option-lookup-clause-rel* 𝒜› **and**
dist: ‹*distinct D*› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ 𝒜 (*mset D*)› **and**
lits-tr: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* 𝒜 *M*› **and**
n-d: ‹*no-dup M*› **and**
tauto: ‹¬*tautology* (*mset D*)› **and**
lits-C: ‹*literals-are-in-*$\mathcal{L}_{in}$ 𝒜 *C*› **and**
no-tauto: ‹⋀*K*. *K* ∈ *set* (*remove1 L D*) ⟹ − *K* ∉# *C*›
‹*clvls* = *card-max-lvl M C*› **and**
out: ‹*out-learned M* (*Some C*) *outl*› **and**
bounded: ‹*isasat-input-bounded* 𝒜› **and**
le2: ‹*length D* = *2*› **and**
L-D: ‹*L* ∈ *set D*›
shows
‹*lookup-merge-eq2 L M D* (*b*, *n*, *xs*) *clvls outl* ≤
⇓(*option-lookup-clause-rel* 𝒜 ×$_r$ *Id* ×$_r$ *Id*)
(*merge-conflict-m-eq2 L M D* (*Some C*))›
(**is** ‹- ≤ ⇓ *?Ref ?Spec*›)
proof −
let *?D* = ‹*remove1 L D*›
have *le-D-le-upper*[*simp*]: ‹*a* < *length D* ⟹ *Suc* (*Suc a*) ≤ *unat32-max*› **for** *a*
using *simple-clss-size-upper-div2*[*of* 𝒜 ‹*mset D*›] *assms* **by** (*auto simp*: *unat32-max-def*)
have *Suc-N-unat32-max*: ‹*Suc n* ≤ *unat32-max*› **and**
size-C-unat32-max: ‹*size C* ≤ *1* + *unat32-max div 2*› **and**
clvls: ‹*clvls* = *card-max-lvl M C*› **and**
tauto-C: ‹¬ *tautology C*› **and**
dist-C: ‹*distinct-mset C*› **and**
atms-le-xs: ‹∀ *L*∈*atms-of* (𝓛$_{all}$ 𝒜). *L* < *length xs*› **and**
map: ‹*mset-as-position xs C*›
using *assms simple-clss-size-upper-div2*[*of* 𝒜 *C*] *mset-as-position-distinct-mset*[*of xs C*]
lookup-clause-rel-not-tautolgy[*of n xs C*] *bounded*
unfolding *option-lookup-clause-rel-def lookup-clause-rel-def*
by (*auto simp*: *unat32-max-def*)
then have *clvls-unat32-max*: ‹*clvls* ≤ *1* + *unat32-max div 2*›
using *size-filter-mset-lesseq*[*of* ‹λ*L*. *get-level M L* = *count-decided M*› *C*]
unfolding *unat32-max-def card-max-lvl-def* **by** *linarith*
have [*intro*]: ‹((*b*, *a*, *ba*), *Some C*) ∈ *option-lookup-clause-rel* 𝒜 ⟹ *literals-are-in-*$\mathcal{L}_{in}$ 𝒜 *C* ⟹
Suc (*Suc a*) ≤ *unat32-max*› **for** *b a ba C*
using *lookup-clause-rel-size*[*of a ba C*, *OF - bounded*] **by** (*auto simp*: *option-lookup-clause-rel-def*
lookup-clause-rel-def unat32-max-def)
have [*simp*]: ‹*remdups-mset C* = *C*›
using *o mset-as-position-distinct-mset*[*of xs C*] **by** (*auto simp*: *option-lookup-clause-rel-def*
lookup-clause-rel-def distinct-mset-remdups-mset-id)
have ‹¬*tautology C*›
using *mset-as-position-tautology o* **by** (*auto simp*: *option-lookup-clause-rel-def*

```
      lookup-clause-rel-def)
  have ‹distinct-mset C›
    using mset-as-position-distinct-mset[of - C] o
    unfolding option-lookup-clause-rel-def lookup-clause-rel-def by auto
  have ‹mset (tl outl) ⊆# C›
     using out by (auto simp: out-learned-def)
  from size-mset-mono[OF this] have outl-le: ‹length outl < unat32-max›
   using simple-clss-size-upper-div2[OF bounded lits-C] dist-C tauto-C by (auto simp: unat32-max-def)
  define L' where ‹L' ≡ if D ! 0 = L then D ! 1 else D ! 0›
  have L'-all: ‹L' ∈# 𝓛_all 𝒜›
    using lits le2 by (cases D; cases ‹tl D›)
      (auto simp: L'-def literals-are-in-𝓛_in-add-mset)
  then have L': ‹atm-of L' ∈ atms-of (𝓛_all 𝒜)›
    by (auto simp: atms-of-def)
  have DLL: ‹mset D = {#L, L'#}› ‹set D = {L, L'}› ‹L ≠ L'› ‹remove1 L D = [L']›
    using le2 L-D dist by (cases D; cases ‹tl D›; auto simp: L'-def; fail)+
  have ‹- L' ∈# C ⟹ False› and [simp]: ‹- L' ∉# C›
    using dist no-tauto by (auto simp: DLL)
  then have o': ‹((False, add-to-lookup-conflict L' (n, xs)), Some ({#L'#} ∪# C))
    ∈ option-lookup-clause-rel 𝒜›
    using o L'-all unfolding option-lookup-clause-rel-def
    by (auto intro!: add-to-lookup-conflict-lookup-clause-rel)
  have [iff]: ‹is-in-lookup-conflict (n, xs) L' ⟷ L' ∈# C›
    using o mset-as-position-in-iff-nth[of xs C L'] L' no-tauto
    apply (auto simp: is-in-lookup-conflict-def option-lookup-clause-rel-def
        lookup-clause-rel-def DLL is-pos-neg-not-is-pos
       split: option.splits)
    by (smt ‹- L' ∉# C› atm-of-uminus is-pos-neg-not-is-pos mset-as-position-in-iff-nth option.inject)
  have clvls-add: ‹clvls-add M L' (n, xs) clvls = card-max-lvl M ({#L'#} ∪# C)›
    by (cases ‹L' ∈# C›)
      (auto simp: clvls-add-def card-max-lvl-add-mset clvls add-mset-union
      dest!: multi-member-split)
  have out': ‹out-learned M (Some ({#L'#} ∪# C)) (outlearned-add M L' (n, xs) outl)›
    using out
    by (cases ‹L' ∈# C›)
      (auto simp: out-learned-def outlearned-add-def add-mset-union
      dest!: multi-member-split)

  show ?thesis
    unfolding lookup-merge-eq2-def prod.simps L'-def[symmetric]
    apply refine-vcg
    subgoal by (rule le2)
    subgoal using literals-are-in-𝓛_in-trail-get-level-unat32-max[OF bounded lits-tr n-d] by blast
    subgoal using atms-le-xs L' by simp
    subgoal using outl-le .
    subgoal using clvls-unat32-max by (auto simp: unat32-max-def)
    subgoal using Suc-N-unat32-max by auto
    subgoal
      using o' clvls-add out'
      by (auto simp: merge-conflict-m-eq2-def DLL
        intro!: RETURN-RES-refine)
    done
qed

definition isasat-lookup-merge-eq2
  :: ‹nat literal ⇒ trail-pol ⇒ arena ⇒ nat ⇒ conflict-option-rel ⇒ nat ⇒
```

```
    out-learned ⇒ (conflict-option-rel × nat × out-learned) nres› where
‹isasat-lookup-merge-eq2 L M N C = (λ(-, zs) clvls outl. do {
  ASSERT(arena-lit-pre N C);
  ASSERT(arena-lit-pre N (C+1));
  let L' = (if arena-lit N C = L then arena-lit N (C + 1) else arena-lit N C);
  ASSERT(get-level-pol-pre (M, L'));
  ASSERT(get-level-pol M L' ≤ Suc (unat32-max div 2));
  ASSERT(atm-of L' < length (snd zs));
  ASSERT(length outl < unat32-max);
  let outl = isa-outlearned-add M L' zs outl;
  ASSERT(clvls < unat32-max);
  ASSERT(fst zs < unat32-max);
  let clvls = isa-clvls-add M L' zs clvls;
  let zs = add-to-lookup-conflict L' zs;
  RETURN((False, zs), clvls, outl)
 })›

lemma isasat-lookup-merge-eq2-lookup-merge-eq2:
 assumes valid: ‹valid-arena arena N vdom› and i: ‹i ∈# dom-m N› and
  lits: ‹literals-are-in-ℒ_in-mm 𝒜 (mset `# ran-mf N)› and
  bxs: ‹((b, xs), C) ∈ option-lookup-clause-rel 𝒜› and
  M'M: ‹(M', M) ∈ trail-pol 𝒜› and
  bound: ‹isasat-input-bounded 𝒜›
 shows
  ‹isasat-lookup-merge-eq2 L M' arena i (b, xs) clvls outl ≤ ⇓ Id
    (lookup-merge-eq2 L M (N ∝ i) (b, xs) clvls outl)›
proof −
 define L' where ‹L' ≡ (if arena-lit arena i = L then arena-lit arena (i + 1)
      else arena-lit arena i)›
 define L'' where ‹L'' ≡ (if N ∝ i ! 0 = L then N ∝ i ! 1 else N ∝ i ! 0)›

 have [simp]: ‹L'' = L'›
  if ‹length (N ∝ i) = 2›
  using that i valid by (auto simp: L''-def L'-def arena-lifting)
 have L'-all: ‹L' ∈# ℒ_all 𝒜›
  if ‹length (N ∝ i) = 2›
  by (use lits i valid that
      literals-are-in-ℒ_in-mm-add-msetD[of 𝒜
   ‹mset (N ∝ i)› - ‹arena-lit arena (Suc i)›]
  literals-are-in-ℒ_in-mm-add-msetD[of 𝒜
   ‹mset (N ∝ i)› - ‹arena-lit arena i›]
  nth-mem[of 0 ‹N ∝ i›] nth-mem[of 1 ‹N ∝ i›]
 in ‹auto simp: arena-lifting ran-m-def L'-def
  simp del: nth-mem
   dest:
  dest!: multi-member-split›)

 show ?thesis
  unfolding isasat-lookup-merge-eq2-def lookup-merge-eq2-def prod.simps
  L'-def[symmetric] L''-def[symmetric]
  apply refine-vcg
  subgoal
   using valid i
   unfolding arena-lit-pre-def arena-is-valid-clause-idx-and-access-def
   by (auto intro!: exI[of - i] exI[of - N])
  subgoal
```

using *valid i*
unfolding *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
by (*auto intro*!: *exI*[*of - i*] *exI*[*of - N*])
subgoal
by (*rule get-level-pol-pre*[*OF - M′M*])
(*use L′-all*
in ‹*auto simp*: *arena-lifting ran-m-def*
simp del: *nth-mem*
dest:
dest!: *multi-member-split*›)
subgoal
by (*subst get-level-get-level-pol*[*OF M′M*, *symmetric*])
(*use L′-all* **in** *auto*)
subgoal by *auto*
subgoal
using *M′M L′-all*
by (*auto simp*: *isa-clvls-add-clvls-add get-level-get-level-pol*
isa-outlearned-add-outlearned-add)
done
qed

definition *merge-conflict-m-eq2-pre* **where**
‹*merge-conflict-m-eq2-pre* $\mathcal{A}$ =
(λ((((((*L*, *M*), *N*), *i*), *xs*), *clvls*), *out*). *i* ∈# *dom-m N* ∧ *xs* ≠ *None* ∧ *distinct* (*N* ∝ *i*) ∧
¬*tautology* (*mset* (*N* ∝ *i*)) ∧
(∀ *K* ∈ *set* (*remove1 L* (*N* ∝ *i*)). − *K* ∉# *the xs*) ∧
literals-are-in-$\mathcal{L}_{in}$ $\mathcal{A}$ (*the xs*) ∧ *clvls* = *card-max-lvl M* (*the xs*) ∧
out-learned M xs out ∧ *no-dup M* ∧
literals-are-in-$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* '# *ran-mf N*) ∧
isasat-input-bounded $\mathcal{A}$ ∧
length (*N* ∝ *i*) = *2* ∧
L ∈ *set* (*N* ∝ *i*))›

definition *merge-conflict-m-g-eq2* :: ‹-› **where**
‹*merge-conflict-m-g-eq2 L M N i D* - - = *merge-conflict-m-eq2 L M* (*N* ∝ *i*) *D*›

lemma *isasat-lookup-merge-eq2*:
‹(*uncurry6 isasat-lookup-merge-eq2*, *uncurry6 merge-conflict-m-g-eq2*) ∈
[*merge-conflict-m-eq2-pre* $\mathcal{A}$]$_f$
Id $\times_f$ *trail-pol* $\mathcal{A}$ $\times_f$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *nat-rel* $\times_f$ *option-lookup-clause-rel*
$\mathcal{A}$
$\times_f$ *nat-rel* $\times_f$ *Id* →
⟨*option-lookup-clause-rel* $\mathcal{A}$ $\times_r$ *nat-rel* $\times_r$ *Id*⟩*nres-rel*›
proof −
have *H1*: ‹*isasat-lookup-merge-eq2 a* (*aa*, *ab*, *ac*, *ad*, *ae*, *b*) *ba bb* (*af*, *ag*, *bc*) *be*
bf
≤ ⇓ *Id* (*lookup-merge-eq2 a bg* (*bh* ∝ *bb*) (*af*, *ag*, *bc*) *be bf*)›
if
‹*merge-conflict-m-eq2-pre* $\mathcal{A}$ (((((((*ah*, *bg*), *bh*), *bi*), *bj*), *bk*)), *bm*)› **and**
‹((((((((*a*, *aa*, *ab*, *ac*, *ad*, *ae*, *b*), *ba*), *bb*), *af*, *ag*, *bc*)), *be*), *bf*),
((((((*ah*, *bg*), *bh*), *bi*), *bj*), *bk*)), *bm*)
∈ *Id* $\times_f$ *trail-pol* $\mathcal{A}$ $\times_f$ {(*arena*, *N*). *valid-arena arena N vdom*} $\times_f$ *nat-rel* $\times_f$
option-lookup-clause-rel $\mathcal{A}$ $\times_f$ *nat-rel* $\times_f$
Id›

for *a aa ab ac ad ae b ba bb af ag bc bd be bf ah bg bh bi bj bm bk*
proof −
have
bi: ‹*bi* ∈# *dom-m bh*› **and**
‹(*bf*, *bm*) ∈ *Id*› **and**
‹*bj* ≠ *None*› **and**
‹*distinct* (*bh* ∝ *bi*)› **and**
‹(*be*, *bk*) ∈ *nat-rel*› **and**
‹¬ *tautology* (*mset* (*bh* ∝ *bi*))› **and**
o: ‹((*af*, *ag*, *bc*), *bj*) ∈ *option-lookup-clause-rel* $\mathcal{A}$› **and**
‹∀ *K*∈*set* (*remove1 ah* (*bh* ∝ *bi*)). − *K* ∉# *the bj*› **and**
st: ‹*bb* = *bi*› **and**
‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*the bj*)› **and**
valid: ‹*valid-arena ba bh vdom*› **and**
‹*bk* = *card-max-lvl bg* (*the bj*)› **and**
‹(*a*, *ah*) ∈ *Id*› **and**
tr: ‹((*aa*, *ab*, *ac*, *ad*, *ae*, *b*), *bg*) ∈ *trail-pol* $\mathcal{A}$› **and**
‹*out-learned bg bj bm*› **and**
‹*no-dup bg*› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf bh*)› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$› **and**
ah: ‹*ah* ∈ *set* (*bh* ∝ *bi*)›
using *that* **unfolding** *merge-conflict-m-eq2-pre-def prod.simps prod-rel-iff*
by *blast*+

show *?thesis*
by (*rule isasat-lookup-merge-eq2-lookup-merge-eq2*[*OF valid bi*[*unfolded st*[*symmetric*]]
lits o tr bounded])
qed
have *H2*: ‹*lookup-merge-eq2 a bg* (*bh* ∝ *bb*) (*af*, *ag*, *bc*) *be bf*
≤ ⇓ (*option-lookup-clause-rel* $\mathcal{A}$ ×$_f$ (*nat-rel* ×$_f$ *Id*))
(*merge-conflict-m-g-eq2 ah bg bh bi bj bl bm*)›
if
‹*merge-conflict-m-eq2-pre* $\mathcal{A}$ (((((((*ah*, *bg*), *bh*), *bi*), *bj*)), *bl*), *bm*)› **and**
‹(((((((*a*, *aa*, *ab*, *ac*, *ad*, *ae*, *b*), *ba*), *bb*), *af*, *ag*, *bc*), *be*), *bf*),
((((((*ah*, *bg*), *bh*), *bi*), *bj*)), *bl*), *bm*)
∈ *Id* ×$_f$ *trail-pol* $\mathcal{A}$ ×$_f$ {(*arena*, *N*). *valid-arena arena N vdom*} ×$_f$ *nat-rel* ×$_f$
option-lookup-clause-rel $\mathcal{A}$ ×$_f$ *nat-rel* ×$_f$ *Id*›
for *a aa ab ac ad ae b ba bb af ag bc be bf ah bg bh bi bj bl bm*
proof −
have
bi: ‹*bi* ∈# *dom-m bh*› **and**
bj: ‹*bj* ≠ *None*› **and**
dist: ‹*distinct* (*bh* ∝ *bi*)› **and**
tauto: ‹¬ *tautology* (*mset* (*bh* ∝ *bi*))› **and**
o: ‹((*af*, *ag*, *bc*), *bj*) ∈ *option-lookup-clause-rel* $\mathcal{A}$› **and**
K: ‹∀ *K*∈*set* (*remove1 ah* (*bh* ∝ *bi*)). − *K* ∉# *the bj*› **and**
st: ‹*bb* = *bi*›
‹*bf* = *bm*›
‹*be* = *bl*›
‹*a* = *ah*› **and**
lits-confl: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*the bj*)› **and**
valid: ‹*valid-arena ba bh vdom*› **and**
bk: ‹*bl* = *card-max-lvl bg* (*the bj*)› **and**
tr: ‹((*aa*, *ab*, *ac*, *ad*, *ae*, *b*), *bg*) ∈ *trail-pol* $\mathcal{A}$› **and**
out: ‹*out-learned bg bj bm*› **and**

```
    ‹no-dup bg› and
    lits: ‹literals-are-in-\mathcal{L}_{in}-mm \mathcal{A} (mset `# ran-mf bh)› and
    bounded: ‹isasat-input-bounded \mathcal{A}› and
    le2: ‹length (bh ∝ bi) = 2› and
    ah: ‹ah ∈ set (bh ∝ bi)›
    using that unfolding merge-conflict-m-eq2-pre-def prod.simps prod-rel-iff
    by blast+
  obtain bj' where bj': ‹bj = Some bj'›
    using bj by (cases bj) auto
  have n-d: ‹no-dup bg› and lits-tr: ‹literals-are-in-\mathcal{L}_{in}-trail \mathcal{A} bg›
    using tr unfolding trail-pol-alt-def
    by auto
  have lits-bi: ‹literals-are-in-\mathcal{L}_{in} \mathcal{A} (mset (bh ∝ bi))›
    using bi lits by (auto simp: literals-are-in-\mathcal{L}_{in}-mm-add-mset ran-m-def
      dest!: multi-member-split)

  show ?thesis
    unfolding st merge-conflict-m-g-eq2-def
    apply (rule lookup-merge-eq2-spec[THEN order-trans, OF o[unfolded bj']
      dist lits-bi lits-tr n-d tauto lits-confl[unfolded bj' option.sel]
      - bk[unfolded bj' option.sel] - bounded le2 ah])
    subgoal using K unfolding bj' by auto
    subgoal using out unfolding bj' .
    subgoal unfolding bj' by auto
    done
qed

show ?thesis
  unfolding lookup-conflict-merge-def uncurry-def
  apply (intro nres-relI frefI)
  apply clarify
  subgoal for a aa ab ac ad ae b ba bb af ag bc bd bf ah bg bh bi bj bk bl
    apply (rule H1[THEN order-trans]; assumption?)
    apply (subst Down-id-eq)
    apply (rule H2)
    apply assumption+
    done
  done
qed

definition (in −) get-count-max-lvls-code where
  ‹get-count-max-lvls-code = (λ(-, -, -, -, -, -, -, clvls, -). clvls)›

lemma atm-of-in-atms-of: ‹atm-of x ∈ atms-of C ⟷ x ∈# C ∨ −x ∈# C›
  using atm-of-notin-atms-of-iff by blast

definition atm-is-in-conflict where
  ‹atm-is-in-conflict L D ⟷ atm-of L ∈ atms-of (the D)›

fun is-in-option-lookup-conflict where
  is-in-option-lookup-conflict-def[simp del]:
  ‹is-in-option-lookup-conflict L (a, n, xs) ⟷ is-in-lookup-conflict (n, xs) L›


lemma is-in-option-lookup-conflict-atm-is-in-conflict-iff:
  assumes
```

```
    ‹ba ≠ None› and aa: ‹aa ∈# 𝓛_all 𝒜› and uaa: ‹− aa ∉# the ba› and
    ‹((b, c, d), ba) ∈ option-lookup-clause-rel 𝒜›
  shows ‹is-in-option-lookup-conflict aa (b, c, d) =
        atm-is-in-conflict aa ba›
proof −
  obtain yb where ba[simp]: ‹ba = Some yb›
    using assms by auto

  have map: ‹mset-as-position d yb› and le: ‹∀ L∈atms-of (𝓛_all 𝒜). L < length d› and [simp]: ‹¬b›
    using assms by (auto simp: option-lookup-clause-rel-def lookup-clause-rel-def)
  have aa-d: ‹atm-of aa < length d› and uaa-d: ‹atm-of (−aa) < length d›
    using le aa by (auto simp: in-𝓛_all-atm-of-in-atms-of-iff)
  from mset-as-position-in-iff-nth[OF map aa-d]
  have 1: ‹(aa ∈# yb) = (d ! atm-of aa = Some (is-pos aa))›
    .

  from mset-as-position-in-iff-nth[OF map uaa-d] have 2: ‹(d ! atm-of aa ≠ Some (is-pos (−aa)))›
    using uaa by simp

  then show ?thesis
    using uaa 1 2
    by (auto simp: is-in-lookup-conflict-def is-in-option-lookup-conflict-def atm-is-in-conflict-def
        atm-of-in-atms-of is-neg-neg-not-is-neg
        split: option.splits)
qed

lemma is-in-option-lookup-conflict-atm-is-in-conflict:
  ‹(uncurry (RETURN oo is-in-option-lookup-conflict), uncurry (RETURN oo atm-is-in-conflict))
   ∈ [λ(L, D). D ≠ None ∧ L ∈# 𝓛_all 𝒜 ∧ −L ∉# the D]_f
      Id ×_f option-lookup-clause-rel 𝒜 → ⟨bool-rel⟩nres-rel›
  apply (intro frefI nres-relI)
  apply (case-tac x, case-tac y)
  by (simp add: is-in-option-lookup-conflict-atm-is-in-conflict-iff[of - - 𝒜])

lemma is-in-option-lookup-conflict-alt-def:
  ‹RETURN oo is-in-option-lookup-conflict =
    RETURN oo (λL (-, n, xs). is-in-lookup-conflict (n, xs) L)›
  by (auto intro!: ext simp: is-in-option-lookup-conflict-def)

end
theory IsaSAT-VDom
  imports IsaSAT-Stats IsaSAT-Clauses
begin
```

AI-vdom We keep all the indices. This is a subset of *vdom* but is cleaned more aggressively. At first we traid to express the relation directly but this was too cumbersome to use, due to having both sets and multisets.

```
type-synonym vdom = ‹nat list›
type-synonym aivdom = ‹vdom × vdom × vdom × vdom›
type-synonym isasat-aivdom = ‹aivdom code-hider›

abbreviation get-aivdom :: ‹isasat-aivdom ⇒ aivdom› where
  ‹get-aivdom ≡ get-content›

abbreviation AIvdom :: ‹aivdom ⇒ isasat-aivdom› where
```

‹*AIvdom* ≡ *Constructor*›

fun *get-vdom-aivdom* **where**
‹*get-vdom-aivdom* (*AIvdom* (*vdom*, *avdom*, *ivdom*, *tvdom*)) = *vdom*›

fun *get-avdom-aivdom* **where**
‹*get-avdom-aivdom* (*AIvdom* (*vdom*, *avdom*, *ivdom*, *tvdom*)) = *avdom*›

fun *get-ivdom-aivdom* **where**
‹*get-ivdom-aivdom* (*AIvdom* (*vdom*, *avdom*, *ivdom*, *tvdom*)) = *ivdom*›

fun *get-tvdom-aivdom* **where**
‹*get-tvdom-aivdom* (*AIvdom* (*vdom*, *avdom*, *ivdom*, *tvdom*)) = *tvdom*›

fun *aivdom-inv* :: ‹*aivdom* ⇒ - ⇒ *bool*› **where**
‹*aivdom-inv* (*vdom*, *avdom*, *ivdom*, *tvdom*) *d* ⟷
set avdom ∩ *set ivdom* = {} ∧
set-mset d ⊆ *set avdom* ∪ *set ivdom* ∧
mset avdom ⊆# *mset vdom* ∧
mset ivdom ⊆# *mset vdom* ∧
distinct-mset d ∧
distinct vdom ∧
distinct tvdom ∧
mset tvdom ⊆# *mset vdom*›

fun *aivdom-inv-strong* :: ‹*aivdom* ⇒ - ⇒ *bool*› **where**
‹*aivdom-inv-strong* (*vdom*, *avdom*, *ivdom*, *tvdom*) *d* ⟷
(*aivdom-inv* (*vdom*, *avdom*, *ivdom*, *tvdom*) *d* ∧ *tvdom* = *vdom*)›

definition *aivdom-inv-dec* :: ‹*isasat-aivdom* ⇒ - ⇒ *bool*› **where**
‹*aivdom-inv-dec* = *aivdom-inv o get-aivdom*›

definition *aivdom-inv-strong-dec* :: ‹*isasat-aivdom* ⇒ - ⇒ *bool*› **where**
‹*aivdom-inv-strong-dec* = *aivdom-inv-strong o get-aivdom*›

lemma *aivdom-inv-strong-dec-def2*:
‹*aivdom-inv-strong-dec x a* ⟷ *aivdom-inv-dec x a* ∧ *get-vdom-aivdom x* = *get-tvdom-aivdom x*›
by (*cases x*) (*auto simp*: *aivdom-inv-strong-dec-def aivdom-inv-dec-def*)

lemma *aivdom-inv-alt-def*:
‹*aivdom-inv* (*vdom*, *avdom*, *ivdom*, *tvdom*) *d* ⟷
(*set avdom* ∩ *set ivdom* = {} ∧
set-mset d ⊆ *set avdom* ∪ *set ivdom* ∧
set avdom ⊆ *set vdom* ∧
set ivdom ⊆ *set vdom* ∧
distinct vdom∧
distinct ivdom∧
distinct-mset d ∧
distinct avdom∧
distinct tvdom ∧
set tvdom ⊆ *set vdom*)›
using *distinct-mset-mono*[*of* ‹*mset* (*avdom*)› ‹*mset* (*vdom*)›]
distinct-mset-mono[*of* ‹*mset* (*ivdom*)› ‹*mset* (*vdom*)›]
distinct-subseteq-iff[*of* ‹*mset* (*avdom*)› ‹*mset* (*vdom*)›]
distinct-subseteq-iff[*of* ‹*mset* (*ivdom*)› ‹*mset* (*vdom*)›]
distinct-subseteq-iff[*of* ‹*mset* (*tvdom*)› ‹*mset* (*vdom*)›]

by *auto*

lemma *AIvdom-split*:
‹*aivdom* = *AIvdom* (*get-vdom-aivdom aivdom*, *get-avdom-aivdom aivdom*, *get-ivdom-aivdom aivdom*, *get-tvdom-aivdom aivdom*)›
by (*cases aivdom*) *auto*

lemma *aivdom-inv-dec-alt-def*:
‹*aivdom-inv-dec aivdom d* ⟷
(*set* (*get-avdom-aivdom aivdom*) ∩ *set* (*get-ivdom-aivdom aivdom*) = {} ∧
set-mset d ⊆ *set* (*get-avdom-aivdom aivdom*) ∪ *set* (*get-ivdom-aivdom aivdom*) ∧
mset (*get-avdom-aivdom aivdom*) ⊆# *mset* (*get-vdom-aivdom aivdom*) ∧
mset (*get-ivdom-aivdom aivdom*) ⊆# *mset* (*get-vdom-aivdom aivdom*) ∧
distinct-mset d ∧ *distinct* (*get-vdom-aivdom aivdom*) ∧
mset (*get-tvdom-aivdom aivdom*) ⊆# *mset* (*get-vdom-aivdom aivdom*) ∧ *distinct* (*get-tvdom-aivdom aivdom*))›
apply (*subst AIvdom-split*)
apply (*subst aivdom-inv-dec-def*)
apply (*subst comp-def*)
apply (*auto simp add*: *aivdom-inv-alt-def*)
done

lemma *aivdom-inv-dec-alt-def2*:
‹*aivdom-inv-dec aivdom d* ⟷
(*set* (*get-avdom-aivdom aivdom*) ∩ *set* (*get-ivdom-aivdom aivdom*) = {} ∧
set-mset d ⊆ *set* (*get-avdom-aivdom aivdom*) ∪ *set* (*get-ivdom-aivdom aivdom*) ∧
mset (*get-avdom-aivdom aivdom*) ⊆# *mset* (*get-vdom-aivdom aivdom*) ∧
mset (*get-ivdom-aivdom aivdom*) ⊆# *mset* (*get-vdom-aivdom aivdom*) ∧
distinct-mset d ∧ *distinct* (*get-vdom-aivdom aivdom*) ∧ *distinct* (*get-avdom-aivdom aivdom*) ∧
distinct (*get-ivdom-aivdom aivdom*)∧
mset (*get-tvdom-aivdom aivdom*) ⊆# *mset* (*get-vdom-aivdom aivdom*) ∧ *distinct* (*get-tvdom-aivdom aivdom*))›
apply (*subst AIvdom-split*)
apply (*subst aivdom-inv-dec-def*)
apply (*subst comp-def*)
apply (*auto dest*: *distinct-mset-mono simp add*: *aivdom-inv-alt-def*)
done

lemmas *aivdom-inv-strong-dec-alt-def* =
aivdom-inv-strong-dec-def2[*unfolded aivdom-inv-dec-alt-def2*]

lemma *distinct-butlast-set*:
‹*distinct xs* ⟹ *set* (*butlast xs*) = *set xs* − {*last xs*}›
by (*cases xs rule*: *rev-cases*) *auto*

lemma *distinct-remove-readd-last-set*:
‹*distinct xs* ⟹ *i* < *length xs* ⟹ *set* (*butlast* (*xs*[*i* := *last xs*])) = *set xs* − {*xs*!*i*}›
by (*cases xs rule*: *rev-cases*) (*auto simp*: *list-update-append set-update-distinct nth-append*)

definition *add-learned-clause-aivdom-int* **where**
‹*add-learned-clause-aivdom-int* = (λ *C* (*vdom*, *avdom*, *ivdom*). (*vdom* @ [*C*], *avdom* @ [*C*], *ivdom*))›

definition *add-learned-clause-aivdom* :: ‹- ⇒ *isasat-aivdom* ⇒ *isasat-aivdom*› **where**
‹*add-learned-clause-aivdom C* ≡ *AIvdom o add-learned-clause-aivdom-int C o get-aivdom*›

lemma *aivdom-inv-intro-add-mset*:
‹*C* ∉# *d* ⟹ *C* ∉ *set vdom* ⟹ *aivdom-inv* (*vdom*, *avdom*, *ivdom*, *tvdom*) *d* ⟹ *aivdom-inv* (*vdom* @ [*C*], *avdom* @ [*C*], *ivdom*, *tvdom*) (*add-mset C d*)›
unfolding *aivdom-inv-alt-def*
by (*cases* ‹*C* ∈ (*set* (*avdom*) ∪ *set* (*ivdom*))›)
(*auto dest*: *subset-mset-imp-subset-add-mset simp*: *add-learned-clause-aivdom-int-def split*: *code-hider.splits*)

lemma *aivdom-inv-dec-intro-add-mset*:
‹*C* ∉# *d* ⟹ *C* ∉ *set* (*get-vdom-aivdom aivdom*) ⟹ *aivdom-inv-dec aivdom d* ⟹ *aivdom-inv-dec* (*add-learned-clause-aivdom C aivdom*) (*add-mset C d*)›
using *aivdom-inv-intro-add-mset*[*of C d* ‹*get-vdom-aivdom aivdom*› ‹*get-avdom-aivdom aivdom*›
‹*get-ivdom-aivdom aivdom*›]
by (*cases aivdom*)
(*auto simp*: *aivdom-inv-dec-def add-learned-clause-aivdom-int-def add-learned-clause-aivdom-def*
simp del: *aivdom-inv.simps*)

definition *add-init-clause-aivdom-int* **where**
‹*add-init-clause-aivdom-int* = (λ *C* (*vdom*, *avdom*, *ivdom*, *tvdom*). (*vdom* @ [*C*], *avdom*, *ivdom* @ [*C*], *tvdom*))›

definition *add-init-clause-aivdom* :: ‹- ⇒ *isasat-aivdom* ⇒ *isasat-aivdom*› **where**
‹*add-init-clause-aivdom C* ≡ *AIvdom o add-init-clause-aivdom-int C o get-aivdom*›

lemma *aivdom-inv-intro-init-add-mset*:
‹*C* ∉# *d* ⟹ *C* ∉ *set vdom* ⟹ *aivdom-inv* (*vdom*, *avdom*, *ivdom*, *tvdom*) *d* ⟹ *aivdom-inv* (*vdom* @ [*C*], *avdom*, *ivdom* @ [*C*], *tvdom*) (*add-mset C d*)›
unfolding *aivdom-inv-alt-def*
by (*cases* ‹*C* ∈ (*set* (*avdom*) ∪ *set* (*ivdom*))›)
(*auto dest*: *subset-mset-imp-subset-add-mset simp*: *add-init-clause-aivdom-int-def split*: *code-hider.splits*)

lemma *aivdom-inv-dec-intro-init-add-mset*:
‹*C* ∉# *d* ⟹ *C* ∉ *set* (*get-vdom-aivdom aivdom*) ⟹ *aivdom-inv-dec aivdom d* ⟹ *aivdom-inv-dec* (*add-init-clause-aivdom C aivdom*) (*add-mset C d*)›
using *aivdom-inv-intro-init-add-mset*[*of C d* ‹*get-vdom-aivdom aivdom*› ‹*get-avdom-aivdom aivdom*›
‹*get-ivdom-aivdom aivdom*› ‹*get-tvdom-aivdom aivdom*›]
by (*cases aivdom*)
(*auto simp*: *aivdom-inv-dec-def add-init-clause-aivdom-int-def add-init-clause-aivdom-def*
simp del: *aivdom-inv.simps*)

definition *remove-inactive-aivdom-tvdom-int* :: ‹- ⇒ *aivdom* ⇒ *aivdom*› **where**
‹*remove-inactive-aivdom-tvdom-int* = (λ*i* (*vdom*, *avdom*, *ivdom*, *tvdom*). (*vdom*, *avdom*, *ivdom*, *delete-index-and-swap tvdom i*))›

definition *remove-inactive-aivdom-tvdom* :: ‹- ⇒ *isasat-aivdom* ⇒ *isasat-aivdom*› **where**
‹*remove-inactive-aivdom-tvdom C* ≡ *AIvdom o remove-inactive-aivdom-tvdom-int C o get-aivdom*›

definition *remove-inactive-aivdom-int* :: ‹- ⇒ *aivdom* ⇒ *aivdom*› **where**
‹*remove-inactive-aivdom-int* = (λ*i* (*vdom*, *avdom*, *ivdom*, *tvdom*). (*vdom*, *delete-index-and-swap avdom i*, *ivdom*, *tvdom*))›

definition *remove-inactive-aivdom* :: ‹- ⇒ *isasat-aivdom* ⇒ *isasat-aivdom*› **where**
‹*remove-inactive-aivdom C* ≡ *AIvdom o remove-inactive-aivdom-int C o get-aivdom*›

lemma *aivdom-inv-remove-and-swap-inactive-tvdom*:
assumes ‹*i* < *length tv*› **and** ‹*aivdom-inv* (*m*, *n*, *s*, *tv*) *baa*›
shows ‹*aivdom-inv* (*m*, *n*, *s*, *butlast* (*tv*[*i* := *last tv*])) (*remove1-mset* (*tv* ! *i*) *baa*)›

proof −
show *?thesis*
using *assms distinct-mset-mono*[*of* ‹*mset n*› ‹*mset m*›]
by (*auto simp*: *aivdom-inv-alt-def distinct-remove-readd-last-set mset-le-subtract distinct-butlast-set*
dest: *in-set-butlastD in-vdom-m-fmdropD simp del*: *nth-mem*)
qed
lemma *aivdom-inv-dec-remove-and-swap-inactive-tvdom*:
assumes ‹*i* < *length* (*get-tvdom-aivdom ai*)› **and** ‹*aivdom-inv-dec ai baa*›
shows ‹*aivdom-inv-dec* (*remove-inactive-aivdom-tvdom i ai*) (*remove1-mset* (*get-tvdom-aivdom ai* ! *i*) *baa*)›
using *aivdom-inv-remove-and-swap-inactive-tvdom*[*of i* ‹*get-tvdom-aivdom ai*›
‹*get-vdom-aivdom ai*› ‹*get-avdom-aivdom ai*› ‹*get-ivdom-aivdom ai*› *baa*] *assms*
by (*cases ai*; *auto simp*: *aivdom-inv-dec-def remove-inactive-aivdom-tvdom-int-def remove-inactive-aivdom-tvdom-def*
simp del: *aivdom-inv.simps*)

lemma *aivdom-inv-remove-and-swap-inactive*:
assumes ‹*i* < *length n*› **and** ‹*aivdom-inv* (*m*, *n*, *s*, *tv*) *baa*›
shows ‹*aivdom-inv* (*m*, *butlast* (*n*[*i* := *last n*]), *s*, *tv*) (*remove1-mset* (*n* ! *i*) *baa*)›
proof −
have [*simp*]: ‹*set n* − {*n* ! *i*} ∪ *set s* = *set n* ∪ *set s* − {*n* ! *i*}›
using *assms nth-mem*[*of i n*]
by (*auto simp*: *aivdom-inv-alt-def distinct-remove-readd-last-set*
dest: *in-set-butlastD in-vdom-m-fmdropD simp del*: *nth-mem*)
have [*simp*]: ‹*mset-set* (*set n* ∪ *set s* − {*n* ! *i*}) = *remove1-mset* (*n*!*i*) (*mset-set* (*set n* ∪ *set s*))›
using *assms*
by (*auto simp*: *aivdom-inv-alt-def mset-set-Diff*)
show *?thesis*
using *assms distinct-mset-mono*[*of* ‹*mset n*› ‹*mset m*›]
by (*auto simp*: *aivdom-inv-alt-def distinct-remove-readd-last-set mset-le-subtract distinct-butlast-set*
dest: *in-set-butlastD in-vdom-m-fmdropD simp del*: *nth-mem*)
qed

lemma *aivdom-inv-dec-remove-and-swap-inactive*:
assumes ‹*i* < *length* (*get-avdom-aivdom ai*)› **and** ‹*aivdom-inv-dec ai baa*›
shows ‹*aivdom-inv-dec* (*remove-inactive-aivdom i ai*) (*remove1-mset* (*get-avdom-aivdom ai* ! *i*) *baa*)›
using *aivdom-inv-remove-and-swap-inactive*[*of i* ‹*get-avdom-aivdom ai*›
‹*get-vdom-aivdom ai*› ‹*get-ivdom-aivdom ai*› ‹*get-tvdom-aivdom ai*› *baa*] *assms*
by (*cases ai*; *auto simp*: *aivdom-inv-dec-def remove-inactive-aivdom-int-def remove-inactive-aivdom-def*
simp del: *aivdom-inv.simps*)

lemma *aivdom-inv-remove-clause*:
‹*aivdom-inv ai baa* ⟹ *aivdom-inv ai* (*remove1-mset C baa*)›
by (*cases ai*) (*auto simp*: *aivdom-inv-alt-def distinct-remove-readd-last-set*
dest: *in-set-butlastD dest*: *in-diffD*)

lemma *aivdom-inv-dec-remove-clause*:
‹*aivdom-inv-dec ai baa* ⟹ *aivdom-inv-dec ai* (*remove1-mset C baa*)›
using *aivdom-inv-remove-clause*[*of* ‹*get-content ai*› *baa*]
by (*auto simp*: *aivdom-inv-dec-def*)

lemma *aivdom-inv-removed-inactive*:
assumes ‹*i* < *length n*› **and** ‹*aivdom-inv* (*m*, *n*, *s*, *tv*) *baa*› ‹*n*!*i* ∉# *baa*›
shows ‹*aivdom-inv* (*m*, *butlast* (*n*[*i* := *last n*]), *s*, *tv*) *baa*›
by (*metis aivdom-inv-remove-and-swap-inactive assms*(*1*) *assms*(*2*) *assms*(*3*) *diff-single-trivial*)

lemma *aivdom-inv-dec-removed-inactive*:

assumes ‹$i < length\ (get\text{-}avdom\text{-}aivdom\ ai)$› **and** ‹$aivdom\text{-}inv\text{-}dec\ ai\ baa$› ‹$get\text{-}avdom\text{-}aivdom\ ai!i \notin\# baa$›
shows ‹$aivdom\text{-}inv\text{-}dec\ (remove\text{-}inactive\text{-}aivdom\ i\ ai)\ baa$›
using $aivdom\text{-}inv\text{-}removed\text{-}inactive[OF\ assms(1),\ of$ ‹$get\text{-}vdom\text{-}aivdom\ ai$› ‹$get\text{-}ivdom\text{-}aivdom\ ai$› ‹$get\text{-}tvdom\text{-}aivdom\ ai$› $baa]\ assms(2-)$
by $(cases\ ai)$
$(auto\ simp: aivdom\text{-}inv\text{-}dec\text{-}def\ remove\text{-}inactive\text{-}aivdom\text{-}int\text{-}def$
$remove\text{-}inactive\text{-}aivdom\text{-}def\ simp\ del: aivdom\text{-}inv.simps)$

lemmas $aivdom\text{-}inv\text{-}remove\text{-}and\text{-}swap\text{-}removed = aivdom\text{-}inv\text{-}removed\text{-}inactive$

lemma $aivdom\text{-}inv\text{-}removed\text{-}inactive\text{-}tvdom$:
assumes ‹$i < length\ tv$› **and** ‹$aivdom\text{-}inv\ (m,\ n,\ s,\ tv)\ baa$› ‹$tv!i \notin\# baa$›
shows ‹$aivdom\text{-}inv\ (m,\ n,\ s,\ butlast\ (tv[i := last\ tv]))\ baa$›
by $(metis\ aivdom\text{-}inv\text{-}remove\text{-}and\text{-}swap\text{-}inactive\text{-}tvdom\ assms(1)\ assms(2)\ assms(3)\ diff\text{-}single\text{-}trivial)$

lemma $aivdom\text{-}inv\text{-}dec\text{-}removed\text{-}inactive\text{-}tvdom$:
assumes ‹$i < length\ (get\text{-}tvdom\text{-}aivdom\ ai)$› **and** ‹$aivdom\text{-}inv\text{-}dec\ ai\ baa$› ‹$get\text{-}tvdom\text{-}aivdom\ ai!i \notin\# baa$›
shows ‹$aivdom\text{-}inv\text{-}dec\ (remove\text{-}inactive\text{-}aivdom\text{-}tvdom\ i\ ai)\ baa$›
using $aivdom\text{-}inv\text{-}removed\text{-}inactive\text{-}tvdom[OF\ assms(1),\ of$ ‹$get\text{-}vdom\text{-}aivdom\ ai$› ‹$get\text{-}avdom\text{-}aivdom\ ai$› ‹$get\text{-}ivdom\text{-}aivdom\ ai$› $baa]\ assms(2-)$
by $(cases\ ai)$
$(auto\ simp: aivdom\text{-}inv\text{-}dec\text{-}def\ remove\text{-}inactive\text{-}aivdom\text{-}tvdom\text{-}int\text{-}def$
$remove\text{-}inactive\text{-}aivdom\text{-}tvdom\text{-}def\ simp\ del: aivdom\text{-}inv.simps)$

lemma $get\text{-}aivdom\text{-}remove\text{-}inactive\text{-}aivdom[simp]$:
‹$get\text{-}vdom\text{-}aivdom\ (remove\text{-}inactive\text{-}aivdom\ i\ m) = get\text{-}vdom\text{-}aivdom\ m$›
‹$get\text{-}avdom\text{-}aivdom\ (remove\text{-}inactive\text{-}aivdom\ i\ m) = (delete\text{-}index\text{-}and\text{-}swap\ (get\text{-}avdom\text{-}aivdom\ m)\ i)$›
‹$get\text{-}ivdom\text{-}aivdom\ (remove\text{-}inactive\text{-}aivdom\ i\ m) = get\text{-}ivdom\text{-}aivdom\ m$›
‹$get\text{-}tvdom\text{-}aivdom\ (remove\text{-}inactive\text{-}aivdom\ i\ m) = get\text{-}tvdom\text{-}aivdom\ m$›
‹$get\text{-}vdom\text{-}aivdom\ (remove\text{-}inactive\text{-}aivdom\text{-}tvdom\ i\ m) = get\text{-}vdom\text{-}aivdom\ m$›
‹$get\text{-}tvdom\text{-}aivdom\ (remove\text{-}inactive\text{-}aivdom\text{-}tvdom\ i\ m) = (delete\text{-}index\text{-}and\text{-}swap\ (get\text{-}tvdom\text{-}aivdom\ m)\ i)$›
‹$get\text{-}avdom\text{-}aivdom\ (remove\text{-}inactive\text{-}aivdom\text{-}tvdom\ i\ m) = get\text{-}avdom\text{-}aivdom\ m$›
‹$get\text{-}ivdom\text{-}aivdom\ (remove\text{-}inactive\text{-}aivdom\text{-}tvdom\ i\ m) = get\text{-}ivdom\text{-}aivdom\ m$›
by $(cases\ m;\ auto\ simp: remove\text{-}inactive\text{-}aivdom\text{-}def\ remove\text{-}inactive\text{-}aivdom\text{-}int\text{-}def$
$remove\text{-}inactive\text{-}aivdom\text{-}tvdom\text{-}def\ remove\text{-}inactive\text{-}aivdom\text{-}tvdom\text{-}int\text{-}def;\ fail)+$

definition $vdom\text{-}aivdom\text{-}at\text{-}int :: ‹aivdom \Rightarrow nat \Rightarrow nat›$ **where**
‹$vdom\text{-}aivdom\text{-}at\text{-}int\ = (\lambda(a,b,c)\ C.\ a\ !\ C)$›

definition $vdom\text{-}aivdom\text{-}at :: ‹isasat\text{-}aivdom \Rightarrow nat \Rightarrow nat›$ **where**
‹$vdom\text{-}aivdom\text{-}at\ ai\ C = get\text{-}vdom\text{-}aivdom\ ai\ !\ C$›

lemma $vdom\text{-}aivdom\text{-}at\text{-}alt\text{-}def$:
‹$vdom\text{-}aivdom\text{-}at = vdom\text{-}aivdom\text{-}at\text{-}int\ o\ get\text{-}content$›
by $(intro\ ext,\ rename\text{-}tac\ x\ y,\ case\text{-}tac\ x)\ (auto\ simp: vdom\text{-}aivdom\text{-}at\text{-}int\text{-}def\ vdom\text{-}aivdom\text{-}at\text{-}def)$

definition $avdom\text{-}aivdom\text{-}at\text{-}int :: ‹aivdom \Rightarrow nat \Rightarrow nat›$ **where**
‹$avdom\text{-}aivdom\text{-}at\text{-}int = (\lambda(a,b,c)\ C.\ b\ !\ C)$›

definition *avdom-aivdom-at* :: ‹*isasat-aivdom* ⇒ *nat* ⇒ *nat*› **where**
‹*avdom-aivdom-at ai C* = *get-avdom-aivdom ai* ! *C*›

lemma *avdom-aivdom-at-alt-def*:
‹*avdom-aivdom-at* = *avdom-aivdom-at-int o get-content*›
by (*intro ext*, *rename-tac x y*, *case-tac x*) (*auto simp*: *avdom-aivdom-at-int-def avdom-aivdom-at-def*)

definition *ivdom-aivdom-at-int* :: ‹*aivdom* ⇒ *nat* ⇒ *nat*› **where**
‹*ivdom-aivdom-at-int* = (λ(*a*,*b*,*c*,*d*) *C*. *c* ! *C*)›

definition *ivdom-aivdom-at* :: ‹*isasat-aivdom* ⇒ *nat* ⇒ *nat*› **where**
‹*ivdom-aivdom-at ai C* = *get-ivdom-aivdom ai* ! *C*›

lemma *ivdom-aivdom-at-alt-def*:
‹*ivdom-aivdom-at* = *ivdom-aivdom-at-int o get-content*›
by (*intro ext*, *rename-tac x y*, *case-tac x*) (*auto simp*: *ivdom-aivdom-at-int-def ivdom-aivdom-at-def*)

definition *tvdom-aivdom-at-int* :: ‹*aivdom* ⇒ *nat* ⇒ *nat*› **where**
‹*tvdom-aivdom-at-int* = (λ(*a*,*b*,*c*,*d*) *C*. *d* ! *C*)›

definition *tvdom-aivdom-at* :: ‹*isasat-aivdom* ⇒ *nat* ⇒ *nat*› **where**
‹*tvdom-aivdom-at ai C* = *get-tvdom-aivdom ai* ! *C*›

lemma *tvdom-aivdom-at-alt-def*:
‹*tvdom-aivdom-at* = *tvdom-aivdom-at-int o get-content*›
by (*intro ext*, *rename-tac x y*, *case-tac x*) (*auto simp*: *tvdom-aivdom-at-int-def tvdom-aivdom-at-def*)

definition *length-ivdom-aivdom-int* :: ‹*aivdom* ⇒ *nat*› **where**
‹*length-ivdom-aivdom-int* = (λ(*a*,*b*,*c*,*d*). *length c*)›

definition *length-ivdom-aivdom* :: ‹*isasat-aivdom* ⇒ *nat*› **where**
‹*length-ivdom-aivdom ai* = *length* (*get-ivdom-aivdom ai*)›

lemma *length-ivdom-aivdom-alt-def*:
‹*length-ivdom-aivdom* = *length-ivdom-aivdom-int o get-content*›
by (*intro ext*, *rename-tac x*, *case-tac x*) (*auto simp*: *length-ivdom-aivdom-def length-ivdom-aivdom-int-def*)

definition *length-avdom-aivdom-int* :: ‹*aivdom* ⇒ *nat*› **where**
‹*length-avdom-aivdom-int* = (λ(*a*,*b*,*c*). *length b*)›

definition *length-avdom-aivdom* :: ‹*isasat-aivdom* ⇒ *nat*› **where**
‹*length-avdom-aivdom ai* = *length* (*get-avdom-aivdom ai*)›

lemma *length-avdom-aivdom-alt-def*:
‹*length-avdom-aivdom* = *length-avdom-aivdom-int o get-content*›
by (*intro ext*, *rename-tac x*, *case-tac x*) (*auto simp*: *length-avdom-aivdom-def length-avdom-aivdom-int-def*)

definition *length-vdom-aivdom-int* :: ‹*aivdom* ⇒ *nat*› **where**
‹*length-vdom-aivdom-int* = (λ(*a*,*b*,*c*). *length a*)›

definition *length-vdom-aivdom* :: ‹*isasat-aivdom* ⇒ *nat*› **where**
‹*length-vdom-aivdom ai* = *length* (*get-vdom-aivdom ai*)›

lemma *length-vdom-aivdom-alt-def*:
‹*length-vdom-aivdom* = *length-vdom-aivdom-int o get-content*›

by (*intro ext*, *rename-tac x*, *case-tac x*) (*auto simp*: *length-vdom-aivdom-def length-vdom-aivdom-int-def*)

definition *length-tvdom-aivdom-int* :: ‹*aivdom* ⇒ *nat*› **where**
‹*length-tvdom-aivdom-int* = (λ(*a*,*b*,*c*,*d*). *length d*)›

definition *length-tvdom-aivdom* :: ‹*isasat-aivdom* ⇒ *nat*› **where**
‹*length-tvdom-aivdom ai* = *length* (*get-tvdom-aivdom ai*)›

lemma *length-tvdom-aivdom-alt-def*:
‹*length-tvdom-aivdom* = *length-tvdom-aivdom-int o get-content*›
by (*intro ext*, *rename-tac x*, *case-tac x*) (*auto simp*: *length-tvdom-aivdom-def length-tvdom-aivdom-int-def*)

definition *add-init-clause-aivdom-strong-int* **where**
‹*add-init-clause-aivdom-strong-int* = (λ *C* (*vdom*, *avdom*, *ivdom*, *tvdom*). (*vdom* @ [*C*], *avdom*, *ivdom* @ [*C*], *tvdom* @ [*C*]))›

definition *add-init-clause-aivdom-strong* :: ‹- ⇒ *isasat-aivdom* ⇒ *isasat-aivdom*› **where**
‹*add-init-clause-aivdom-strong C* ≡ *AIvdom o add-init-clause-aivdom-strong-int C o get-aivdom*›

lemma *aivdom-inv-intro-init-strong-add-mset*:
‹*C* ∉# *d* ⟹ *C* ∉ *set vdom* ⟹ *aivdom-inv-strong* (*vdom*, *avdom*, *ivdom*, *tvdom*) *d* ⟹ *aivdom-inv-strong* (*vdom* @ [*C*], *avdom*, *ivdom* @ [*C*], *tvdom* @ [*C*]) (*add-mset C d*)›
unfolding *aivdom-inv-alt-def*
by (*cases* ‹*C* ∈ (*set* (*avdom*) ∪ *set* (*ivdom*))›)
(*auto dest*: *subset-mset-imp-subset-add-mset simp*: *add-init-clause-aivdom-strong-int-def*
split: *code-hider.splits*)

lemma *aivdom-inv-dec-intro-init-strong-add-mset*:
‹*C* ∉# *d* ⟹ *C* ∉ *set* (*get-vdom-aivdom aivdom*) ⟹ *aivdom-inv-strong-dec aivdom d* ⟹ *aivdom-inv-strong-dec* (*add-init-clause-aivdom-strong C aivdom*) (*add-mset C d*)›
using *aivdom-inv-intro-init-strong-add-mset*[*of C d* ‹*get-vdom-aivdom aivdom*› ‹*get-avdom-aivdom aivdom*›
‹*get-ivdom-aivdom aivdom*› ‹*get-tvdom-aivdom aivdom*›]
by (*cases aivdom*)
(*auto simp*: *aivdom-inv-dec-def add-init-clause-aivdom-strong-int-def add-init-clause-aivdom-strong-def*
aivdom-inv-strong-dec-def
simp del: *aivdom-inv.simps*)

definition *add-learned-clause-aivdom-strong-int* **where**
‹*add-learned-clause-aivdom-strong-int* = (λ *C* (*vdom*, *avdom*, *ivdom*, *tvdom*). (*vdom* @ [*C*], *avdom* @ [*C*], *ivdom*, *tvdom* @ [*C*]))›

definition *add-learned-clause-aivdom-strong* :: ‹- ⇒ *isasat-aivdom* ⇒ *isasat-aivdom*› **where**
‹*add-learned-clause-aivdom-strong C* ≡ *AIvdom o add-learned-clause-aivdom-strong-int C o get-aivdom*›

lemma *aivdom-inv-intro-learned-strong-add-mset*:
‹*C* ∉# *d* ⟹ *C* ∉ *set vdom* ⟹ *aivdom-inv-strong* (*vdom*, *avdom*, *ivdom*, *tvdom*) *d* ⟹ *aivdom-inv-strong* (*vdom* @ [*C*], *avdom* @ [*C*], *ivdom*, *tvdom* @ [*C*]) (*add-mset C d*)›
unfolding *aivdom-inv-alt-def*
by (*cases* ‹*C* ∈ (*set* (*avdom*) ∪ *set* (*ivdom*))›)
(*auto dest*: *subset-mset-imp-subset-add-mset simp*: *add-learned-clause-aivdom-strong-int-def*
split: *code-hider.splits*)

lemma *aivdom-inv-dec-intro-learned-strong-add-mset*:
‹*C* ∉# *d* ⟹ *C* ∉ *set* (*get-vdom-aivdom aivdom*) ⟹ *aivdom-inv-strong-dec aivdom d* ⟹ *aiv-*

dom-inv-strong-dec (*add-learned-clause-aivdom-strong C aivdom*) (*add-mset C d*)›
using *aivdom-inv-intro-learned-strong-add-mset*[*of C d* ‹*get-vdom-aivdom aivdom*› ‹*get-avdom-aivdom aivdom*›
‹*get-ivdom-aivdom aivdom*› ‹*get-tvdom-aivdom aivdom*›]
by (*cases aivdom*)
(*auto simp*: *aivdom-inv-dec-def add-learned-clause-aivdom-strong-int-def add-learned-clause-aivdom-strong-def aivdom-inv-strong-dec-def*
simp del: *aivdom-inv.simps*)

lemma [*simp*]:
‹*get-vdom-aivdom* (*add-learned-clause-aivdom C aivdom*) = *get-vdom-aivdom aivdom* @ [*C*]›
‹*get-avdom-aivdom* (*add-learned-clause-aivdom C aivdom*) = *get-avdom-aivdom aivdom* @ [*C*]›
‹*get-ivdom-aivdom* (*add-learned-clause-aivdom C aivdom*) = *get-ivdom-aivdom aivdom*›
‹*get-tvdom-aivdom* (*add-learned-clause-aivdom C aivdom*) = *get-tvdom-aivdom aivdom*›
‹*get-vdom-aivdom* (*add-init-clause-aivdom C aivdom*) = *get-vdom-aivdom aivdom* @ [*C*]›
‹*get-avdom-aivdom* (*add-init-clause-aivdom C aivdom*) = *get-avdom-aivdom aivdom*›
‹*get-ivdom-aivdom* (*add-init-clause-aivdom C aivdom*) = *get-ivdom-aivdom aivdom* @ [*C*]›
‹*get-tvdom-aivdom* (*add-init-clause-aivdom C aivdom*) = *get-tvdom-aivdom aivdom*›
‹*get-vdom-aivdom* (*add-learned-clause-aivdom-strong C aivdom*) = *get-vdom-aivdom aivdom* @ [*C*]›
‹*get-avdom-aivdom* (*add-learned-clause-aivdom-strong C aivdom*) = *get-avdom-aivdom aivdom* @ [*C*]›
‹*get-ivdom-aivdom* (*add-learned-clause-aivdom-strong C aivdom*) = *get-ivdom-aivdom aivdom*›
‹*get-tvdom-aivdom* (*add-learned-clause-aivdom-strong C aivdom*) = *get-tvdom-aivdom aivdom* @ [*C*]›
‹*get-vdom-aivdom* (*add-init-clause-aivdom-strong C aivdom*) = *get-vdom-aivdom aivdom* @ [*C*]›
‹*get-avdom-aivdom* (*add-init-clause-aivdom-strong C aivdom*) = *get-avdom-aivdom aivdom*›
‹*get-ivdom-aivdom* (*add-init-clause-aivdom-strong C aivdom*) = *get-ivdom-aivdom aivdom* @ [*C*]›
‹*get-tvdom-aivdom* (*add-init-clause-aivdom-strong C aivdom*) = *get-tvdom-aivdom aivdom* @ [*C*]›
by (*cases aivdom*; *auto simp*: *add-learned-clause-aivdom-def add-learned-clause-aivdom-int-def*
add-learned-clause-aivdom-strong-def add-learned-clause-aivdom-strong-int-def
add-init-clause-aivdom-strong-def add-init-clause-aivdom-strong-int-def
add-init-clause-aivdom-def add-init-clause-aivdom-int-def; *fail*)+

definition *empty-aivdom-int* **where**
‹*empty-aivdom-int* = (λ(*vdom*, *avdom*, *ivdom*, *tvdom*). (*take 0 vdom*, *take 0 avdom*, *take 0 ivdom*, *take 0 tvdom*))›

definition *empty-aivdom* :: ‹*isasat-aivdom* ⇒ *isasat-aivdom*› **where**
‹*empty-aivdom* = *Constructor o empty-aivdom-int o get-content*›

lemma [*simp*]:
‹*get-vdom-aivdom* (*empty-aivdom aivdom*) = []›
‹*get-avdom-aivdom* (*empty-aivdom aivdom*) = []›
‹*get-ivdom-aivdom* (*empty-aivdom aivdom*) = []›
‹*get-tvdom-aivdom* (*empty-aivdom aivdom*) = []›
‹*aivdom-inv-dec* (*empty-aivdom aivdom*) {#}›
‹*aivdom-inv-strong-dec* (*empty-aivdom aivdom*) {#}›
by (*auto simp*: *empty-aivdom-def empty-aivdom-int-def aivdom-inv-dec-alt-def*
aivdom-inv-strong-dec-def)

fun *swap-avdom-aivdom* **where**
‹*swap-avdom-aivdom* (*AIvdom* (*vdom*, *avdom*, *ivdom*, *tvdom*)) *i j* =
(*AIvdom* (*vdom*, *swap avdom i j*, *ivdom*, *tvdom*))›

lemma [*simp*]:
‹*get-avdom-aivdom* (*swap-avdom-aivdom aivdom i j*) = *swap* (*get-avdom-aivdom aivdom*) *i j*›
‹*get-vdom-aivdom* (*swap-avdom-aivdom aivdom i j*) = (*get-vdom-aivdom aivdom*)›
‹*get-ivdom-aivdom* (*swap-avdom-aivdom aivdom i j*) = (*get-ivdom-aivdom aivdom*)›

‹*get-tvdom-aivdom* (*swap-avdom-aivdom aivdom i j*) = (*get-tvdom-aivdom aivdom*)›
by (*cases aivdom*; *auto simp*:)+

fun *take-avdom-aivdom* **where**
‹*take-avdom-aivdom i* (*AIvdom* (*vdom*, *avdom*, *ivdom*, *tvdom*)) =
(*AIvdom* (*vdom*, *take i avdom*, *ivdom*, *tvdom*))›

lemma [*simp*]:
‹*get-avdom-aivdom* (*take-avdom-aivdom i aivdom*) = *take i* (*get-avdom-aivdom aivdom*)›
‹*get-vdom-aivdom* (*take-avdom-aivdom i aivdom*) = *get-vdom-aivdom aivdom*›
‹*get-tvdom-aivdom* (*take-avdom-aivdom i aivdom*) = *get-tvdom-aivdom aivdom*›
‹*get-ivdom-aivdom* (*take-avdom-aivdom i aivdom*) = *get-ivdom-aivdom aivdom*›
by (*cases aivdom*; *auto simp*:; *fail*)+

definition *map-vdom-aivdom* :: ‹-› **where**
‹*map-vdom-aivdom f* = (λ*x*. *case x of AIvdom* (*vdom*, *avdom*, *ivdom*, *tvdom*) ⇒ *do* {
avdom ← *f avdom*;
RETURN (*AIvdom* (*vdom*, *avdom*, *ivdom*, *tvdom*))
})›

definition *map-tvdom-aivdom* :: ‹-› **where**
‹*map-tvdom-aivdom f* = (λ*x*. *case x of AIvdom* (*vdom*, *avdom*, *ivdom*, *tvdom*) ⇒ *do* {
tvdom ← *f tvdom*;
RETURN (*AIvdom* (*vdom*, *avdom*, *ivdom*, *tvdom*))
})›

definition *AIvdom-init* :: ‹*nat list* ⇒ *nat list* ⇒ *nat list* ⇒ *isasat-aivdom*› **where**
‹*AIvdom-init vdom avdom ivdom* = *AIvdom* (*vdom*, *avdom*, *ivdom*, *vdom*)›

definition *push-to-tvdom-int* **where**
‹*push-to-tvdom-int* = (λ *C* (*vdom*, *avdom*, *ivdom*, *tvdom*). (*vdom*, *avdom*, *ivdom*, *tvdom* @ [*C*]))›

definition *push-to-tvdom* :: ‹- ⇒ *isasat-aivdom* ⇒ *isasat-aivdom*› **where**
‹*push-to-tvdom C* ≡ *AIvdom o push-to-tvdom-int C o get-aivdom*›

lemma *aivdom-inv-push-to-tvdom-int*:
‹*C* ∈ *set vdom* ⟹ *C* ∉ *set tvdom* ⟹ *aivdom-inv* (*vdom*, *avdom*, *ivdom*, *tvdom*) *d* ⟹ *aivdom-inv*
(*vdom*, *avdom*, *ivdom*, *tvdom* @ [*C*]) *d*›
unfolding *aivdom-inv-alt-def*
by (*auto dest*: *subset-mset-imp-subset-add-mset simp*: *add-learned-clause-aivdom-strong-int-def*
split: *code-hider.splits*)

lemma *aivdom-push-to-tvdom*:
‹*C* ∈ *set* (*get-vdom-aivdom aivdom*) ⟹ *C* ∉ *set* (*get-tvdom-aivdom aivdom*) ⟹ *aivdom-inv-dec*
aivdom d ⟹ *aivdom-inv-dec* (*push-to-tvdom C aivdom*) *d*›
using *aivdom-inv-push-to-tvdom-int*[*of C* ‹*get-vdom-aivdom aivdom*› ‹*get-tvdom-aivdom aivdom*›
‹*get-avdom-aivdom aivdom*› ‹*get-ivdom-aivdom aivdom*› *d*]
by (*cases aivdom*)
(*auto simp*: *aivdom-inv-dec-def add-learned-clause-aivdom-strong-int-def add-learned-clause-aivdom-strong-def*
aivdom-inv-strong-dec-def push-to-tvdom-def push-to-tvdom-int-def
simp del: *aivdom-inv.simps*)

fun *empty-tvdom-int* **where**
‹*empty-tvdom-int* (*vdom*, *avdom*, *ivdom*, *tvdom*) = (*vdom*, *avdom*, *ivdom*, *take 0 tvdom*)›

definition *empty-tvdom* **where**
‹*empty-tvdom* = *AIvdom o empty-tvdom-int o get-content*›

lemma *get-aivdom-add-learned-clause-aivdom*[*simp*]:
‹*get-vdom-aivdom* (*add-learned-clause-aivdom x2 vdom*) = *get-vdom-aivdom vdom* @ [*x2*]›
‹*get-avdom-aivdom* (*add-learned-clause-aivdom x2 vdom*) = *get-avdom-aivdom vdom* @ [*x2*]›
‹*get-ivdom-aivdom* (*add-learned-clause-aivdom x2 vdom*) = *get-ivdom-aivdom vdom*›
by (*cases vdom*; *auto simp*: *add-learned-clause-aivdom-def add-learned-clause-aivdom-int-def*; *fail*)+

end
theory *IsaSAT-Occurence-List*
imports *IsaSAT-Literals IsaSAT-Watch-List IsaSAT-Mark*
begin

9.1 Occurrence lists

Occurrence lists (in a single watched way) are very similar to watch lists. For simplification purpose, we use occurrence lists and watch lists, but Kissat uses only the latter for memory efficiency.

This file started as an experiment. We attempted to do better than our watch lists because we know and understand a lot more on refinement. This turned out to do really work out as expected however.

There are two ways to achieve the refinement:

- the most abstract requires to know the set of variable. Which means that we have to somehow get it. Which is impossible (some of the information is deleted anyway), but also something we probably should not try to do.

- just use a bound.

What is the conclusion of all that? Well not much, except that full abstraction is hard to get. So we still end up with the concrete data in the isasat state, which I find sad, but I don't see any way to do it better – and no I am no going back to a locale with an upper bound on the literals; been there, done that.

9.1.1 Abstract Occurrence Lists

type-synonym *raw-occurences* = ‹*nat literal* ⇒ *nat list*›
type-synonym *occurences* = ‹*nat set* × *raw-occurences*›

definition *valid-occurrences* **where**
‹*valid-occurrences* $\mathcal{B}$ = (λ($\mathcal{A}$, *xs*). *set-mset* $\mathcal{B}$ = $\mathcal{A}$)›

Only useful for proofs.

definition *occ-list* :: ‹*occurences* ⇒ *nat literal* ⇒ *nat list*› **where**
‹*occ-list* = (λ($\mathcal{A}$, *xs*) *L*. *xs L*)›

definition *occ-list-at* :: ‹*occurences* ⇒ *nat literal* ⇒ *nat* ⇒ *nat*› **where**
‹*occ-list-at* $\mathcal{A}$*xs L i* = *occ-list* $\mathcal{A}$*xs L* ! *i*›

definition *occ-list-at-pre* :: ‹*occurences* ⇒ *nat literal* ⇒ *nat* ⇒ *bool*› **where**
‹*occ-list-at-pre* = (λ(*n*, *xs*) *L i*. *atm-of L* ∈ *n* ∧ *i* < *length* (*xs L*))›

definition *mop-occ-list-at* :: ‹*occurences* ⇒ *nat literal* ⇒ *nat* ⇒ *nat nres*› **where**
‹*mop-occ-list-at* = (λ$\mathcal{A}$*xs L i. do* {
ASSERT (*occ-list-at-pre* $\mathcal{A}$*xs L i*);
RETURN (*occ-list-at* $\mathcal{A}$*xs L i*)
})›

definition *occ-list-length-pre* :: ‹*occurences* ⇒ *nat literal* ⇒ *bool*› **where**
‹*occ-list-length-pre* = (λ($\mathcal{A}$, *xs*) *L. atm-of L* ∈ $\mathcal{A}$)›

definition *occ-list-length* :: ‹*occurences* ⇒ *nat literal* ⇒ *nat*› **where**
‹*occ-list-length* = (λ($\mathcal{A}$, *xs*) *L. do* {
(*length* (*xs L*))
})›

definition *mop-occ-list-length* :: ‹*occurences* ⇒ *nat literal* ⇒ *nat nres*› **where**
‹*mop-occ-list-length* = (λ$\mathcal{A}$*xs L. do* {
ASSERT (*occ-list-length-pre* $\mathcal{A}$*xs L*);
RETURN (*occ-list-length* $\mathcal{A}$*xs L*)
})›

definition *occ-list-append-pre* :: ‹*occurences* ⇒ *nat literal* ⇒ *bool*› **where**
‹*occ-list-append-pre* = (λ($\mathcal{A}$, *xs*) *L. atm-of L* ∈ $\mathcal{A}$)›

definition *occ-list-append* :: ‹*nat* ⇒ *occurences* ⇒ *nat literal* ⇒ *occurences*› **where**
‹*occ-list-append* = (λ*x* ($\mathcal{A}$, *xs*) *L.* ($\mathcal{A}$, *xs* (*L* := *xs L* @ [*x*])))›

definition *mop-occ-list-append* :: ‹*nat* ⇒ *occurences* ⇒ *nat literal* ⇒ *occurences nres*› **where**
‹*mop-occ-list-append* = (λ*x* $\mathcal{A}$*xs L . do* {
ASSERT (*occ-list-append-pre* ($\mathcal{A}$*xs*) *L*);
RETURN (*occ-list-append x* ($\mathcal{A}$*xs*) *L*)
})›

definition *occ-list-clear-at-pre* :: ‹*occurences* ⇒ *nat literal* ⇒ *bool*› **where**
‹*occ-list-clear-at-pre* = (λ($\mathcal{A}$, *xs*) *L. atm-of L* ∈ $\mathcal{A}$)›

definition *occ-list-clear-at* :: ‹*occurences* ⇒ *nat literal* ⇒ *occurences nres*› **where**
‹*occ-list-clear-at* = (λ($\mathcal{A}$, *xs*) *L . do* {
ASSERT (*occ-list-clear-at-pre* ($\mathcal{A}$, *xs*) *L*);
RETURN ($\mathcal{A}$, *xs* (*L* := []))
})›

definition *occ-list-clear-all-pre* :: ‹*occurences* ⇒ *bool*› **where**
‹*occ-list-clear-all-pre* = (λ($\mathcal{A}$, *xs*). *True*)›

definition *occ-list-clear-all* :: ‹*occurences* ⇒ *occurences nres*› **where**
‹*occ-list-clear-all* = (λ($\mathcal{A}$, *xs*) . *do* {
ASSERT (*occ-list-clear-all-pre* ($\mathcal{A}$, *xs*));
RETURN ($\mathcal{A}$, λ-. [])
})›

definition *all-occurrences* :: ‹*nat multiset* ⇒ *occurences* ⇒ *nat multiset*› **where**
‹*all-occurrences* $\mathcal{A}$ = (λ(*n*, *xs*). $\sum_{\#}$ (*mset* ‘# *xs* ‘# *Pos* ‘# *remdups-mset* $\mathcal{A}$ +
mset ‘# *xs* ‘# *Neg* ‘# *remdups-mset* $\mathcal{A}$))›

definition *occ-list-create-pre* :: ‹*nat set* ⇒ *nat set* ⇒ *bool*› **where**
‹*occ-list-create-pre n* = (λ$\mathcal{A}$. *True*)›

definition *mop-occ-list-create* :: ‹*nat set* ⇒ *occurences nres*› **where**
‹*mop-occ-list-create* = (λ*n*. *do* {
ASSERT (*finite n*);
RETURN (*n*, λ-. [])
})›

abbreviation *raw-empty-occs-list* :: ‹*nat literal* ⇒ *nat list*› **where**
‹*raw-empty-occs-list* ≡ (λ-. [])›

definition *empty-occs-list* :: ‹*nat multiset* ⇒ *nat set* × (*nat literal* ⇒ *nat list*)› **where**
‹*empty-occs-list* $\mathcal{A}$ ≡ (*set-mset* $\mathcal{A}$, λ-. [])›

lemma *empty-occs-list-cong*:
‹*set-mset A* = *set-mset B* ⟹ *empty-occs-list A* = *empty-occs-list B*›
unfolding *empty-occs-list-def* **by** *auto*

definition *occurrence-list* **where**
‹*occurrence-list* $\mathcal{A}$ = {((*n*, *ys*), *xs*). (*ys*, *xs*)∈*Id* ∧ *n* = (*set-mset* $\mathcal{A}$)}›

lemma *mop-occ-list-create*:
shows ‹*mop-occ-list-create* (*set-mset* $\mathcal{A}$) ≤ *SPEC* (λ*c*. (*c*, *raw-empty-occs-list*) ∈ *occurrence-list* $\mathcal{A}$)›
(**is** ‹*?A* ≤ *?B*›)
unfolding *mop-occ-list-create-def*
by *refine-vcg*
(*use* **in** ‹*auto simp*: *RETURN-RES-refine-iff occurrence-list-def*›)

lemma *mop-occ-list-at*:
assumes ‹*occ-list-at-pre occs L i*›
shows ‹*mop-occ-list-at occs L i* ≤ *SPEC* (λ*c*. (*c*, *occ-list-at occs L i*) ∈ *Id*)› (**is** ‹*?A* ≤ *?B*›)
using *assms* **unfolding** *mop-occ-list-at-def*
by *refine-vcg auto*

lemma *mop-occ-list-append*:
assumes ‹*occ-list-append-pre occs L*›
shows ‹*mop-occ-list-append x occs L* ≤ *SPEC* (λ*c*. (*c*, *occ-list-append x occs L*) ∈ *Id*)›
using *assms* **unfolding** *mop-occ-list-append-def*
by *refine-vcg auto*

abbreviation *occ-list-append-r* :: ‹*nat literal* ⇒ *nat* ⇒ (*nat literal* ⇒ *nat list*) ⇒ -› **where**
‹*occ-list-append-r L x xs* ≡ *xs* (*L* := *xs L* @ [*x*])›

9.1.2 Concrete Occurrence lists

We use *cocc* for concrete occurrence lists.

type-synonym *occurences-ref* = ‹*nat list list*›

abbreviation D_1 :: ‹*nat set* ⇒ (*nat* × *nat literal*) *set*› **where**
‹D_1 $\mathcal{A}_{in}$ ≡ (λ*L*. (*nat-of-lit L*, *L*)) ‘ (*Pos* ‘ $\mathcal{A}_{in}$ ∪ *Neg* ‘ $\mathcal{A}_{in}$)›

definition *occurrence-list-ref* :: ‹(*occurences-ref* × *occurences*) *set*› **where**

‹*occurrence-list-ref* ≡ {(*xs*, (*n*, *ys*)). (*xs*, *ys*)∈⟨⟨*nat-rel*⟩*list-rel*⟩*map-fun-rel* (*D*$_1$ *n*) ∧ (∀ *L*. *L* ∉ *fst* ‘ (*D*$_1$ *n*) ⟶ *L* < *length xs* ⟶ *xs* ! *L* = [])}›

lemma *empty-occs-list-cong′*:
‹*set-mset A* = *set-mset B* ⟹ (*occs*, *empty-occs-list A*) ∈ *occurrence-list-ref* ⟹ (*occs*, *empty-occs-list B*) ∈ *occurrence-list-ref*›
unfolding *empty-occs-list-def* **by** *auto*

definition *cocc-list-at* :: ‹*occurences-ref* ⇒ *nat literal* ⇒ *nat* ⇒ *nat*› **where**
‹*cocc-list-at xs L i* = *xs* ! *nat-of-lit L* ! *i*›

definition *cocc-list-at-pre* :: ‹*occurences-ref* ⇒ *nat literal* ⇒ *nat* ⇒ *bool*› **where**
‹*cocc-list-at-pre* = (λ*xs L i*. *i* < *length* (*xs* ! *nat-of-lit L*) ∧ *nat-of-lit L* < *length xs*)›

definition *mop-cocc-list-at* :: ‹*occurences-ref* ⇒ *nat literal* ⇒ *nat* ⇒ *nat nres*› **where**
‹*mop-cocc-list-at* = (λ*𝒜xs L i*. *do* {
ASSERT (*cocc-list-at-pre 𝒜xs L i*);
RETURN (*cocc-list-at 𝒜xs L i*)
})›

lemma $\mathcal{L}_{all}$*-add-mset*:
‹*set-mset* ($\mathcal{L}_{all}$ (*add-mset K A*)) = *set-mset* ($\mathcal{L}_{all}$ *A*) ∪ {*Pos K*, *Neg K*}›
by (*auto simp*: $\mathcal{L}_{all}$*-def*)

lemma *mop-cocc-list-at-mop-occ-list-at*:
assumes
‹(*xs*, *𝒜xs*) ∈ *occurrence-list-ref*›
‹(*L*,*L′*)∈*Id*›
‹(*i*,*i′*)∈*nat-rel*›
shows
‹*mop-cocc-list-at xs L i* ≤ ⇓{(*K*,*K′*). (*K*,*K′*)∈*nat-rel* ∧ *K* = *occ-list-at 𝒜xs L′ i* ∧ *K* = *cocc-list-at xs L′ i* ∧ *nat-of-lit L′* < *length xs* ∧ *i* < *length* (*xs* ! *nat-of-lit L*)} (*mop-occ-list-at 𝒜xs L′ i′*)›
using *assms* **unfolding** *mop-cocc-list-at-def mop-occ-list-at-def occurrence-list-ref-def*
apply *refine-rcg*
subgoal
by (*cases L*)
(*auto simp*: *cocc-list-at-pre-def occ-list-at-pre-def* $\mathcal{L}_{all}$*-add-mset*
map-fun-rel-def dest: *bspec*[*of* - - *L*])
subgoal
by (*cases L*)
(*auto simp*: *cocc-list-at-pre-def occ-list-at-pre-def* $\mathcal{L}_{all}$*-add-mset cocc-list-at-def occ-list-at-def*
map-fun-rel-def occ-list-def dest: *bspec*[*of* - - *L*])
done

definition *cocc-list-length-pre* :: ‹*occurences-ref* ⇒ *nat literal* ⇒ *bool*› **where**
‹*cocc-list-length-pre* = (λ(*xs*) *L*. *nat-of-lit L* < *length xs*)›

definition *cocc-list-length* :: ‹*occurences-ref* ⇒ *nat literal* ⇒ *nat*› **where**
‹*cocc-list-length* = (λ*xs L*. *do* {
(*length* (*xs* ! *nat-of-lit L*))
})›

definition *mop-cocc-list-length* :: ‹*occurences-ref* ⇒ *nat literal* ⇒ *nat nres*› **where**
‹*mop-cocc-list-length* = (λ$\mathcal{A}$*xs L. do* {
ASSERT (*cocc-list-length-pre* $\mathcal{A}$*xs L*);
RETURN (*cocc-list-length* $\mathcal{A}$*xs L*)
})›

lemma *mop-cocc-list-length-mop-occ-list-length*:
assumes
‹(*xs*, $\mathcal{A}$*xs*) ∈ *occurrence-list-ref*›
‹(*L*,*L'*)∈*Id*›
shows
‹*mop-cocc-list-length xs L* ≤ ⇓*nat-rel* (*mop-occ-list-length* $\mathcal{A}$*xs L'*)›
using *assms*
unfolding *mop-cocc-list-length-def mop-occ-list-length-def occurrence-list-ref-def*
apply *refine-vcg*
subgoal
by (*cases L*)
(*auto simp*: *occ-list-length-pre-def cocc-list-length-pre-def* $\mathcal{L}_{all}$*-add-mset*
map-fun-rel-def dest!: *multi-member-split*)
subgoal
by (*cases L*)
(*auto simp*: *occ-list-length-pre-def cocc-list-length-pre-def* $\mathcal{L}_{all}$*-add-mset*
map-fun-rel-def occ-list-length-def cocc-list-length-def dest: *bspec*[*of - - L*])
done

definition *cocc-list-append-pre* :: ‹*occurences-ref* ⇒ *nat literal* ⇒ *bool*› **where**
‹*cocc-list-append-pre* = (λ*xs L. nat-of-lit L* < *length xs*)›

definition *cocc-list-append* :: ‹*nat* ⇒ *occurences-ref* ⇒ *nat literal* ⇒ *occurences-ref*› **where**
‹*cocc-list-append* = (λ*x xs L. xs* [*nat-of-lit L* := *xs* ! (*nat-of-lit L*) @ [*x*]])›

definition *mop-cocc-list-append* :: ‹*nat* ⇒ *occurences-ref* ⇒ *nat literal* ⇒ *occurences-ref nres*› **where**
‹*mop-cocc-list-append* = (λ*x* $\mathcal{A}$*xs L . do* {
ASSERT (*cocc-list-append-pre* ($\mathcal{A}$*xs*) *L*);
RETURN (*cocc-list-append x* ($\mathcal{A}$*xs*) *L*)
})›

lemma *mop-cocc-list-append-mop-occ-list-append*:
assumes
‹(*xs*, $\mathcal{A}$*xs*) ∈ *occurrence-list-ref*›
‹(*L*,*L'*)∈*Id*› **and**
‹(*x*,*x'*)∈*nat-rel*›
shows
‹*mop-cocc-list-append x xs L* ≤ ⇓{(*a*,*b*). (*a*,*b*)∈*occurrence-list-ref* ∧ *a* = *cocc-list-append x xs L* ∧ *nat-of-lit L* < *length xs*} (*mop-occ-list-append x'* $\mathcal{A}$*xs L'*)›
using *assms*
unfolding *mop-cocc-list-append-def mop-occ-list-append-def occurrence-list-ref-def*
apply *refine-vcg*
subgoal
by (*cases L*)
(*auto simp*: *cocc-list-append-pre-def occ-list-append-pre-def* $\mathcal{L}_{all}$*-add-mset*
map-fun-rel-def dest!: *bspec*[*of - - L*])
subgoal
apply (*cases L*)
apply (*auto simp*: *cocc-list-append-pre-def occ-list-append-pre-def* $\mathcal{L}_{all}$*-add-mset*

```
      map-fun-rel-def cocc-list-append-def occ-list-append-def)
   apply (force simp add: image-image image-Un)+
   done
 done

definition mop-cocc-list-create :: ‹nat ⇒ occurences-ref nres› where
 ‹mop-cocc-list-create = (λn. do {
   RETURN (replicate n [])
 })›

lemma mop-cocc-list-create-mop-occ-list-create:
 assumes ‹n > 2* Max 𝒜 + 1› ‹finite 𝒜›
 shows ‹mop-cocc-list-create n ≤ ⇓(occurrence-list-ref) (mop-occ-list-create 𝒜)›
 unfolding mop-cocc-list-create-def mop-occ-list-create-def occurrence-list-ref-def
 using assms
 by (auto simp: map-fun-rel-def Max-gr-iff dest: Max-ge)

definition cocc-list-clear-at-pre :: ‹occurences-ref ⇒ nat literal ⇒ bool› where
 ‹cocc-list-clear-at-pre = (λ(xs) L. nat-of-lit L < length xs)›

definition cocc-list-clear-at :: ‹occurences-ref ⇒ nat literal ⇒ occurences-ref nres› where
 ‹cocc-list-clear-at = (λxs L . do {
   ASSERT (cocc-list-clear-at-pre xs L);
   RETURN (xs [nat-of-lit L := []])
 })›

lemma cocc-list-clear-at-occ-list-clear-at:
 assumes
   ‹(xs, 𝒜xs) ∈ occurrence-list-ref›
   ‹(L,L')∈Id›
 shows
   ‹cocc-list-clear-at xs L ≤ ⇓(occurrence-list-ref) (occ-list-clear-at 𝒜xs L')›
 using assms
 unfolding cocc-list-clear-at-def occ-list-clear-at-def case-prod-beta occurrence-list-ref-def
 apply refine-vcg
 subgoal
   by (cases L)
    (auto simp: cocc-list-clear-at-pre-def occ-list-clear-at-pre-def ℒ_all-add-mset
     map-fun-rel-def  dest: bspec[of - - L])
 subgoal
   by (cases L)
    (auto simp: cocc-list-append-pre-def occ-list-append-pre-def ℒ_all-add-mset
       map-fun-rel-def cocc-list-append-def occ-list-append-def; force)+
 done

definition cocc-list-clear-all-pre :: ‹occurences-ref ⇒ bool› where
 ‹cocc-list-clear-all-pre = (λxs. True)›

definition cocc-list-clear-all :: ‹occurences-ref ⇒ occurences-ref nres› where
 ‹cocc-list-clear-all = (λ(xs) . do {
   ASSERT (cocc-list-clear-all-pre (xs));
   RETURN (replicate (length xs) [])
 })›


lemma cocc-list-clear-all-occ-list-clear-all:
```

assumes
‹$(xs, \mathcal{A}xs) \in$ *occurrence-list-ref*›
shows
‹*cocc-list-clear-all xs* $\leq \Downarrow$(*occurrence-list-ref*) (*occ-list-clear-all* $\mathcal{A}xs$)›
using *assms*
unfolding *cocc-list-clear-all-def occ-list-clear-all-def case-prod-beta occurrence-list-ref-def*
apply *refine-vcg*
subgoal
by
(*auto simp*: *cocc-list-clear-all-pre-def occ-list-clear-all-pre-def* $\mathcal{L}_{all}$*-add-mset*
map-fun-rel-def dest: *bspec*[*of - - L*])
subgoal
by (*auto simp*: *cocc-list-append-pre-def occ-list-append-pre-def* $\mathcal{L}_{all}$*-add-mset*
map-fun-rel-def cocc-list-append-def occ-list-append-def; *force*)+
done

9.2 Clause Marking

Experiment: This should eventually replace the stuff used for the conflicts. Experiment in the current state to see if useful. If it is this should be generalized. However, not clear because distinct, so not really multisets.

Experiment: is keeping the set of variables as sets useful? is the refinement from there on okay (both from doing it and also performance wise)

9.2.1 Abstract Representation

type-synonym *clause-hash* = ‹(*nat set* × *nat clause*)›
definition *clause-hash-ref* **where**
‹*clause-hash-ref* $\mathcal{A}$ = {(($\mathcal{B}$, *C*), *D*). *C* = *D* ∧ *set-mset* $\mathcal{A}$ = $\mathcal{B}$ ∧ *atms-of C* ⊆ $\mathcal{B}$}›

definition *ch-create-pre* :: ‹*nat set* ⇒ *bool*› **where**
‹*ch-create-pre n* = (*True*)›

The fact that the nat set must be passed as argument is really ugly

definition *mop-ch-create* :: ‹*nat set* ⇒ *clause-hash nres*› **where**
‹*mop-ch-create n* = *do* {
ASSERT (*ch-create-pre n*);
RETURN (*n*, {#})
}›

definition *ch-add* :: ‹*nat literal* ⇒ *clause-hash* ⇒ *clause-hash*› **where**
‹*ch-add L* = (λ($\mathcal{A}$, *C*). ($\mathcal{A}$, *add-mset L C*))›

definition *ch-add-pre* :: ‹*nat literal* ⇒ *clause-hash* ⇒ *bool*› **where**
‹*ch-add-pre L* = (λ($\mathcal{A}$, *C*). *atm-of L* ∈ $\mathcal{A}$ ∧ *L* ∉# *C*)›

definition *mop-ch-add* :: ‹*nat literal* ⇒ *clause-hash* ⇒ *clause-hash nres*› **where**
‹*mop-ch-add L C* = *do* {
ASSERT (*ch-add-pre L C*);
RETURN (*ch-add L C*)
}›

definition *ch-remove* :: ‹*nat literal* ⇒ *clause-hash* ⇒ *clause-hash*› **where**

‹*ch-remove* L = ($\lambda(\mathcal{A}, C)$. ($\mathcal{A}$, *remove1-mset* L C))›

definition *ch-remove-pre* :: ‹*nat literal* ⇒ *clause-hash* ⇒ *bool*› **where**
‹*ch-remove-pre* L = ($\lambda(\mathcal{A}, C)$. *atm-of* $L \in \mathcal{A} \wedge$ $L \in\#$ C)›

definition *mop-ch-remove* :: ‹*nat literal* ⇒ *clause-hash* ⇒ *clause-hash nres*› **where**
‹*mop-ch-remove* L C = *do* {
ASSERT (*ch-remove-pre* L C);
RETURN (*ch-remove* L C)
}›

definition *ch-in* :: ‹*nat literal* ⇒ *clause-hash* ⇒ *bool*› **where**
‹*ch-in* L = ($\lambda(\mathcal{A}, C)$. $L \in\#$ C)›

definition *ch-in-pre* :: ‹*nat literal* ⇒ *clause-hash* ⇒ *bool*› **where**
‹*ch-in-pre* L = ($\lambda(\mathcal{A}, C)$. *atm-of* $L \in \mathcal{A}$)›

definition *mop-ch-in* :: ‹*nat literal* ⇒ *clause-hash* ⇒ *bool nres*› **where**
‹*mop-ch-in* L C = *do* {
ASSERT (*ch-in-pre* L C);
RETURN (*ch-in* L C)
}›

definition *ch-remove-clause* :: ‹*nat clause* ⇒ *clause-hash* ⇒ *clause-hash*› **where**
‹*ch-remove-clause* D = ($\lambda(\mathcal{A}, C)$. ($\mathcal{A}$, $C - D$))›

definition *ch-remove-clause-pre* :: ‹*nat clause* ⇒ *clause-hash* ⇒ *bool*› **where**
‹*ch-remove-clause-pre* D = ($\lambda(\mathcal{A}, C)$. $D \subseteq\#$ C)›

definition *mop-ch-remove-clause* :: ‹*nat clause* ⇒ *clause-hash* ⇒ *clause-hash nres*› **where**
‹*mop-ch-remove-clause* L C = *do* {
ASSERT (*ch-remove-clause-pre* L C);
RETURN (*ch-remove-clause* L C)
}›

definition *ch-remove-all* :: ‹*clause-hash* ⇒ *clause-hash*› **where**
‹*ch-remove-all* = ($\lambda(\mathcal{A}, C)$. ($\mathcal{A}$, {#}))›

definition *mop-ch-remove-all* :: ‹*nat clause* ⇒ *clause-hash* ⇒ *clause-hash nres*› **where**
‹*mop-ch-remove-all* D C = *do* {
ASSERT (D = *snd* $C \wedge$ *atm-of* ‘ (*set-mset* D) ⊆ *fst* C);
RETURN (*ch-remove-all* C)
}›

definition *ch-add-all* :: ‹*nat clause* ⇒ *clause-hash* ⇒ *clause-hash*› **where**
‹*ch-add-all* D = ($\lambda(\mathcal{A}, C)$. ($\mathcal{A}$, $C + D$))›

definition *ch-add-all-pre* :: ‹*nat clause* ⇒ *clause-hash* ⇒ *bool*› **where**
‹*ch-add-all-pre* D = ($\lambda(\mathcal{A}, C)$. *atms-of* $D \subseteq \mathcal{A} \wedge C \cap\# D = \{\#\} \wedge$ *distinct-mset* D)›

definition *mop-ch-add-all* :: ‹*nat clause* ⇒ *clause-hash* ⇒ *clause-hash nres*› **where**
‹*mop-ch-add-all* L C = *do* {
ASSERT (*ch-add-all-pre* L C);

RETURN (*ch-add-all L C*)
}›

lemma *mop-ch-create*:
shows ‹*mop-ch-create* (*set-mset* $\mathcal{A}$) ≤ *SPEC* (λ*c*. (*c*, {#}) ∈ *clause-hash-ref* $\mathcal{A}$)› (**is** ‹*?A* ≤ *?B*›)
unfolding *mop-ch-create-def*
by *refine-vcg*
(*auto simp*: *clause-hash-ref-def ch-create-pre-def RETURN-RES-refine-iff*)

lemma *mop-ch-add*:
assumes ‹(*C*, *D*) ∈ *clause-hash-ref* $\mathcal{A}$› **and** ‹*atm-of L* ∈# $\mathcal{A}$› **and** ‹(*L*,*L'*)∈*Id*› ‹*L*∉#*D*›
shows ‹*mop-ch-add L C* ≤ *SPEC*(λ*c*. (*c*, *add-mset L' D*) ∈ *clause-hash-ref* $\mathcal{A}$)›
using *assms* **unfolding** *mop-ch-add-def*
apply *refine-vcg*
subgoal unfolding *ch-add-pre-def* **by** (*auto simp*: *clause-hash-ref-def*)
subgoal by (*auto simp*: *clause-hash-ref-def ch-add-def*)
done

lemma *mop-ch-add-all*:
assumes ‹(*C*, *D*) ∈ *clause-hash-ref* $\mathcal{A}$› **and** ‹*atms-of L* ⊆ *set-mset* $\mathcal{A}$› **and** ‹(*L*,*L'*)∈*Id*› ‹*D* ∩# *L'* = {#}› **and**
‹*distinct-mset L'*›
shows ‹*mop-ch-add-all L C* ≤ *SPEC*(λ*c*. (*c*, *L'* + *D*) ∈ *clause-hash-ref* $\mathcal{A}$)›
using *assms* **unfolding** *mop-ch-add-all-def*
apply *refine-vcg*
subgoal unfolding *ch-add-all-pre-def* **by** (*auto simp*: *clause-hash-ref-def*)
subgoal by (*auto simp*: *clause-hash-ref-def ch-add-all-def*)
done

lemma *mop-ch-in*:
assumes ‹(*C*, *D*) ∈ *clause-hash-ref* $\mathcal{A}$› **and** ‹*atm-of L* ∈# $\mathcal{A}$› **and** ‹(*L*,*L'*)∈*Id*›
shows ‹*mop-ch-in L C* ≤ *SPEC*(λ*c*. (*c*, *L'* ∈# *D*) ∈ *bool-rel*)›
using *assms* **unfolding** *mop-ch-in-def*
apply *refine-vcg*
subgoal unfolding *ch-in-pre-def* **by** (*auto simp*: *clause-hash-ref-def*)
subgoal by (*auto simp*: *clause-hash-ref-def ch-in-def*)
done

lemma *mop-ch-remove*:
assumes ‹(*C*, *D*) ∈ *clause-hash-ref* $\mathcal{A}$› **and** ‹*atm-of L* ∈# $\mathcal{A}$› **and** ‹(*L*,*L'*)∈*Id*› ‹*L*∈#*D*›
shows ‹*mop-ch-remove L C* ≤ *SPEC*(λ*c*. (*c*, *remove1-mset L' D*) ∈ *clause-hash-ref* $\mathcal{A}$)›
using *assms* **unfolding** *mop-ch-remove-def*
apply *refine-vcg*
subgoal unfolding *ch-remove-pre-def* **by** (*auto simp*: *clause-hash-ref-def*)
subgoal by (*auto simp*: *clause-hash-ref-def ch-remove-def dest*: *in-atms-of-minusD*)
done

lemma *mop-ch-remove-all*:
assumes ‹(*C*, *D*) ∈ *clause-hash-ref* $\mathcal{A}$› ‹*atm-of* ‘ *set-mset D* ⊆ *set-mset* $\mathcal{A}$›
shows ‹*mop-ch-remove-all D C* ≤ *SPEC*(λ*c*. (*c*, {#}) ∈ *clause-hash-ref* $\mathcal{A}$)›
using *assms* **unfolding** *mop-ch-remove-all-def*
apply *refine-vcg*
subgoal by (*auto simp*: *clause-hash-ref-def ch-remove-all-def*)
subgoal by (*auto simp*: *clause-hash-ref-def ch-remove-all-def*)
subgoal by (*auto simp*: *clause-hash-ref-def ch-remove-all-def*)
done

9.2.2 Concrete Representation

TODO: The mark structure should probably be replaced by our abstract ch-stuff

TODO: the alternative version consists in keeping the multiset, but replacing the atoms by the upper bound. This makes it possible to keep the abstraction (kind of). However, it is very clear what would be the difference.

definition *clause-hash* :: ‹(*bool list* × *clause-hash*) *set*› **where**
‹*clause-hash* = {(*xs*, (*A*, *C*)). (∀ *L* ∈ *snd* ‘ D_1 *A*. *xs* ! *nat-of-lit L* ⟷ *L* ∈# *C*) ∧
(∀ *L*∈*fst* ‘ D_1 *A*. *L* < *length xs*) ∧ *distinct-mset C*}›

definition *mop-cch-create* :: ‹*nat* ⇒ *bool list nres*› **where**
‹*mop-cch-create n* = *do* {
RETURN (*replicate n False*)
}›

lemma *mop-cch-create-mop-cch-create*:
assumes ‹∀ *L*∈*fst* ‘ D_1 *A*. *L* < *n*›
shows ‹*mop-cch-create n* ≤ ⇓*clause-hash* (*mop-ch-create A*)›
using *assms*
unfolding *mop-cch-create-def mop-ch-create-def*
by *refine-vcg*
(*auto simp*: *clause-hash-def mset-as-position.intros*)

definition *cch-add* :: ‹*nat literal* ⇒ *bool list* ⇒ *bool list*› **where**
‹*cch-add L* = (λ*C*. *C* [*nat-of-lit L* := *True*])›

definition *cch-add-pre* :: ‹*nat literal* ⇒ *bool list* ⇒ *bool*› **where**
‹*cch-add-pre L* = (λ*C*. *nat-of-lit L* < *length C*)›

definition *mop-cch-add* :: ‹*nat literal* ⇒ *bool list* ⇒ *bool list nres*› **where**
‹*mop-cch-add L C* = *do* {
ASSERT (*cch-add-pre L C*);
RETURN (*cch-add L C*)
}›

lemma *mop-cch-add-mop-cch-add*:
assumes ‹(*C*, *D*) ∈ *clause-hash*› **and**
‹(*L*,*L*′)∈*Id*›
shows ‹*mop-cch-add L C* ≤ ⇓ *clause-hash* (*mop-ch-add L*′ *D*)›
proof −
have [*iff*]: ‹2 ∗ *x1* = *Suc* (2 ∗ *xa*) ⟷ *False*› ‹*Suc* (2 ∗ *x1*) = 2 ∗ *xa* ⟷ *False*› **for** *xa x1* :: *nat*
by *presburger*+
show *?thesis*
using *assms* **unfolding** *mop-cch-add-def mop-ch-add-def clause-hash-def*
apply *refine-rcg*
subgoal by (*cases L*) (*auto simp*: *ch-add-pre-def cch-add-pre-def dest*: *bspec*)
subgoal
by (*cases L*′)
(*auto simp*: *cch-add-def ch-add-def cch-add-pre-def ch-add-pre-def intro*!: *mset-as-position.intros*
dest: *bspec*[*of* - - *L*′])
done
qed

definition *cch-remove* :: ‹*nat literal* ⇒ *bool list* ⇒ *bool list*› **where**
‹*cch-remove L* = (λ*C*. *C*[*nat-of-lit L* := *False*])›

definition *cch-remove-pre* :: ‹*nat literal* ⇒ *bool list* ⇒ *bool*› **where**
‹*cch-remove-pre L* = (λ*C*. *nat-of-lit L* < *length C*)›

definition *mop-cch-remove* :: ‹*nat literal* ⇒ *bool list* ⇒ *bool list nres*› **where**
‹*mop-cch-remove L C* = *do* {
ASSERT (*cch-remove-pre L C*);
RETURN (*cch-remove L C*)
}›

lemma *mop-cch-remove-mop-ch-remove*:
assumes ‹(*C*, *D*) ∈ *clause-hash*› **and**
‹(*L*,*L*′)∈*Id*›
shows ‹*mop-cch-remove L C* ≤ ⇓ *clause-hash* (*mop-ch-remove L*′ *D*)›
proof −
have [*iff*]: ‹2 ∗ *x1* = *Suc* (2 ∗ *xa*) ⟷ *False*› ‹*Suc* (2 ∗ *x1*) = 2 ∗ *xa* ⟷ *False*› **for** *xa x1* :: *nat*
by *presburger*+
show *?thesis*
using *assms* **unfolding** *mop-cch-remove-def mop-ch-remove-def clause-hash-def*
apply *refine-rcg*
subgoal by (*cases L*) (*auto simp*: *ch-remove-pre-def cch-remove-pre-def dest*: *bspec*)
subgoal
by (*cases L*′)
(*auto simp*: *cch-remove-def ch-remove-def cch-remove-pre-def ch-remove-pre-def distinct-mset-remove1-All*
intro!: *mset-as-position.intros*
dest: *bspec*[*of - - L*′])
done
qed

definition *cch-in* :: ‹*nat literal* ⇒ *bool list* ⇒ *bool*› **where**
‹*cch-in L* = (λ*C*. *C* ! *nat-of-lit L*)›

definition *cch-in-pre* :: ‹*nat literal* ⇒ *bool list* ⇒ *bool*› **where**
‹*cch-in-pre L* = (λ*C*. *nat-of-lit L* < *length C*)›

definition *mop-cch-in* :: ‹*nat literal* ⇒ *bool list* ⇒ *bool nres*› **where**
‹*mop-cch-in L C* = *do* {
ASSERT (*cch-in-pre L C*);
RETURN (*cch-in L C*)
}›

lemma *mop-cch-in-mop-ch-in*:
assumes ‹(*C*, *D*) ∈ *clause-hash*› **and**
‹(*L*,*L*′)∈*Id*›
shows ‹*mop-cch-in L C* ≤ ⇓ *bool-rel* (*mop-ch-in L*′ *D*)›
proof −
have [*iff*]: ‹2 ∗ *x1* = *Suc* (2 ∗ *xa*) ⟷ *False*› ‹*Suc* (2 ∗ *x1*) = 2 ∗ *xa* ⟷ *False*› **for** *xa x1* :: *nat*
by *presburger*+
show *?thesis*
using *assms* **unfolding** *mop-cch-in-def mop-ch-in-def clause-hash-def*
apply *refine-rcg*
subgoal by (*cases L*) (*auto simp*: *ch-in-pre-def cch-in-pre-def dest*: *bspec*)
subgoal
by (*cases L*′)

(*auto simp*: *cch-in-def ch-in-def cch-in-pre-def ch-in-pre-def*
intro!: *mset-as-position.intros*
dest: *bspec*[*of* - - *L'*])
done
qed

definition *mop-cch-remove-all* :: ‹*nat clause-l* ⇒ *bool list* ⇒ *bool list nres*› **where**
‹*mop-cch-remove-all C D* = *do* {
(-, *D*) ← *WHILE*$_T$ (λ(*i*, *D*). *i* < *length C*)
(λ(*i*, *D*). *RETURN* (*i*+*1*, *D*[*nat-of-lit* (*C*!*i*) := *False*]))
(*0*, *D*);
RETURN D
}›

abbreviation *cocc-content* :: ‹*nat list list* ⇒ *nat multiset*› **where**
‹*cocc-content coccs* ≡ *sum-list* (*map mset coccs*)›

definition *cocc-content-set* :: ‹*nat list list* ⇒ *nat set*› **where**
‹*cocc-content-set coccs* ≡ ⋃(*image set* (*set coccs*))›

lemma *sum-list-update-mset*:
k < *size xs* ⟹ *sum-list* (*xs*[*k* := *x*]) = *sum-list xs* + *x* − *xs* ! *k* **for** *xs* :: ‹*'a multiset list*›
unfolding *sum-mset-sum-list*[*symmetric*]
apply (*subst mset-update*, *assumption*)
apply (*auto simp*: *cancel-comm-monoid-add-class.sum-mset-diff ac-simps*)
done

lemma *sum-list-cocc-list-append*[*simp*]: ‹*nat-of-lit La* < *length coccs* ⟹ *sum-list* (*map mset* (*cocc-list-append C coccs La*)) = *add-mset C* (*sum-list* (*map mset coccs*))›
apply (*auto simp*: *cocc-list-append-def map-update sum-list-update sum-list-update-mset*)
done

lemma *cocc-content-set-append*[*simp*]:
‹*nat-of-lit La* < *length coccs* ⟹ *cocc-content-set* (*cocc-list-append C coccs La*) = *insert C* (*cocc-content-set coccs*)›
apply (*simp only*: *cocc-content-set-def cocc-list-append-def in-set-upd-eq*)
apply *auto*
apply (*smt* (*verit*, *ccfv-threshold*) *in-set-conv-nth in-set-upd-cases length-Suc-rev-conv nat-neq-iff not-less-eq nth-append-first nth-append-length nth-mem*)
unfolding *Bex-def*
apply (*subst in-set-upd-eq*, *simp*)
apply (*metis in-set-conv-nth length-Suc-rev-conv nat-in-between-eq*(*1*) *nth-append-length*)
by (*smt* (*verit*, *best*) *in-set-conv-nth le-imp-less-Suc length-Suc-rev-conv length-list-update less-imp-le-nat list-update-append1 list-update-id nth-list-update-neq set-update-memI*)

lemma *all-occurrences-add-mset*: ‹*all-occurrences* (*add-mset* (*atm-of L*) *A*) *occs* =
all-occurrences (*removeAll-mset* (*atm-of L*) *A*) *occs* + *mset* (*occ-list occs L*) + *mset* (*occ-list occs* (−*L*))›
by (*cases L*; *cases occs*)
(*auto simp*: *all-occurrences-def occ-list-def remdups-mset-removeAll*
remdups-mset-singleton-removeAll
removeAll-notin simp del: *remdups-mset-singleton-sum*)

lemma *all-occurrences-add-mset2*: ‹*all-occurrences* (*add-mset* (*L*) *A*) *occs* =
all-occurrences (*removeAll-mset* (*L*) *A*) *occs* + *mset* (*occ-list occs* (*Pos L*)) + *mset* (*occ-list occs* (*Neg L*))›
by (*cases occs*)
(*auto simp*: *all-occurrences-def occ-list-def remdups-mset-removeAll*
remdups-mset-singleton-removeAll
removeAll-notin simp del: *remdups-mset-singleton-sum*)

lemma *all-occurrences-insert-lit*:
‹*all-occurrences A* (*insert* (*atm-of L*) *B*, *occs*) = *all-occurrences* (*A*) (*B*, *occs*)› **and**
all-occurrences-occ-list-append-r:
‹*all-occurrences* (*removeAll-mset* (*atm-of L*) *A*) (*B*, *occ-list-append-r L C b*) =
all-occurrences (*removeAll-mset* (*atm-of L*) *A*) (*B*, *b*)›
apply (*auto simp*: *all-occurrences-def*)
by (*smt* (*verit*) *distinct-mset-remdups-mset distinct-mset-remove1-All image-mset-cong2*
literal.sel(*1*) *literal.sel*(*2*) *remdups-mset-removeAll removeAll-subseteq-remove1-mset*
subset-mset-removeAll-iff)

end
theory *IsaSAT-ACIDS*
imports *IsaSAT-Literals*
Pairing-Heap-LLVM.Heaps-Abs
Watched-Literals-VMTF
begin

Instead of using VSIDS (which requires float), we use the more stable ACIDS variant that works simply on integers and does not seem much worse.

We use ACIDS in a practical way, i.e., when the weight reaches the maximum integer, we simply stop incrementing it.

9.3 ACIDS

type-synonym ('*a*, '*v*) *acids* = ‹('*a multiset* × '*a multiset* × ('*a* ⇒ '*v*)) × '*v*›
definition *acids* :: ‹'*a multiset* ⇒ ('*a*, '*ann*) *ann-lits* ⇒ ('*a*, '*v*::{*zero*,*linorder*}) *acids set*› **where**
‹*acids* $\mathcal{A}$ *M* = {(($\mathcal{B}$, *b*, *w*), *m*). *set-mset* $\mathcal{B}$ = *set-mset* $\mathcal{A}$ ∧ *b* ⊆# $\mathcal{A}$ ∧ *Max* ({*0*} ∪ *w* ‘ *set-mset b*) ≤ *m* ∧ (∀ *L* ∈#$\mathcal{A}$. *L* ∉# *b* ⟶ *defined-lit M* (*Pos L*)) ∧ *distinct-mset b*}›

lemma *acids-prepend*: ‹*ac* ∈ *acids* $\mathcal{A}$ *M* ⟹ *ac* ∈ *acids* $\mathcal{A}$ (*L* # *M*)›
unfolding *acids-def* **by** (*auto simp*: *defined-lit-map*)

interpretation *ACIDS*: *hmstruct-with-prio* **where**
le = ‹(≥) :: *nat* ⇒ *nat* ⇒ *bool*› **and**
lt = ‹(>)›
apply *unfold-locales*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *transp-def*)
subgoal by (*auto simp*: *totalp-on-def*)
done

definition *acids-tl-pre* :: ‹'*a* ⇒ ('*a*, '*v*) *acids* ⇒ *bool*› **where**
‹*acids-tl-pre L* = (λ(*ac*, *m*). *L* ∈# *fst ac*)›

definition *acids-tl* :: ‹'*a* ⇒ ('*a*, '*v*::*ord*) *acids* ⇒ ('*a*, '*v*) *acids nres*› **where**
‹*acids-tl L* = (λ(*ac*, *m*). *do* {

```
    ASSERT (acids-tl-pre L (ac, m));
    ac ← ACIDS.mop-prio-insert-unchanged L ac;
    w ← ACIDS.mop-prio-old-weight L ac;
    RETURN (ac, max m w)
  })›

lemma acids-tl:
  ‹ac ∈ acids A M ⟹ L ∈# A ⟹ M ≠ [] ⟹ L = atm-of (lit-of (hd M)) ⟹ acids-tl L ac ≤ RES
(acids A (tl M))›
  unfolding acids-tl-def ACIDS.mop-prio-insert-unchanged-def
    ACIDS.mop-prio-insert-raw-unchanged-def nres-monad3
    ACIDS.mop-prio-is-in-def
    ACIDS.mop-prio-old-weight-def
    ACIDS.mop-prio-insert-def RES-RES-RETURN-RES RETURN-def
    ACIDS.mop-prio-old-weight-def case-prod-beta nres-monad1
  apply (refine-vcg lhs-step-If)
  subgoal
    by (cases M) (auto simp: acids-def ACIDS.mop-prio-insert-unchanged-def insert-subset-eq-iff
        acids-tl-pre-def
      intro!: subset-add-mset-notin-subset)
  subgoal
    by (auto simp: acids-def ACIDS.mop-prio-insert-unchanged-def insert-subset-eq-iff
        acids-tl-pre-def
      intro!: subset-add-mset-notin-subset)
  subgoal
    apply (auto simp: acids-def ACIDS.mop-prio-insert-unchanged-def RES-RES-RETURN-RES
      defined-lit-map acids-tl-pre-def dest: subset-add-mset-notin-subset
      dest: multi-member-split)
    apply (smt (verit, best) image-iff not-hd-in-tl)
    apply refine-vcg
    apply fastforce
    apply fastforce
    apply (smt (verit, del-insts) Union-iff imageE insert-DiffM insert-subset-eq-iff prod.simps(1)
      singletonD subset-add-mset-notin-subset-mset)
    apply (auto simp: max-def)
    apply fastforce
    by (smt (verit, best) image-iff not-hd-in-tl)
  done

definition acids-get-min :: ‹('a, nat) acids ⇒ 'a nres› where
  ‹acids-get-min = (λ(ac, m). do {
    L ← ACIDS.mop-prio-peek-min ac;
    RETURN L
  })›

definition acids-mset :: ‹('a, 'v) acids ⇒ -› where
  ‹acids-mset x = fst (snd (fst x))›

lemma acids-get-min:
  ‹acids-mset x ≠ {#} ⟹ acids-get-min x ≤ SPEC (λv. ACIDS.prio-peek-min (fst x) v)›
  unfolding acids-get-min-def ACIDS.mop-prio-peek-min-def acids-mset-def
  by refine-vcg (auto simp: ACIDS.prio-peek-min-def)

definition acids-pop-min :: ‹('a, nat) acids ⇒ ('a × ('a, nat) acids) nres› where
  ‹acids-pop-min = (λ(ac, m). do {
    (v, ac) ← ACIDS.mop-prio-pop-min ac;
```

```
    RETURN (v, (ac, m))
  })›

definition acids-find-next-undef :: ‹nat multiset ⇒ (nat, nat) acids ⇒ (nat, nat) ann-lits ⇒ (nat option
× (nat, nat) acids) nres› where
‹acids-find-next-undef 𝒜 = (λac M. do {
  WHILE_T(λ(L, ac).       (L = None ⟶ ac ∈ acids 𝒜 M) ∧       (L ≠ None ⟶ ac ∈ acids 𝒜 (Decided (Pos (the L)) # M)
      (λ(nxt, ac). nxt = None ∧ acids-mset ac ≠ {#})
      (λ(a, ac). do {
        ASSERT (a = None);
        (L, ac) ← acids-pop-min ac;
        ASSERT(Pos L ∈# ℒ_all 𝒜);
        if defined-lit M (Pos L) then RETURN (None, ac)
        else RETURN (Some L, ac)
       }
      )
      (None, ac)
  })›

lemma acids-pop-min:
  ‹acids-mset x ≠ {#} ⟹ x ∈ acids 𝒜 M ⟹
  acids-pop-min x ≤ SPEC (λ(v, ac). acids-mset ac = acids-mset x - {#v#} ∧ v ∈# acids-mset x ∧
    ACIDS.prio-peek-min (fst x) v ∧
    (defined-lit M (Pos v) ⟶ ac ∈ acids 𝒜 M) ∧
    (undefined-lit M (Pos v) ⟶ ac ∈ acids 𝒜 (Decided (Pos v) # M)))›
  unfolding ACIDS.mop-prio-pop-min-def acids-pop-min-def
    ACIDS.mop-prio-peek-min-def ACIDS.mop-prio-del-def
  by refine-vcg
   (auto simp: acids-def ACIDS.prio-peek-min-def distinct-mset-remove1-All ACIDS.prio-del-def
      defined-lit-map acids-mset-def dest: in-diffD)

lemma acids-find-next-undef:
  assumes
    vmtf: ‹ac ∈ acids 𝒜 M›
  shows ‹acids-find-next-undef 𝒜 ac M
     ≤ ⇓ Id (SPEC (λ(L, ac).
        (L = None ⟶ ac ∈ acids 𝒜 M ∧ (∀ L∈#ℒ_all 𝒜. defined-lit M L)) ∧
      (L ≠ None ⟶ ac ∈ acids 𝒜 (Decided (Pos (the L)) # M) ∧ Pos (the L) ∈# ℒ_all 𝒜 ∧ undefined-lit
M (Pos (the L)))))›
proof −
  have [refine0]: ‹wf (measure (λ(-, ac). size (acids-mset ac)))›
    by auto
  show ?thesis
    unfolding acids-find-next-undef-def
    apply (refine-vcg acids-pop-min[of - 𝒜 M, THEN order-trans])
    subgoal using assms by auto
    subgoal by auto
    subgoal by (auto simp: ACIDS.prio-peek-min-def acids-def acids-mset-def
      in-ℒ_all-atm-of-𝒜_in)
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by (auto simp: acids-def ACIDS.prio-peek-min-def
      in-ℒ_all-atm-of-𝒜_in)
    subgoal by auto
```

subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *ACIDS.prio-peek-min-def acids-mset-def dest*: *multi-member-split*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *ACIDS.prio-peek-min-def acids-mset-def dest*: *multi-member-split*)
subgoal by *auto*
subgoal by (*auto simp*: *ACIDS.prio-peek-min-def acids-mset-def acids-def*
in-$\mathcal{L}_{all}$-atm-of-$\mathcal{A}_{in}$ dest!: *multi-member-split*[*of* ‹- :: *nat*›])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

definition *acids-push-literal-pre* **where**
‹*acids-push-literal-pre* $\mathcal{A}$ *L* = (λ*ac*. *L* $\in\#$ $\mathcal{A}$)›

definition *acids-push-literal* :: ‹'*a* ⇒ ('*a*, *nat*) *acids* ⇒ ('*a*, *nat*) *acids nres*› **where**
‹*acids-push-literal L* = (λ(*ac*, *m*). *do* {
ASSERT (*L* $\in\#$ *fst ac*);
w ← *ACIDS.mop-prio-old-weight L ac*;
let w = *min m w*;
ASSERT (*w* ≤ *m*);
ASSERT ((*m* − *w*) *div* 2 ≤ *m*);
let w = *m* − ((*m* − *w*) *div* 2);
ac ← *ACIDS.mop-prio-insert-maybe L w ac*;
RETURN (*ac*, *m*)
})›

lemma *acids-push-literal*:
‹*ac* ∈ *acids* $\mathcal{A}$ *M* ⟹ *acids-push-literal-pre* $\mathcal{A}$ *L ac* ⟹ *acids-push-literal L ac* ≤ *SPEC* (λ*ac*. *ac* ∈ *acids* $\mathcal{A}$ *M*)›
unfolding *acids-push-literal-def ACIDS.mop-prio-insert-maybe-def*
ACIDS.mop-prio-old-weight-def acids-push-literal-pre-def
ACIDS.mop-prio-insert-def ACIDS.mop-prio-change-weight-def
ACIDS.mop-prio-is-in-def
apply *refine-vcg*
subgoal by (*auto simp*: *acids-def acids-mset-def*)
subgoal by (*auto simp*: *acids-def dest*!: *multi-member-split*)
subgoal by (*auto simp*: *ACIDS.mop-prio-change-weight-def acids-def*
dest!: *multi-member-split*)
subgoal by (*auto simp*: *acids-def dest*!: *multi-member-split*)
subgoal by (*auto simp*: *acids-def acids-mset-def*)
subgoal by (*auto simp*: *acids-def acids-mset-def dest*!: *multi-member-split*
dest: *subset-add-mset-notin-subset*)
subgoal by (*auto simp*: *acids-def acids-mset-def dest*!: *multi-member-split*
dest: *subset-add-mset-notin-subset*)
subgoal by (*auto simp*: *acids-def acids-mset-def dest*!: *multi-member-split*
dest: *subset-add-mset-notin-subset*)
subgoal by (*auto simp*: *acids-def acids-mset-def dest*!: *multi-member-split*
dest: *subset-add-mset-notin-subset*)
subgoal by (*auto simp*: *acids-def acids-mset-def dest*!: *multi-member-split*

dest: *subset-add-mset-notin-subset*)
subgoal by (*auto simp*: *acids-def acids-mset-def dest*!: *multi-member-split*
dest: *subset-add-mset-notin-subset*)
subgoal by (*auto simp*: *acids-def acids-mset-def dest*!: *multi-member-split*
dest: *subset-add-mset-notin-subset*)
subgoal by (*auto simp*: *acids-def acids-mset-def dest*!: *multi-member-split*
dest: *subset-add-mset-notin-subset*)
subgoal by (*auto simp*: *acids-def acids-mset-def dest*!: *multi-member-split*
dest: *subset-add-mset-notin-subset*)
subgoal by (*auto simp*: *acids-def acids-mset-def dest*!: *multi-member-split*
dest: *subset-add-mset-notin-subset*)
subgoal by (*auto simp*: *acids-def acids-mset-def dest*!: *multi-member-split*
dest: *subset-add-mset-notin-subset*)
subgoal by (*auto simp*: *acids-def acids-mset-def dest*!: *multi-member-split*
dest: *subset-add-mset-notin-subset*)
subgoal by (*auto simp*: *acids-def acids-mset-def dest*!: *multi-member-split*
dest: *subset-add-mset-notin-subset*)
subgoal by (*auto simp*: *acids-def acids-mset-def dest*!: *multi-member-split*
dest: *subset-add-mset-notin-subset*)
subgoal by (*auto simp*: *acids-def acids-mset-def dest*!: *multi-member-split*
dest: *subset-add-mset-notin-subset*)
subgoal by (*auto simp*: *acids-def acids-mset-def dest*!: *multi-member-split*
dest: *subset-add-mset-notin-subset*)
done

definition *acids-flush-int* :: ‹*nat multiset* ⇒ (*nat*,*nat*) *ann-lits* ⇒ (*nat*, *nat*) *acids* ⇒ - ⇒ ((*nat*, *nat*) *acids* × -)*nres*› **where**

```
‹acids-flush-int $\mathcal{A}_{in}$ = (λM vm (to-remove, h). do {
    ASSERT(length to-remove ≤ unat32-max);
  (-, vm, h) ← WHILE$_T$$^{λ(i, vm', h). i ≤ length to-remove ∧}$        (i < length to-remove ⟶ acids-push-literal-pre $\mathcal{A}_{in}$ (to-re
      (λ(i, vm, h). i < length to-remove)
      (λ(i, vm, h). do {
          ASSERT(i < length to-remove);
          ASSERT(to-remove!i ∈# $\mathcal{A}_{in}$);
          ASSERT(atoms-hash-del-pre (to-remove!i) h);
          vm ← acids-push-literal (to-remove!i) vm;
          RETURN (i+1, vm, atoms-hash-del (to-remove!i) h)})
      (0, vm, h);
    RETURN (vm, (emptied-list to-remove, h))
  })›
```

definition *acids-flush*
:: ‹*nat multiset* ⇒ (*nat*,*nat*) *ann-lits* ⇒ (*nat*, *nat*) *acids* ⇒ *nat set* ⇒ ((*nat*, *nat*) *acids* × *nat set*) *nres*›
where
‹*acids-flush* $\mathcal{A}_{in}$ = (λ*M vm remove-int*. *SPEC* (λ*x*. (*fst x*) ∈ *acids* $\mathcal{A}_{in}$ *M* ∧ *snd x* = {}))›

lemma *acids-change-to-remove-order*:
assumes
vmtf: ‹*ac* ∈ *acids* $\mathcal{A}_{in}$ *M*› **and**
CD-rem: ‹((*C*, *D*), *to-remove*) ∈ *distinct-atoms-rel* $\mathcal{A}_{in}$› **and**
nempty: ‹*isasat-input-nempty* $\mathcal{A}_{in}$› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}_{in}$› **and**
t: ‹*to-remove* ⊆ *set-mset* $\mathcal{A}_{in}$›

shows ‹*acids-flush-int* $\mathcal{A}_{in}$ *M ac* (*C*, *D*) ≤ ⇓(*Id* ×$_r$ *distinct-atoms-rel* $\mathcal{A}_{in}$) (*acids-flush* $\mathcal{A}_{in}$ *M ac to-remove*)›
proof −
have *to-C*: ‹*to-remove* = *set C*›
using *CD-rem* **by** (*auto simp*: *distinct-atoms-rel-def distinct-hash-atoms-rel-def*)
have *length-le*: ‹*length* (*fst* (*C*,*D*)) ≤ *unat32-max*›
proof −
have *lits*: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ (*Pos* ‘# *mset C*)› **and**
dist: ‹*distinct C*›
using *vmtf CD-rem t* **unfolding** *vmtf-def*
vmtf-$\mathcal{L}_{all}$*-def*
apply (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-alt-def distinct-atoms-rel-alt-def inj-on-def*)
by (*metis atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *in-mono*)
have *dist*: ‹*distinct-mset* (*Pos* ‘# *mset C*)›
by (*subst distinct-image-mset-inj*)
(*use dist* **in** ‹*auto simp*: *inj-on-def*›)
have *tauto*: ‹¬ *tautology* (*poss* (*mset C*))›
by (*auto simp*: *tautology-decomp*)

show *?thesis*
using *simple-clss-size-upper-div2*[*OF bounded lits dist tauto*]
by (*auto simp*: *unat32-max-def*)
qed
have *acids-push-literal-pre*: ‹*acids-push-literal-pre* $\mathcal{A}_{in}$ (*C* ! *i*) *ac*›
if ‹*i* < *length C*› **for** *i*
using *t that CD-rem* **unfolding** *acids-push-literal-pre-def distinct-atoms-rel-def*
distinct-hash-atoms-rel-def **by** *auto*
define *I* **where** ‹*I* ≡ λ(*i*::*nat*, *vm*'::(*nat*, *nat*)*acids*, *h*). *vm*' ∈ *acids* $\mathcal{A}_{in}$ *M* ∧
((*drop i C*, *h*), *to-remove* − *set* (*take i C*)) ∈ *distinct-atoms-rel* $\mathcal{A}_{in}$ ∧ *i* ≤ *length C*›

have *sin*: ‹*fst s* < *length* (*fst* (*C*, *D*)) ⟹ *fst* (*C*, *D*) ! *fst s* ∈# $\mathcal{A}_{in}$› **and**
atms: ‹*I s* ⟹ *fst s* < *length* (*fst* (*C*, *D*)) ⟹ *atoms-hash-del-pre* (*fst* (*C*, *D*) ! *fst s*) (*snd* (*snd s*))›
for *s*
using *t CD-rem nth-mem*[*of* ‹*fst s*› *C*]
unfolding *acids-push-literal-pre-def distinct-atoms-rel-def*
distinct-hash-atoms-rel-def I-def atoms-hash-del-pre-def atoms-hash-rel-def **by** (*auto simp del*: *nth-mem*)

let *?R* = ‹*measure* (λ(*i*, *vm*', *h*). *length C* − *i*)›

have *I-inv1-acids-push-literal-pre*: ‹*I s* ⟹
fst (*C*, *D*) ! *fst s* ∈# $\mathcal{A}_{in}$ ⟹
x ∈ *acids* $\mathcal{A}_{in}$ *M* ⟹
fst (*fst s* + *1*, *x*,
atoms-hash-del (*fst* (*C*, *D*) ! *fst s*) (*snd* (*snd s*)))
< *length* (*fst* (*C*, *D*)) ⟹
acids-push-literal-pre $\mathcal{A}_{in}$
(*fst* (*C*, *D*) ! *fst* (*fst s* + *1*, *x*,
atoms-hash-del (*fst* (*C*, *D*) ! *fst s*) (*snd* (*snd s*))))
(*fst* (*snd* (*fst s* + *1*, *x*, *atoms-hash-del* (*fst* (*C*, *D*) ! *fst s*) (*snd* (*snd s*)))))›
for *s x*
using *t CD-rem* **unfolding** *acids-push-literal-pre-def distinct-atoms-rel-def*
distinct-hash-atoms-rel-def **by** *auto*
have *I-Suc*: ‹*I s* ⟹
fst s < *length* (*fst* (*C*, *D*)) ⟹
fst (*C*, *D*) ! *fst s* ∈# $\mathcal{A}_{in}$ ⟹

```
    atoms-hash-del-pre (fst (C, D) ! fst s) (snd (snd s)) ⟹
    x ∈ acids 𝒜_in M ⟹
    I (fst s + 1, x, atoms-hash-del (fst (C, D) ! fst s) (snd (snd s)))›
    for s x
    apply (auto simp: I-def distinct-atoms-rel-def
      distinct-hash-atoms-rel-def
       intro!: relcompI[of - ‹(drop (Suc (fst s)) C, (snd (snd s))[C ! (fst s) := False], to-remove − set
(take (Suc (fst s)) C))›])
    apply (rule  relcompI[of - ‹(drop (Suc (fst s)) C, to-remove − set (take (Suc (fst s)) C))›])
    subgoal
      by (auto simp: atoms-hash-rel-def atoms-hash-del-def take-Suc-conv-app-nth)
    subgoal
      by (auto simp: take-Suc-conv-app-nth simp flip: Cons-nth-drop-Suc)
    done

  show ?thesis
    unfolding acids-flush-int-def acids-flush-def case-prod-beta
    apply (refine-vcg specify-left[OF WHILEIT-rule-stronger-inv[where Φ = ‹(λx. I x ∧ fst x =length
(fst (C, D)))› and I' = ‹I› and R = ?R]]
      acids-push-literal[where 𝒜=𝒜_in and M=M])
    subgoal by (rule length-le)
    subgoal by auto
    subgoal by auto
    subgoal by (auto intro!: acids-push-literal-pre)
    subgoal using assms by (auto simp: I-def)
    subgoal by (rule sin)
    subgoal by (rule atms)
    subgoal by (auto simp: I-def)
    subgoal by auto
    subgoal by auto
    subgoal for s x by (rule I-inv1-acids-push-literal-pre)
    subgoal by (rule I-Suc)
    subgoal for s x by (auto simp: I-def)
    subgoal by (auto simp: emptied-list-def conc-fun-RES I-def)
    subgoal by (auto simp add: emptied-list-def conc-fun-RES I-def Image-iff to-C)
    done
qed
end
theory Tuple4
  imports
    More-Sepref.WB-More-Refinement IsaSAT-Literals
begin
```

This is the setup for accessing and modifying the state as an abstract tuple of 4 elements. The construction is kept generic (even if still targetting only our state). There is a lot of copy-paste that would be nice to automate at some point.

We define 3 sort of operations:

1. extracting an element, replacing it by an default element. Modifies the state. The name starts with *exctr*
2. reinserting an element, freeing the current one. Modifies the state. The name starts with *update*
3. in-place reading a value, possibly with pure parameters. Does not modify the state. The name starts with *read*

datatype ('a, 'b, 'c, 'd) *tuple4* = *Tuple4*
  (*Tuple4-a*: 'a)
  (*Tuple4-b*: 'b)
  (*Tuple4-c*: 'c)
  (*Tuple4-d*: 'd)

context
begin

qualified fun *set-a* :: ‹'a ⇒ ('a, 'b, 'c, 'd) *tuple4* ⇒ -› **where**
  ‹*set-a M* (*Tuple4* - *N D i*) = (*Tuple4 M N D i*)›

qualified fun *set-b* :: ‹'b ⇒ - ⇒ ('a, 'b, 'c, 'd) *tuple4*› **where**
  ‹*set-b N* (*Tuple4 M* - *D i*) = (*Tuple4 M N D i*)›

qualified fun *set-c* :: ‹'c ⇒ ('a, 'b, 'c, 'd) *tuple4* ⇒ ('a, 'b, 'c, 'd) *tuple4*› **where**
  ‹*set-c D* (*Tuple4 M N* - *i*) = (*Tuple4 M N D i*)›

qualified fun *set-d* :: ‹'d ⇒ ('a, 'b, 'c, 'd) *tuple4* ⇒ ('a, 'b, 'c, 'd) *tuple4*› **where**
  ‹*set-d i* (*Tuple4 M N D* -) = (*Tuple4 M N D i*)›

end

lemma *lambda-comp-true*: ‹(λ*S*. *True*) ∘ *f* = (λ-. *True*)› ‹*uncurry* (λ*a b*. *True*) = (λ-. *True*)› ‹*uncurry2* (λ*a b c*. *True*) = (λ-. *True*)›
  ‹*case-tuple4* (λ*M* - - -. *True*) = (λ-. *True*)›
  by (*auto intro*!: *ext split*: *Tuple4.tuple4.splits*)

named-theorems *Tuple4-state-simp* ‹*Simplify the state setter and extractors*›
lemma [*Tuple4-state-simp*]:
  ‹*Tuple4-a* (*Tuple4.set-a a S*) = *a*›
  ‹*Tuple4-b* (*Tuple4.set-a b S*) = *Tuple4-b S*›
  ‹*Tuple4-c* (*Tuple4.set-a b S*) = *Tuple4-c S*›
  ‹*Tuple4-d* (*Tuple4.set-a b S*) = *Tuple4-d S*›
  by (*cases S*; *auto*; *fail*)+

lemma [*Tuple4-state-simp*]:
  ‹*Tuple4-a* (*Tuple4.set-b b S*) = *Tuple4-a S*›
  ‹*Tuple4-b* (*Tuple4.set-b b S*) = *b*›
  ‹*Tuple4-c* (*Tuple4.set-b b S*) = *Tuple4-c S*›
  ‹*Tuple4-d* (*Tuple4.set-b b S*) = *Tuple4-d S*›
  by (*cases S*; *auto*; *fail*)+

lemma [*Tuple4-state-simp*]:
  ‹*Tuple4-a* (*Tuple4.set-c b S*) = *Tuple4-a S*›
  ‹*Tuple4-b* (*Tuple4.set-c b S*) = *Tuple4-b S*›
  ‹*Tuple4-c* (*Tuple4.set-c b S*) = *b*›
  ‹*Tuple4-d* (*Tuple4.set-c b S*) = *Tuple4-d S*›
  by (*cases S*; *auto*; *fail*)+

lemma [*Tuple4-state-simp*]:
  ‹*Tuple4-a* (*Tuple4.set-d b S*) = *Tuple4-a S*›
  ‹*Tuple4-b* (*Tuple4.set-d b S*) = *Tuple4-b S*›
  ‹*Tuple4-c* (*Tuple4.set-d b S*) = *Tuple4-c S*›
  ‹*Tuple4-d* (*Tuple4.set-d b S*) = *b*›
  by (*cases S*; *auto*; *fail*)+

declare *Tuple4-state-simp*[*simp*]
end
theory *IsaSAT-Bump-Heuristics-State*
imports *Watched-Literals-VMTF*
IsaSAT-ACIDS
Tuple4
begin

type-synonym *bump-heuristics* = ‹((*nat*, *nat*) *acids*, *vmtf*, *bool*, *nat list* × *bool list*) *tuple4*›

abbreviation *Bump-Heuristics* :: ‹- ⇒ - ⇒ - ⇒ - ⇒ *bump-heuristics*› **where**
‹*Bump-Heuristics a b c d* ≡ *Tuple4 a b c d*›

lemmas *bump-heuristics-splits* = *Tuple4.tuple4.splits*
hide-fact *tuple4.splits*

abbreviation *get-stable-heuristics* :: ‹*bump-heuristics* ⇒ (*nat*, *nat*) *acids*› **where**
‹*get-stable-heuristics* ≡ *Tuple4-a*›

abbreviation *get-focused-heuristics* :: ‹*bump-heuristics* ⇒ *vmtf*› **where**
‹*get-focused-heuristics* ≡ *Tuple4-b*›

abbreviation *is-focused-heuristics* :: ‹*bump-heuristics* ⇒ *bool*› **where**
‹*is-focused-heuristics* ≡ *Tuple4-c*›

abbreviation *is-stable-heuristics*:: ‹*bump-heuristics* ⇒ *bool*› **where**
‹*is-stable-heuristics x* ≡ ¬*is-focused-heuristics x*›

abbreviation *get-bumped-variables* :: ‹*bump-heuristics* ⇒ *nat list* × *bool list*› **where**
‹*get-bumped-variables* ≡ *Tuple4-d*›

abbreviation *set-stable-heuristics* :: ‹(*nat*, *nat*) *acids* ⇒*bump-heuristics* ⇒ -› **where**
‹*set-stable-heuristics* ≡ *Tuple4.set-a*›

abbreviation *set-focused-heuristics* :: ‹*vmtf* ⇒*bump-heuristics* ⇒ -› **where**
‹*set-focused-heuristics* ≡ *Tuple4.set-b*›

abbreviation *set-is-focused-heuristics* :: ‹*bool* ⇒*bump-heuristics* ⇒ -› **where**
‹*set-is-focused-heuristics* ≡ *Tuple4.set-c*›

abbreviation *set-bumped-variables* :: ‹*nat list* × *bool list* ⇒*bump-heuristics* ⇒ -› **where**
‹*set-bumped-variables* ≡ *Tuple4.set-d*›

definition *get-unit-trail* **where**
‹*get-unit-trail M* = (*rev* (*takeWhile* (λ*x*. ¬*is-decided x*) (*rev M*)))›

definition *bump-heur* :: ‹- ⇒ - ⇒ *bump-heuristics set*› **where**
‹*bump-heur* 𝒜 *M* = {*x*.
(*is-focused-heuristics x* ⟶
(*get-stable-heuristics x* ∈ *acids* 𝒜 (*get-unit-trail M*) ∧
get-focused-heuristics x ∈ *vmtf* 𝒜 *M*)) ∧
(¬*is-focused-heuristics x* ⟶
(*get-stable-heuristics x* ∈ *acids* 𝒜 *M* ∧
get-focused-heuristics x ∈ *vmtf* 𝒜 (*get-unit-trail M*))) ∧

(*get-bumped-variables x*, *set* (*fst* (*get-bumped-variables x*))) ∈ *distinct-atoms-rel* $\mathcal{A}$
}›

definition *switch-bump-heur* :: ‹*bump-heuristics* ⇒ *bump-heuristics*› **where**
‹*switch-bump-heur x* = *do* {
(*set-is-focused-heuristics* (¬(*is-focused-heuristics x*)) *x*)
}›

lemma *get-unit-trail-count-decided-0*[*simp*]: ‹*count-decided M* = *0* ⟹ *get-unit-trail M* = *M*›
by (*auto simp*: *get-unit-trail-def count-decided-0-iff*)
(*metis rev-swap set-rev takeWhile-eq-all-conv*)

lemma *switch-bump-heur*:
assumes ‹*x* ∈ *bump-heur* $\mathcal{A}$ *M*› **and**
‹*count-decided M* = *0*›
shows ‹*switch-bump-heur x* ∈ *bump-heur* $\mathcal{A}$ *M*›
using *assms*
by (*cases x*)
(*auto simp*: *switch-bump-heur-def bump-heur-def*)

9.3.1 Access Function

definition *isa-bump-unset-pre* **where**
‹*isa-bump-unset-pre* = (λ*L x*.
(*is-focused-heuristics x* ⟶ *vmtf-unset-pre L* (*get-focused-heuristics x*)) ∧
(*is-stable-heuristics x* ⟶ *acids-tl-pre L* (*get-stable-heuristics x*))
)›
definition *isa-bump-unset* :: ‹*nat* ⇒ *bump-heuristics* ⇒ *bump-heuristics nres*› **where**
‹*isa-bump-unset L vm* = (*case vm of Tuple4* (*hstable*) (*focused*) *foc a* ⇒ *do* {
hstable ← (*if* ¬*foc then acids-tl L hstable else RETURN hstable*);
let focused = (*if foc then vmtf-unset L focused else focused*);
RETURN (*Tuple4 hstable focused foc a*)
})›

lemma *get-unit-trail-simps*[*simp*]: ‹*is-decided L* ⟹ *get-unit-trail* (*L* # *M*) = *get-unit-trail M*›
‹¬*is-decided L* ⟹ *count-decided M* = *0* ⟹ *get-unit-trail* (*L* # *M*) = *L* # *M*›
‹¬*is-decided L* ⟹ *count-decided M* > *0* ⟹ *get-unit-trail* (*L* # *M*) = *get-unit-trail M*›
apply (*auto simp*: *get-unit-trail-def takeWhile-append*)
apply (*metis set-rev takeWhile-eq-all-conv*)
apply (*metis count-decided-0-iff nat-neq-iff*)
using *bot-nat-0.not-eq-extremum count-decided-0-iff* **by** *blast*

lemma *get-unit-trail-cons-if*:
‹*get-unit-trail* (*L* # *M*) = (*if is-decided L then get-unit-trail M else if count-decided M* = *0 then L* # *M else get-unit-trail M*)›
by (*auto simp*: *takeWhile-append*)

lemma *get-unit-trail-tl*[*simp*]: ‹*count-decided M* > *0* ⟹ *get-unit-trail* (*tl M*) = *get-unit-trail M*›
by (*cases M*; *cases* ‹*hd M*›) *auto*

lemma *isa-vmtf-consD*:
‹*x* ∈ *bump-heur* $\mathcal{A}$ *M* ⟹ *x* ∈ *bump-heur* $\mathcal{A}$ (*L* # *M*)›
by (*auto simp add*: *bump-heur-def takeWhile-append get-unit-trail-cons-if*
intro!: *vmtf-consD'* *acids-prepend*)

lemma *isa-bump-unset-vmtf-tl*:
  fixes *M*
  defines [*simp*]: ‹*L* ≡ *atm-of* (*lit-of* (*hd M*))›
  assumes *vmtf*: ‹*x*∈ *bump-heur* $\mathcal{A}$ *M*› **and**
    L-N: ‹*L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)› **and** [*simp*]: ‹*M* ≠ []› **and**
    nz: ‹*count-decided M* > 0›
  shows ‹*isa-bump-unset L x* ≤ *SPEC* (λ*a*. *a* ∈ *bump-heur* $\mathcal{A}$ (*tl M*))›
proof −
  obtain *ns m fst-As lst-As next-search* **where**
    ‹*is-focused-heuristics x* ⟹ *get-focused-heuristics x* = ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*))›
   by (*cases* ‹*get-focused-heuristics x*›; *cases* ‹*is-focused-heuristics x*›) *auto*
  then show *?thesis*
    using *vmtf-unset-vmtf-tl*[*of ns m fst-As lst-As next-search* $\mathcal{A}$ *M*] *nz*
      assms **unfolding** *isa-bump-unset-def* **apply** (*cases x*, *simp only*: *tuple4.case Let-def*)
    apply (*cases* ‹*is-focused-heuristics x*›)
    subgoal
      by (*refine-vcg acids-tl*[*of* - $\mathcal{A}$ *M*, *THEN order-trans*])
       (*auto simp*: *bump-heur-def isa-bump-unset-def split*: *bump-heuristics-splits*)
    subgoal
      by (*refine-vcg acids-tl*[*of* - $\mathcal{A}$ *M*, *THEN order-trans*])
        (*auto simp*: *bump-heur-def isa-bump-unset-def atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *split*: *bump-heuristics-splits*)
    done
qed

definition *bump-get-heuristics* :: ‹- ⇒ *vmtf*› **where**
  ‹*bump-get-heuristics x* = (*get-focused-heuristics x*)›

definition *length-bumped-vmtf-array* :: ‹*bump-heuristics* ⇒ *nat*› **where**
  ‹*length-bumped-vmtf-array x* =
  length (*fst* (*bump-get-heuristics x*))›

definition *current-vmtf-array-nxt-score* :: ‹*bump-heuristics* ⇒ *nat*› **where**
  ‹*current-vmtf-array-nxt-score x* = *fst* (*snd* (*bump-get-heuristics x*))›
definition *access-focused-vmtf-array* **where**
  ‹*access-focused-vmtf-array x i* = *do* {
  ASSERT (*i* < *length* (*fst* (*bump-get-heuristics x*)));
  RETURN (*fst* (*bump-get-heuristics x*) ! *i*)}›

definition *bumped-vmtf-array-fst* **where**
  ‹*bumped-vmtf-array-fst x* =
  fst (*snd* (*snd* (*bump-get-heuristics x*)))›

definition *isa-bump-mark-to-rescore*
  :: ‹*nat* ⇒ *bump-heuristics* ⇒ *bump-heuristics nres*›
where
  ‹*isa-bump-mark-to-rescore L x* = (*case x of Bump-Heuristics a b c d* ⇒ *do* {
    ASSERT (*atms-hash-insert-pre L d*);
    RETURN (*Bump-Heuristics a b c* (*atoms-hash-insert L d*))
  })›

end
theory *IsaSAT-Setup*
  imports

Tuple17
IsaSAT-Phasing
Watched-Literals.Watched-Literals-Watch-List-Initialisation
IsaSAT-Lookup-Conflict
IsaSAT-Clauses IsaSAT-Arena IsaSAT-Watch-List LBD
IsaSAT-Options
IsaSAT-Rephase
IsaSAT-EMA
IsaSAT-Stats
IsaSAT-Profile
IsaSAT-VDom
IsaSAT-Occurence-List
IsaSAT-Bump-Heuristics-State
begin

Chapter 10

Complete state

hide-const (**open**) *IsaSAT-VDom.get-aivdom*

hide-const (**open**) *NEMonad.ASSERT NEMonad.RETURN NEMonad.SPEC*

We here define the last step of our refinement: the step with all the heuristics and fully deterministic code.

After the result of benchmarking, we concluded that the use of *nat* leads to worse performance than using *sint64*. As, however, the later is not complete, we do so with a switch: as long as it fits, we use the faster (called 'bounded') version. After that we switch to the 'unbounded' version (which is still bounded by memory anyhow) if we generate Standard ML code.

We have successfully killed all natural numbers when generating LLVM. However, the LLVM binding does not have a binding to GMP integers.

fun *get-unkept-unit-init-clss-wl* :: ‹*'v twl-st-wl* ⇒ *'v clauses*› **where**
‹*get-unkept-unit-init-clss-wl* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *Q*, *W*) = *NE*›

fun *get-unkept-unit-learned-clss-wl* :: ‹*'v twl-st-wl* ⇒ *'v clauses*› **where**
‹*get-unkept-unit-learned-clss-wl* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *Q*, *W*) = *UE*›

fun *get-kept-unit-init-clss-wl* :: ‹*'v twl-st-wl* ⇒ *'v clauses*› **where**
‹*get-kept-unit-init-clss-wl* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *Q*, *W*) = *NEk*›

fun *get-kept-unit-learned-clss-wl* :: ‹*'v twl-st-wl* ⇒ *'v clauses*› **where**
‹*get-kept-unit-learned-clss-wl* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *Q*, *W*) = *UEk*›

lemma *get-unit-init-clss-wl-alt-def*:
‹*get-unit-init-clss-wl T* = *get-unkept-unit-init-clss-wl T* + *get-kept-unit-init-clss-wl T*›
by (*cases T*) *auto*

lemma *get-unit-learned-clss-wl-alt-def*:
‹*get-unit-learned-clss-wl T* = *get-unkept-unit-learned-clss-wl T* + *get-kept-unit-learned-clss-wl T*›
by (*cases T*) *auto*

10.1 VMTF

type-synonym *out-learned* = ‹*nat clause-l*›

10.1.1 Conflict

definition *size-conflict-wl* :: ‹*nat twl-st-wl* ⇒ *nat*› **where**

⟨*size-conflict-wl S = size* (*the* (*get-conflict-wl S*))⟩

definition *size-conflict* :: ⟨*nat clause option ⇒ nat*⟩ **where**
⟨*size-conflict D = size* (*the D*)⟩

definition *size-conflict-int* :: ⟨*conflict-option-rel ⇒ nat*⟩ **where**
⟨*size-conflict-int* = (λ(-, *n*, -). *n*)⟩

10.2 Full state

heur stands for heuristic.

Definition **type-synonym** *isasat* = ⟨(*trail-pol*, *arena*,
conflict-option-rel, *nat*, (*nat watcher*) *list list*, *bump-heuristics*,
nat, *conflict-min-cach-l*, *lbd*, *out-learned*, *isasat-stats*, *isasat-restart-heuristics*,
isasat-aivdom, *clss-size*, *opts*, *arena*, *occurences-ref*) *tuple17*⟩

abbreviation *IsaSAT* **where**
⟨*IsaSAT a b c d e f g h i j k l m n xo p occs* ≡ *Tuple17 a b c d e f g h i j k l m n xo p occs* :: *isasat*⟩

lemmas *isasat-int-splits* = *Tuple17.tuple17.splits*
hide-fact *tuple17.splits*

abbreviation *case-isasat-int* :: ⟨- ⇒ *isasat* ⇒ -⟩ **where**
⟨*case-isasat-int* ≡ *case-tuple17*⟩

abbreviation *get-trail-wl-heur* :: ⟨*isasat ⇒ trail-pol*⟩ **where**
⟨*get-trail-wl-heur* ≡ *Tuple17-get-a*⟩

abbreviation *get-clauses-wl-heur* :: ⟨*isasat ⇒ arena*⟩ **where**
⟨*get-clauses-wl-heur* ≡ *Tuple17-get-b*⟩

abbreviation *get-conflict-wl-heur* :: ⟨*isasat ⇒ conflict-option-rel*⟩ **where**
⟨*get-conflict-wl-heur* ≡ *Tuple17-get-c*⟩

abbreviation *literals-to-update-wl-heur* :: ⟨*isasat ⇒ nat*⟩ **where**
⟨*literals-to-update-wl-heur* ≡ *Tuple17-get-d*⟩

abbreviation *get-watched-wl-heur* :: ⟨*isasat ⇒* (*nat watcher*) *list list*⟩ **where**
⟨*get-watched-wl-heur* ≡ *Tuple17-get-e*⟩

abbreviation *get-vmtf-heur* :: ⟨*isasat ⇒ bump-heuristics*⟩ **where**
⟨*get-vmtf-heur* ≡ *Tuple17-get-f*⟩

abbreviation *get-count-max-lvls-heur* :: ⟨*isasat ⇒ nat*⟩ **where**
⟨*get-count-max-lvls-heur* ≡ *Tuple17-get-g*⟩

abbreviation *get-conflict-cach* :: ⟨*isasat ⇒ conflict-min-cach-l*⟩ **where**
⟨*get-conflict-cach* ≡ *Tuple17-get-h*⟩

abbreviation *get-lbd* :: ⟨*isasat ⇒ lbd*⟩ **where**
⟨*get-lbd* ≡ *Tuple17-get-i*⟩

abbreviation *get-outlearned-heur* :: ‹*isasat* ⇒ *out-learned*› **where**
‹*get-outlearned-heur* ≡ *Tuple17-get-j*›

abbreviation *get-stats-heur* :: ‹*isasat* ⇒ *isasat-stats*› **where**
‹*get-stats-heur* ≡ *Tuple17-get-k*›

abbreviation *get-heur* :: ‹*isasat* ⇒ *isasat-restart-heuristics*› **where**
‹*get-heur* ≡ *Tuple17-get-l*›

abbreviation *get-aivdom* :: ‹*isasat* ⇒ *isasat-aivdom*› **where**
‹*get-aivdom* ≡ *Tuple17-get-m*›

abbreviation *get-learned-count* :: ‹*isasat* ⇒ *clss-size*› **where**
‹*get-learned-count* ≡ *Tuple17-get-n*›

abbreviation *get-opts* :: ‹*isasat* ⇒ *opts*› **where**
‹*get-opts* ≡ *Tuple17-get-o*›

abbreviation *get-old-arena* :: ‹*isasat* ⇒ *arena*› **where**
‹*get-old-arena* ≡ *Tuple17-get-p*›

abbreviation *get-occs* :: ‹*isasat* ⇒ *occurences-ref*› **where**
‹*get-occs* ≡ *Tuple17-get-q*›

abbreviation *set-trail-wl-heur* :: ‹*trail-pol* ⇒*isasat* ⇒ -› **where**
‹*set-trail-wl-heur* ≡ *Tuple17.set-a*›

abbreviation *set-clauses-wl-heur* :: ‹*arena* ⇒*isasat* ⇒ -› **where**
‹*set-clauses-wl-heur* ≡ *Tuple17.set-b*›

abbreviation *set-conflict-wl-heur* :: ‹*conflict-option-rel* ⇒*isasat* ⇒ -› **where**
‹*set-conflict-wl-heur* ≡ *Tuple17.set-c*›

abbreviation *set-literals-to-update-wl-heur* :: ‹*nat* ⇒*isasat* ⇒ -› **where**
‹*set-literals-to-update-wl-heur* ≡ *Tuple17.set-d*›

abbreviation *set-watched-wl-heur* :: ‹*nat watcher list list* ⇒*isasat* ⇒ -› **where**
‹*set-watched-wl-heur* ≡ *Tuple17.set-e*›

abbreviation *set-vmtf-wl-heur* :: ‹*bump-heuristics* ⇒*isasat* ⇒ -› **where**
‹*set-vmtf-wl-heur* ≡ *Tuple17.set-f*›

abbreviation *set-count-max-wl-heur* :: ‹*nat* ⇒*isasat* ⇒ -› **where**
‹*set-count-max-wl-heur* ≡ *Tuple17.set-g*›

abbreviation *set-ccach-max-wl-heur* :: ‹*conflict-min-cach-l* ⇒*isasat* ⇒ -› **where**
‹*set-ccach-max-wl-heur* ≡ *Tuple17.set-h*›

abbreviation *set-lbd-wl-heur* :: ‹*lbd* ⇒*isasat* ⇒ -› **where**
‹*set-lbd-wl-heur* ≡ *Tuple17.set-i*›

abbreviation *set-outl-wl-heur* :: ‹*out-learned* ⇒*isasat* ⇒ -› **where**
‹*set-outl-wl-heur* ≡ *Tuple17.set-j*›

abbreviation *set-stats-wl-heur* :: ‹*isasat-stats* ⇒ *isasat* ⇒ *isasat*› **where**
‹*set-stats-wl-heur* ≡ *Tuple17.set-k*›

abbreviation *set-heur-wl-heur* :: ‹*isasat-restart-heuristics* ⇒*isasat* ⇒ *isasat*› **where**
‹*set-heur-wl-heur* ≡ *Tuple17.set-l*›

abbreviation *set-aivdom-wl-heur* :: ‹*isasat-aivdom* ⇒*isasat* ⇒ -› **where**
‹*set-aivdom-wl-heur* ≡ *Tuple17.set-m*›

abbreviation *set-learned-count-wl-heur* :: ‹*clss-size* ⇒*isasat* ⇒ -› **where**
‹*set-learned-count-wl-heur* ≡ *Tuple17.set-n*›

abbreviation *set-opts-wl-heur* :: ‹*opts* ⇒*isasat* ⇒ -› **where**
‹*set-opts-wl-heur* ≡ *Tuple17.set-o*›

abbreviation *set-old-arena-wl-heur* :: ‹*arena* ⇒*isasat* ⇒ -› **where**
‹*set-old-arena-wl-heur* ≡ *Tuple17.set-p*›

abbreviation *set-occs-wl-heur* :: ‹*occurences-ref* ⇒*isasat* ⇒ -› **where**
‹*set-occs-wl-heur* ≡ *Tuple17.set-q*›

fun *watched-by-int* :: ‹*isasat* ⇒ *nat literal* ⇒ *nat watched*› **where**
‹*watched-by-int S L* = *get-watched-wl-heur S* ! *nat-of-lit L*›

definition *watched-by-app-heur-pre* **where**
‹*watched-by-app-heur-pre* = (λ((*S*, *L*), *K*). *nat-of-lit L* < *length* (*get-watched-wl-heur S*) ∧
K < *length* (*watched-by-int S L*))›

definition *watched-by-app-heur* :: ‹*isasat* ⇒ *nat literal* ⇒ *nat* ⇒ *nat watcher*› **where**
‹*watched-by-app-heur S L K* = *watched-by-int S L* ! *K*›

definition *mop-watched-by-app-heur* :: ‹*isasat* ⇒ *nat literal* ⇒ *nat* ⇒ *nat watcher nres*› **where**
‹*mop-watched-by-app-heur S L K* = *do* {
ASSERT(*K* < *length* (*watched-by-int S L*));
ASSERT(*nat-of-lit L* < *length* (*get-watched-wl-heur S*));
RETURN (*watched-by-int S L* ! *K*)}›

definition *watched-by-app* :: ‹*nat twl-st-wl* ⇒ *nat literal* ⇒ *nat* ⇒ *nat watcher*› **where**
‹*watched-by-app S L K* = *watched-by S L* ! *K*›

fun *get-fast-ema-heur* :: ‹*isasat* ⇒ *ema*› **where**
‹*get-fast-ema-heur S* = *fast-ema-of* (*get-heur S*)›

fun *get-slow-ema-heur* :: ‹*isasat* ⇒ *ema*› **where**
‹*get-slow-ema-heur S* = *slow-ema-of* (*get-heur S*)›

fun *get-conflict-count-heur* :: ‹*isasat* ⇒ *restart-info*› **where**
‹*get-conflict-count-heur S* = *restart-info-of* (*get-heur S*)›

abbreviation *get-vdom* :: ‹*isasat* ⇒ *nat list*› **where**
‹*get-vdom S* ≡ *get-vdom-aivdom* (*get-aivdom S*)›

abbreviation *get-avdom* :: ‹*isasat* ⇒ *nat list*› **where**
‹*get-avdom S* ≡ *get-avdom-aivdom* (*get-aivdom S*)›

abbreviation *get-ivdom* :: ‹*isasat* ⇒ *nat list*› **where**
‹*get-ivdom S* ≡ *get-ivdom-aivdom* (*get-aivdom S*)›

abbreviation *get-tvdom* :: ‹*isasat* ⇒ *nat list*› **where**
‹*get-tvdom S* ≡ *get-tvdom-aivdom* (*get-aivdom S*)›

abbreviation *get-learned-count-number* :: ‹*isasat* ⇒ *nat*› **where**
‹*get-learned-count-number S* ≡ *clss-size-lcount* (*get-learned-count S*)›

definition *get-restart-phase* :: ‹*isasat* ⇒ *64 word*› **where**
‹*get-restart-phase* = (λ*S*.
current-restart-phase (*get-heur S*))›

definition *cach-refinement-empty* **where**
‹*cach-refinement-empty* 𝒜 *cach* ⟷
(*cach*, λ-. *SEEN-UNKNOWN*) ∈ *cach-refinement* 𝒜›

VMTF *vdom* is an upper bound on all the address of the clauses that are used in the state. *avdom* includes the active clauses.

definition *twl-st-heur* :: ‹(*isasat* × *nat twl-st-wl*) *set*› **where**
[*unfolded Let-def*]: ‹*twl-st-heur* =
{(*S*, *T*).
let M′ = *get-trail-wl-heur S*; *N′* = *get-clauses-wl-heur S*; *D′* = *get-conflict-wl-heur S*;
W′ = *get-watched-wl-heur S*; *j* = *literals-to-update-wl-heur S*; *outl* = *get-outlearned-heur S*;
cach = *get-conflict-cach S*; *clvls* = *get-count-max-lvls-heur S*;
vm = *get-vmtf-heur S*;
vdom = *get-aivdom S*; *heur* = *get-heur S*; *old-arena* = *get-old-arena S*;
lcount = *get-learned-count S*;
occs = *get-occs S in*
let M = *get-trail-wl T*; *N* = *get-clauses-wl T*; *D* = *get-conflict-wl T*;
Q = *literals-to-update-wl T*;
W = *get-watched-wl T*; *N0* = *get-init-clauses0-wl T*; *U0* = *get-learned-clauses0-wl T*;
NS = *get-subsumed-init-clauses-wl T*; *US* = *get-subsumed-learned-clauses-wl T*;
NEk = *get-kept-unit-init-clss-wl T*; *UEk* = *get-kept-unit-learned-clss-wl T*;
NE = *get-unkept-unit-init-clss-wl T*; *UE* = *get-unkept-unit-learned-clss-wl T in*
(*M′*, *M*) ∈ *trail-pol* (*all-atms-st T*) ∧
valid-arena N′ N (*set* (*get-vdom-aivdom vdom*)) ∧
(*D′*, *D*) ∈ *option-lookup-clause-rel* (*all-atms-st T*) ∧
(*D* = *None* ⟶ *j* ≤ *length M*) ∧
Q = *uminus* ‘# *lit-of* ‘# *mset* (*drop j* (*rev M*)) ∧
(*W′*, *W*) ∈ ⟨*Id*⟩*map-fun-rel* (*D*$_0$ (*all-atms-st T*)) ∧
vm ∈ *bump-heur* (*all-atms-st T*) *M* ∧
no-dup M ∧
clvls ∈ *counts-maximum-level M D* ∧
cach-refinement-empty (*all-atms-st T*) *cach* ∧
out-learned M D outl ∧
clss-size-corr N NE UE NEk UEk NS US N0 U0 lcount ∧
vdom-m (*all-atms-st T*) *W N* ⊆ *set* (*get-vdom-aivdom vdom*) ∧
aivdom-inv-dec vdom (*dom-m N*) ∧
isasat-input-bounded (*all-atms-st T*) ∧
isasat-input-nempty (*all-atms-st T*) ∧
old-arena = [] ∧
heuristic-rel (*all-atms-st T*) *heur* ∧
(*occs*, *empty-occs-list* (*all-atms-st T*)) ∈ *occurrence-list-ref*
}›

lemma *twl-st-heur-state-simp*:

assumes ‹$(S,\ S') \in$ *twl-st-heur*›
shows
‹(*get-trail-wl-heur* S, *get-trail-wl* S') $\in$ *trail-pol* (*all-atms-st* S')› **and**
twl-st-heur-state-simp-watched: ‹$C \in\#$ $\mathcal{L}_{all}$ (*all-atms-st* S') $\Longrightarrow$
watched-by-int S C = *watched-by* S' C›
‹$C \in\#$ $\mathcal{L}_{all}$ (*all-atms-st* S') $\Longrightarrow$
get-watched-wl-heur S ! (*nat-of-lit* C) = *get-watched-wl* S' C›**and**
‹*literals-to-update-wl* S' =
uminus ‘# *lit-of* ‘# *mset* (*drop* (*literals-to-update-wl-heur* S) (*rev* (*get-trail-wl* S')))› **and**
twl-st-heur-state-simp-watched2: ‹$C \in\#$ $\mathcal{L}_{all}$ (*all-atms-st* S') $\Longrightarrow$
nat-of-lit $C <$ *length*(*get-watched-wl-heur* S)›
using *assms* **unfolding** *twl-st-heur-def* **by** (*solves* ‹*cases* S; *cases* S'; *auto simp add*: *Let-def map-fun-rel-def ac-simps all-atms-st-def*›)+

This is the version of the invariants where some informations are already lost. For example, losing statistics does not matter if UNSAT was derived.

definition *twl-st-heur-loop* :: ‹(*isasat* × *nat twl-st-wl*) *set*› **where**
[*unfolded Let-def*]: ‹*twl-st-heur-loop* =
{(S, T).
let M' = *get-trail-wl-heur* S; N' = *get-clauses-wl-heur* S; D' = *get-conflict-wl-heur* S;
W' = *get-watched-wl-heur* S; j = *literals-to-update-wl-heur* S; *outl* = *get-outlearned-heur* S;
cach = *get-conflict-cach* S; *clvls* = *get-count-max-lvls-heur* S;
vm = *get-vmtf-heur* S;
vdom = *get-aivdom* S; *heur* = *get-heur* S; *old-arena* = *get-old-arena* S;
lcount = *get-learned-count* S; *occs* = *get-occs* S *in*
let M = *get-trail-wl* T; N = *get-clauses-wl* T; D = *get-conflict-wl* T;
Q = *literals-to-update-wl* T;
W = *get-watched-wl* T; *N0* = *get-init-clauses0-wl* T; *U0* = *get-learned-clauses0-wl* T;
NS = *get-subsumed-init-clauses-wl* T; *US* = *get-subsumed-learned-clauses-wl* T;
NEk = *get-kept-unit-init-clss-wl* T; *UEk* = *get-kept-unit-learned-clss-wl* T;
NE = *get-unkept-unit-init-clss-wl* T; *UE* = *get-unkept-unit-learned-clss-wl* T *in*
(M', M) $\in$ *trail-pol* (*all-atms-st* T) $\wedge$
valid-arena N' N (*set* (*get-vdom-aivdom vdom*)) $\wedge$
(D', D) $\in$ *option-lookup-clause-rel* (*all-atms-st* T) $\wedge$
(D = *None* $\longrightarrow$ $j \leq$ *length* M) $\wedge$
Q = *uminus* ‘# *lit-of* ‘# *mset* (*drop* j (*rev* M)) $\wedge$
(W', W) $\in \langle Id \rangle$*map-fun-rel* (D_0 (*all-atms-st* T)) $\wedge$
vm $\in$ *bump-heur* (*all-atms-st* T) M $\wedge$
no-dup M $\wedge$
clvls $\in$ *counts-maximum-level* M D $\wedge$
cach-refinement-empty (*all-atms-st* T) *cach* $\wedge$
out-learned M D *outl* $\wedge$
(D = *None* $\longrightarrow$ *clss-size-corr* N *NE UE NEk UEk NS US N0 U0 lcount*) $\wedge$
vdom-m (*all-atms-st* T) W $N \subseteq$ *set* (*get-vdom-aivdom vdom*) $\wedge$
aivdom-inv-dec vdom (*dom-m* N) $\wedge$
isasat-input-bounded (*all-atms-st* T) $\wedge$
isasat-input-nempty (*all-atms-st* T) $\wedge$
old-arena = [] $\wedge$
heuristic-rel (*all-atms-st* T) *heur*$\wedge$
(*occs*, *empty-occs-list* (*all-atms-st* T)) $\in$ *occurrence-list-ref*
}›

abbreviation *learned-clss-count-lcount* :: ‹-› **where**
‹*learned-clss-count-lcount* $S \equiv$ *clss-size-lcount* (S) +
clss-size-lcountUE (S) + *clss-size-lcountUEk* (S) +
clss-size-lcountUS (S) +

clss-size-lcountU0 (S)›

definition *learned-clss-count* :: ‹*isasat* ⇒ *nat*› **where**
‹*learned-clss-count S* = *learned-clss-count-lcount* (*get-learned-count S*)›

lemma *get-learned-count-learned-clss-countD*:
‹*get-learned-count S* = *clss-size-resetUS* (*get-learned-count T*) ⟹
learned-clss-count S ≤ *learned-clss-count T*›
‹*get-learned-count S* = *clss-size-resetUS0* (*get-learned-count T*) ⟹
learned-clss-count S ≤ *learned-clss-count T*›
by (*cases S*; *cases T*; *auto simp*: *learned-clss-count-def*; *fail*)+

lemma *get-learned-count-learned-clss-countD2*:
‹*get-learned-count S* = (*get-learned-count T*) ⟹
learned-clss-count S = *learned-clss-count T*›
by (*cases S*; *cases T*) (*auto simp*: *learned-clss-count-def*)

abbreviation *twl-st-heur'''*
:: ‹*nat* ⇒ (*isasat* × *nat twl-st-wl*) *set*›
where
‹*twl-st-heur'''* r ≡ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur* ∧
length (*get-clauses-wl-heur S*) = *r*}›

abbreviation *twl-st-heur''''u*
:: ‹*nat* ⇒ *nat* ⇒ (*isasat* × *nat twl-st-wl*) *set*›
where
‹*twl-st-heur''''u* r u ≡ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur* ∧
length (*get-clauses-wl-heur S*) = *r* ∧
learned-clss-count S ≤ *u*}›

lemma *twl-st-heur'''-twl-st-heur''''uD*:
‹(*x*, *y*) ∈ *twl-st-heur'''* r ⟹
(*x*, *y*) ∈ *twl-st-heur''''u* r (*learned-clss-count x*)›
by *auto*

definition *twl-st-heur'* :: ‹*nat multiset* ⇒ (*isasat* × *nat twl-st-wl*) *set*› **where**
‹*twl-st-heur'* N = {(*S*, *S'*). (*S*, *S'*) ∈ *twl-st-heur* ∧ *dom-m* (*get-clauses-wl S'*) = *N*}›

definition *twl-st-heur-conflict-ana*
:: ‹(*isasat* × *nat twl-st-wl*) *set*›
where
[*unfolded Let-def*]: ‹*twl-st-heur-conflict-ana* =
{(*S*, *T*).
let M' = *get-trail-wl-heur S*; *N'* = *get-clauses-wl-heur S*; *D'* = *get-conflict-wl-heur S*;
W' = *get-watched-wl-heur S*; *j* = *literals-to-update-wl-heur S*; *outl* = *get-outlearned-heur S*;
cach = *get-conflict-cach S*; *clvls* = *get-count-max-lvls-heur S*;
vm = *get-vmtf-heur S*;
vdom = *get-aivdom S*; *heur* = *get-heur S*; *old-arena* = *get-old-arena S*;
lcount = *get-learned-count S*; *occs* = *get-occs S in*
let M = *get-trail-wl T*; *N* = *get-clauses-wl T*; *D* = *get-conflict-wl T*;
Q = *literals-to-update-wl T*;
W = *get-watched-wl T*; *N0* = *get-init-clauses0-wl T*; *U0* = *get-learned-clauses0-wl T*;
NS = *get-subsumed-init-clauses-wl T*; *US* = *get-subsumed-learned-clauses-wl T*;
NEk = *get-kept-unit-init-clss-wl T*; *UEk* = *get-kept-unit-learned-clss-wl T*;
NE = *get-unkept-unit-init-clss-wl T*; *UE* = *get-unkept-unit-learned-clss-wl T in*
(*M'*, *M*) ∈ *trail-pol* (*all-atms-st T*) ∧

valid-arena N′ N (set (get-vdom-aivdom vdom)) ∧
(D′, D) ∈ option-lookup-clause-rel (all-atms-st T) ∧
(W′, W) ∈ ⟨Id⟩map-fun-rel (D₀ (all-atms-st T)) ∧
vm ∈ bump-heur (all-atms-st T) M ∧
no-dup M ∧
clvls ∈ counts-maximum-level M D ∧
cach-refinement-empty (all-atms-st T) cach ∧
out-learned M D outl ∧
clss-size-corr N NE UE NEk UEk NS US N0 U0 lcount ∧
vdom-m (all-atms-st T) W N ⊆ set (get-vdom-aivdom vdom) ∧
aivdom-inv-dec vdom (dom-m N) ∧
isasat-input-bounded (all-atms-st T) ∧
isasat-input-nempty (all-atms-st T) ∧
old-arena = [] ∧
heuristic-rel (all-atms-st T) heur∧
(occs, empty-occs-list (all-atms-st T)) ∈ occurrence-list-ref
}›

lemma *twl-st-heur-twl-st-heur-conflict-ana*:
‹(*S*, *T*) ∈ *twl-st-heur* ⟹ (*S*, *T*) ∈ *twl-st-heur-conflict-ana*›
by (*cases S*; *cases T*; *auto simp*: *twl-st-heur-def twl-st-heur-conflict-ana-def Let-def ac-simps all-atms-st-def*)

lemma *twl-st-heur-ana-state-simp*:
assumes ‹(*S*, *S′*) ∈ *twl-st-heur-conflict-ana*›
shows
‹(*get-trail-wl-heur S*, *get-trail-wl S′*) ∈ *trail-pol* (*all-atms-st S′*)› **and**
‹*C* ∈# $\mathcal{L}_{all}$ (*all-atms-st S′*) ⟹ *watched-by-int S C* = *watched-by S′ C*›
using *assms* **unfolding** *twl-st-heur-conflict-ana-def* **by** (*solves* ‹*cases S*; *cases S′*; *auto simp*: *map-fun-rel-def ac-simps*
all-atms-st-def Let-def›)+

This relations decouples the conflict that has been minimised and appears abstractly from the refined state, where the conflict has been removed from the data structure to a separate array.

definition *twl-st-heur-bt* :: ‹(*isasat* × *nat twl-st-wl*) *set*› **where**
[*unfolded Let-def*]: ‹*twl-st-heur-bt* =
{(*S*, *T*).
let M′ = *get-trail-wl-heur S*; *N′* = *get-clauses-wl-heur S*; *D′* = *get-conflict-wl-heur S*;
W′ = *get-watched-wl-heur S*; *j* = *literals-to-update-wl-heur S*; *outl* = *get-outlearned-heur S*;
cach = *get-conflict-cach S*; *clvls* = *get-count-max-lvls-heur S*;
vm = *get-vmtf-heur S*;
vdom = *get-aivdom S*; *heur* = *get-heur S*; *old-arena* = *get-old-arena S*;
lcount = *get-learned-count S*; *occs* = *get-occs S in*
let M = *get-trail-wl T*; *N* = *get-clauses-wl T*; *D* = *get-conflict-wl T*;
Q = *literals-to-update-wl T*;
W = *get-watched-wl T*; *N0* = *get-init-clauses0-wl T*; *U0* = *get-learned-clauses0-wl T*;
NS = *get-subsumed-init-clauses-wl T*; *US* = *get-subsumed-learned-clauses-wl T*;
NEk = *get-kept-unit-init-clss-wl T*; *UEk* = *get-kept-unit-learned-clss-wl T*;
NE = *get-unkept-unit-init-clss-wl T*; *UE* = *get-unkept-unit-learned-clss-wl T in*
(M′, M) ∈ trail-pol (all-atms-st T) ∧
valid-arena N′ N (set (get-vdom-aivdom vdom)) ∧
(D′, None) ∈ option-lookup-clause-rel (all-atms-st T) ∧
(W′, W) ∈ ⟨Id⟩map-fun-rel (D₀ (all-atms-st T)) ∧
vm ∈ bump-heur (all-atms-st T) M ∧
no-dup M ∧
clvls ∈ counts-maximum-level M None ∧
cach-refinement-empty (all-atms-st T) cach ∧

out-learned M None outl ∧
clss-size-corr N NE UE NEk UEk NS US N0 U0 lcount ∧
vdom-m (*all-atms-st T*) *W N* ⊆ *set* (*get-vdom-aivdom vdom*) ∧
aivdom-inv-dec vdom (*dom-m N*) ∧
isasat-input-bounded (*all-atms-st T*) ∧
isasat-input-nempty (*all-atms-st T*) ∧
old-arena = [] ∧
heuristic-rel (*all-atms-st T*) *heur* ∧
(*occs*, *empty-occs-list* (*all-atms-st T*)) ∈ *occurrence-list-ref*
}›

The difference between *isasat-unbounded-assn* and *isasat-bounded-assn* corresponds to the following condition:

definition *isasat-fast* :: ‹*isasat* ⇒ *bool*› **where**
‹*isasat-fast S* ⟷ (*length* (*get-clauses-wl-heur S*) ≤ *snat64-max* − (*unat32-max div 2* + *MAX-HEADER-SIZE+1*)
∧
learned-clss-count S < *unat64-max*)›

lemma *isasat-fast-length-leD*: ‹*isasat-fast S* ⟹ *length* (*get-clauses-wl-heur S*) ≤ *snat64-max*› **and**
isasat-fast-countD:
‹*isasat-fast S* ⟹ *clss-size-lcount* (*get-learned-count S*) < *unat64-max*›
‹*isasat-fast S* ⟹ *clss-size-lcountUS* (*get-learned-count S*) < *unat64-max*›
‹*isasat-fast S* ⟹ *clss-size-lcountUE* (*get-learned-count S*) < *unat64-max*›
‹*isasat-fast S* ⟹ *clss-size-lcountU0* (*get-learned-count S*) < *unat64-max*›
by (*solves* ‹*cases S*; *auto simp*: *isasat-fast-def clss-size-lcountUS-def*
clss-size-lcountUE-def clss-size-lcount-def clss-size-lcountU0-def
clss-size-allcount-def learned-clss-count-def›)+

10.3 Lift Operations to State

definition *polarity-st* :: ‹'*v twl-st-wl* ⇒ '*v literal* ⇒ *bool option*› **where**
‹*polarity-st S* = *polarity* (*get-trail-wl S*)›

definition *get-conflict-wl-is-None-heur* :: ‹*isasat* ⇒ *bool*› **where**
‹*get-conflict-wl-is-None-heur S* = (λ(*b*, -). *b*) (*get-conflict-wl-heur S*)›

lemma *get-conflict-wl-is-None-heur-get-conflict-wl-is-None*:
‹(*RETURN o get-conflict-wl-is-None-heur*, *RETURN o get-conflict-wl-is-None*) ∈
twl-st-heur →$_f$ ⟨*Id*⟩*nres-rel*›
unfolding *get-conflict-wl-is-None-heur-def get-conflict-wl-is-None-def comp-def*
apply (*intro frefI nres-relI*) **apply** *refine-rcg*
by (*auto simp*: *twl-st-heur-def get-conflict-wl-is-None-heur-def get-conflict-wl-is-None-def*
option-lookup-clause-rel-def Let-def
split: *option.splits prod.splits*)

definition *count-decided-st* :: ‹*nat twl-st-wl* ⇒ *nat*› **where**
‹*count-decided-st* = (λ(*M*, -). *count-decided M*)›

lemma *count-decided-st-alt-def*: ‹*count-decided-st S* = *count-decided* (*get-trail-wl S*)›
unfolding *count-decided-st-def*
by (*cases S*) *auto*

definition (**in** −) *is-in-conflict-st* :: ‹*nat literal* ⇒ *nat twl-st-wl* ⇒ *bool*› **where**
‹*is-in-conflict-st L S* ⟷ *is-in-conflict L* (*get-conflict-wl S*)›

definition *atm-is-in-conflict-st-heur* :: ‹*nat literal* ⇒ *isasat* ⇒ *bool nres*› **where**
‹*atm-is-in-conflict-st-heur L S* = (λ(-, *D*). *do* {
ASSERT (*atm-in-conflict-lookup-pre* (*atm-of L*) *D*); *RETURN* (¬*atm-in-conflict-lookup* (*atm-of L*) *D*) }) (*get-conflict-wl-heur S*)›

lemma *atm-is-in-conflict-st-heur-alt-def*:
‹*atm-is-in-conflict-st-heur* = (λ*L S*. *case* (*get-conflict-wl-heur S*) *of* (-, (-, *D*)) ⇒ *do* {*ASSERT* ((*atm-of L*) < *length D*); *RETURN* (*D* ! (*atm-of L*) = *None*)})›
unfolding *atm-is-in-conflict-st-heur-def* **by** (*auto intro*!: *ext simp*: *atm-in-conflict-lookup-def atm-in-conflict-lookup-pre-def split*:*option.splits*
intro!: *prod.case-cong*)

lemma *all-lits-st-alt-def*: ‹*set-mset* (*all-lits-st S*) = *set-mset* (*all-atms-st S*)›
by (*auto simp*: *all-lits-st-def all-lits-def all-lits-of-mm-union*
in-all-lits-of-mm-ain-atms-of-iff in-$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *all-atms-st-def*
all-atms-def)

lemma *atm-is-in-conflict-st-heur-is-in-conflict-st*:
‹(*uncurry* (*atm-is-in-conflict-st-heur*), *uncurry* (*mop-lit-notin-conflict-wl*)) ∈
[λ(*L*, *S*). *True*]$_f$
Id ×$_r$ *twl-st-heur* → ⟨*Id*⟩ *nres-rel*›
proof −
have *1*: ‹*aaa* ∈# $\mathcal{L}_{all}$ *A* ⟹ *atm-of aaa* ∈ *atms-of* ($\mathcal{L}_{all}$ *A*)› **for** *aaa A*
by (*auto simp*: *atms-of-def*)
show *?thesis*
unfolding *atm-is-in-conflict-st-heur-def twl-st-heur-def option-lookup-clause-rel-def uncurry-def comp-def*
mop-lit-notin-conflict-wl-def all-lits-st-alt-def Let-def
apply (*intro frefI nres-relI*)
apply *refine-rcg*
apply *clarsimp*
subgoal
by (*rule atm-in-conflict-lookup-pre*)
subgoal for *x y x1 x2*
apply (*subst atm-in-conflict-lookup-atm-in-conflict*[*THEN fref-to-Down-unRET-uncurry-Id*, *of* ‹*all-atms-st x2*› ‹*atm-of x1*› ‹*the* (*get-conflict-wl* (*snd y*))›])
apply (*simp add*: $\mathcal{L}_{all}$*-all-atms-all-lits atms-of-def*)[]
apply (*auto simp add*: $\mathcal{L}_{all}$*-all-atms-all-lits atms-of-def option-lookup-clause-rel-def*
ac-simps)[]
apply (*simp add*: *atm-in-conflict-def atm-of-in-atms-of*)
done
done
qed

abbreviation *nat-lit-lit-rel* **where**
‹*nat-lit-lit-rel* ≡ *Id* :: (*nat literal* × -) *set*›

10.4 More theorems

lemma *valid-arena-DECISION-REASON*:
‹*valid-arena arena NU vdom* ⟹ *DECISION-REASON* ∉# *dom-m NU*›
using *arena-lifting*[*of arena NU vdom DECISION-REASON*]
by (*auto simp*: *DECISION-REASON-def SHIFTS-def*)

definition *count-decided-st-heur* :: ‹*isasat* ⇒ -› **where**
‹*count-decided-st-heur S* = *count-decided-pol* (*get-trail-wl-heur S*)›

lemma *count-decided-st-count-decided-st*:
‹(*RETURN o count-decided-st-heur*, *RETURN o count-decided-st*) ∈ *twl-st-heur* →$_f$ ⟨*nat-rel*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *count-decided-st-def twl-st-heur-def count-decided-st-heur-def Let-def*
count-decided-trail-ref[*THEN fref-to-Down-unRET-Id*])

lemma *twl-st-heur-count-decided-st-alt-def*:
fixes *S* :: *isasat*
shows ‹(*S*, *T*) ∈ *twl-st-heur* ⟹ *count-decided-st-heur S* = *count-decided* (*get-trail-wl T*)›
unfolding *count-decided-st-def twl-st-heur-def trail-pol-def*
by (*cases S*) (*auto simp*: *count-decided-st-heur-def count-decided-pol-def*)

lemma *twl-st-heur-isa-length-trail-get-trail-wl*:
fixes *S* :: *isasat*
shows ‹(*S*, *T*) ∈ *twl-st-heur* ⟹ *isa-length-trail* (*get-trail-wl-heur S*) = *length* (*get-trail-wl T*)›
unfolding *isa-length-trail-def twl-st-heur-def trail-pol-def*
by (*cases S*) (*auto dest*: *ann-lits-split-reasons-map-lit-of*)

lemma *trail-pol-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *L* ∈ *trail-pol* $\mathcal{A}$ ⟹ *L* ∈ *trail-pol* $\mathcal{B}$›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
by (*auto simp*: *trail-pol-def ann-lits-split-reasons-def*)

lemma *distinct-atoms-rel-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *L* ∈ *distinct-atoms-rel* $\mathcal{A}$ ⟹ *L* ∈ *distinct-atoms-rel* $\mathcal{B}$›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
unfolding *vmtf-def vmtf-*$\mathcal{L}_{all}$*-def distinct-atoms-rel-def distinct-hash-atoms-rel-def*
atoms-hash-rel-def
by (*auto simp*:)

lemma *phase-saving-rel-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *phase-saving* $\mathcal{A}$ *heur* ⟹ *phase-saving* $\mathcal{B}$ *heur*›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
by (*auto simp*: *phase-save-heur-rel-def phase-saving-def*)

lemma *phase-save-heur-rel-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *phase-save-heur-rel* $\mathcal{A}$ *heur* ⟹ *phase-save-heur-rel* $\mathcal{B}$ *heur*›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
by (*auto simp*: *phase-save-heur-rel-def phase-saving-def*)

lemma *heuristic-rel-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *heuristic-rel* $\mathcal{A}$ *heur* ⟹ *heuristic-rel* $\mathcal{B}$ *heur*›
using *phase-save-heur-rel-cong*[*of* $\mathcal{A}$ $\mathcal{B}$ ‹(λ(-, -, -, -, *a*, -). *a*) (*get-restart-heuristics heur*)›]
using *phase-saving-rel-cong*[*of* $\mathcal{A}$ $\mathcal{B}$ ‹(λ(-, -, -, -, -, -, -, *a*, -). *a*) (*get-restart-heuristics heur*)›]
by (*auto simp*: *heuristic-rel-def heuristic-rel-stats-def*)

lemma *vmtf-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *L* ∈ *vmtf* $\mathcal{A}$ *M* ⟹ *L* ∈ *vmtf* $\mathcal{B}$ *M*›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
unfolding *vmtf-def vmtf-*$\mathcal{L}_{all}$*-def*
by *auto*

lemma *acids-cong*:

‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B} \Longrightarrow L \in$ *acids* $\mathcal{A}$ $M \Longrightarrow L \in$ *acids* $\mathcal{B}$ M›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
unfolding *acids-def vmtf-*$\mathcal{L}_{all}$-*def*
apply (*auto simp*: *distinct-subseteq-iff*)
by (*metis distinct-subseteq-iff*)

lemma *isa-vmtf-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B} \Longrightarrow L \in$ *bump-heur* $\mathcal{A}$ $M \Longrightarrow L \in$ *bump-heur* $\mathcal{B}$ M›
using *vmtf-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *distinct-atoms-rel-cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
acids-cong
apply (*subst* (*asm*) *bump-heur-def*)
apply (*subst bump-heur-def*)
by *blast*

lemma *isa-vmtf-cong'*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B} \Longrightarrow$ *bump-heur* $\mathcal{A}$ = *bump-heur* $\mathcal{B}$›
using *isa-vmtf-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *isa-vmtf-cong*[*of* $\mathcal{B}$ $\mathcal{A}$]
by *blast*

lemma *option-lookup-clause-rel-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B} \Longrightarrow L \in$ *option-lookup-clause-rel* $\mathcal{A} \Longrightarrow L \in$ *option-lookup-clause-rel* $\mathcal{B}$›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
unfolding *option-lookup-clause-rel-def lookup-clause-rel-def*
by (*cases L*) *auto*

lemma D_0-*cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B} \Longrightarrow D_0$ $\mathcal{A}$ = D_0 $\mathcal{B}$›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
by *auto*

lemma *phase-saving-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B} \Longrightarrow$ *phase-saving* $\mathcal{A}$ = *phase-saving* $\mathcal{B}$›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
by (*auto simp*: *phase-saving-def*)

lemma *cach-refinement-empty-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B} \Longrightarrow$ *cach-refinement-empty* $\mathcal{A}$ = *cach-refinement-empty* $\mathcal{B}$›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
by (*force simp*: *cach-refinement-empty-def cach-refinement-alt-def*
distinct-subseteq-iff[*symmetric*] *intro*!: *ext*)

lemma *vdom-m-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B} \Longrightarrow$ *vdom-m* $\mathcal{A}$ *x y* = *vdom-m* $\mathcal{B}$ *x y*›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
by (*auto simp*: *vdom-m-def intro*!: *ext*)

lemma *isasat-input-bounded-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B} \Longrightarrow$ *isasat-input-bounded* $\mathcal{A}$ = *isasat-input-bounded* $\mathcal{B}$›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
by (*auto simp*: *intro*!: *ext*)

lemma *isasat-input-nempty-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B} \Longrightarrow$ *isasat-input-nempty* $\mathcal{A}$ = *isasat-input-nempty* $\mathcal{B}$›
using $\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$-*cong*[*of* $\mathcal{A}$ $\mathcal{B}$]

by (*auto simp*: *intro*!: *ext*)

10.5 Shared Code Equations

definition *clause-not-marked-to-delete* **where**
‹*clause-not-marked-to-delete S C* ⟷ *C* ∈# *dom-m* (*get-clauses-wl S*)›

definition *clause-not-marked-to-delete-pre* **where**
‹*clause-not-marked-to-delete-pre* =
(λ(*S*, *C*). *C* ∈ *vdom-m* (*all-atms-st S*) (*get-watched-wl S*) (*get-clauses-wl S*))›

definition *clause-not-marked-to-delete-heur-pre* **where**
‹*clause-not-marked-to-delete-heur-pre* =
(λ(*S*, *C*). *arena-is-valid-clause-vdom* (*get-clauses-wl-heur S*) *C*)›

definition *clause-not-marked-to-delete-heur* :: ‹- ⇒ *nat* ⇒ *bool*›
where
‹*clause-not-marked-to-delete-heur S C* ⟷
arena-status (*get-clauses-wl-heur S*) *C* ≠ *DELETED*›

lemma *clause-not-marked-to-delete-rel*:
‹(*uncurry* (*RETURN oo clause-not-marked-to-delete-heur*),
uncurry (*RETURN oo clause-not-marked-to-delete*)) ∈
[*clause-not-marked-to-delete-pre*]$_f$
twl-st-heur ×$_f$ *nat-rel* → ⟨*bool-rel*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*use arena-dom-status-iff in-dom-in-vdom* **in**
‹*auto 5 5 simp*: *clause-not-marked-to-delete-def twl-st-heur-def Let-def*
clause-not-marked-to-delete-heur-def arena-dom-status-iff
clause-not-marked-to-delete-pre-def ac-simps›)

definition (**in** −) *access-lit-in-clauses-heur-pre* **where**
‹*access-lit-in-clauses-heur-pre* =
(λ((*S*, *i*), *j*).
arena-lit-pre (*get-clauses-wl-heur S*) (*i*+*j*))›

definition (**in** −) *access-lit-in-clauses-heur* **where**
‹*access-lit-in-clauses-heur S i j* = *arena-lit* (*get-clauses-wl-heur S*) (*i* + *j*)›

definition (**in** −) *mop-access-lit-in-clauses-heur* **where**
‹*mop-access-lit-in-clauses-heur S i j* = *mop-arena-lit2* (*get-clauses-wl-heur S*) *i j*›

lemma *access-lit-in-clauses-heur-fast-pre*:
‹*arena-lit-pre* (*get-clauses-wl-heur a*) (*ba* + *b*) ⟹
isasat-fast a ⟹ *ba* + *b* ≤ *snat64-max*›
by (*auto simp*: *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
dest!: *arena-lifting*(*10*)
dest!: *isasat-fast-length-leD*)[]

lemma $\mathcal{L}_{all}$*-add-mset*:
‹*set-mset* ($\mathcal{L}_{all}$ (*add-mset L C*)) = *insert* (*Pos L*) (*insert* (*Neg L*) (*set-mset* ($\mathcal{L}_{all}$ *C*)))›
by (*auto simp*: $\mathcal{L}_{all}$*-def*)

lemma *correct-watching-dom-watched*:
assumes ‹*correct-watching S*› **and** ‹⋀*C*. *C* ∈# *ran-mf* (*get-clauses-wl S*) ⟹ *C* ≠ []›
shows ‹*set-mset* (*dom-m* (*get-clauses-wl S*)) ⊆
⋃(((‘) *fst*) ‘ *set* ‘ (*get-watched-wl S*) ‘ *set-mset* (*all-lits-st S*))›
(**is** ‹*?A* ⊆ *?B*›)
proof
fix *C*
assume ‹*C* ∈ *?A*›
then obtain *D* **where**
D: ‹*D* ∈# *ran-mf* (*get-clauses-wl S*)› **and**
D': ‹*D* = *get-clauses-wl S* ∝ *C*› **and**
C: ‹*C* ∈# *dom-m* (*get-clauses-wl S*)›
by *auto*
have ‹(*hd D*) ∈# *all-lits-st S*›
using *D D'* *assms*(*2*)[*of D*]
by (*cases S*; *cases D*)
(*auto simp*: *all-lits-def all-lits-st-def all-lits-def*
all-lits-of-mm-add-mset all-lits-of-m-add-mset
dest!: *multi-member-split*)
then show ‹*C* ∈ *?B*›
using *assms*(*1*) *assms*(*2*)[*of D*] *D D'*
multi-member-split[*OF C*]
by (*cases S*; *cases* ‹*get-clauses-wl S* ∝ *C*›;
cases ‹*hd* (*get-clauses-wl S* ∝ *C*)›)
(*auto dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$-*add-mset correct-watching.simps clause-to-update-def*
eq-commute[*of* ‹- # -›] *atm-of-eq-atm-of*
split: *if-splits*
dest!: *arg-cong*[*of* ‹*filter-mset* - -› ‹*add-mset* - -› *set-mset*] *eq-insertD*)
qed

lemma *arena-lit-pre-le-snat64-max*:
‹*length ba* ≤ *snat64-max* ⟹
arena-lit-pre ba a ⟹ *a* ≤ *snat64-max*›
using *arena-lifting*(*10*)[*of ba* - -]
by (*fastforce simp*: *arena-lifting arena-is-valid-clause-idx-def arena-lit-pre-def*
arena-is-valid-clause-idx-and-access-def)

definition *rewatch-heur-vdom* **where**
‹*rewatch-heur-vdom vdom* = *rewatch-heur* (*get-tvdom-aivdom vdom*)›

definition *rewatch-heur-st*
:: ‹*isasat* ⇒ *isasat nres*›
where
‹*rewatch-heur-st* = (λ*S*. *do* {
ASSERT(*length* (*get-tvdom-aivdom* (*get-aivdom S*)) ≤ *length* (*get-clauses-wl-heur S*));
W ← *rewatch-heur* (*get-tvdom-aivdom* (*get-aivdom S*)) (*get-clauses-wl-heur S*) (*get-watched-wl-heur S*);
RETURN (*set-watched-wl-heur W S*)
})›

definition *rewatch-heur-st-fast* **where**
‹*rewatch-heur-st-fast* = *rewatch-heur-st*›

definition *rewatch-heur-st-fast-pre* **where**
‹*rewatch-heur-st-fast-pre S* =
((∀ *x* ∈ *set* (*get-tvdom S*). *x* ≤ *snat64-max*) ∧ *length* (*get-clauses-wl-heur S*) ≤ *snat64-max*)›

definition *rewatch-st* :: ‹'*v twl-st-wl* ⇒ '*v twl-st-wl nres*› **where**
‹*rewatch-st S* = *do*{
(*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*) ← *RETURN S*;
W ← *rewatch N W*;
RETURN ((*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*))
}›

definition *rewatch-heur-and-reorder-st*
:: ‹*isasat* ⇒ *isasat nres*›
where
‹*rewatch-heur-and-reorder-st* = (λ*S*. *do* {
ASSERT(*length* (*get-tvdom-aivdom* (*get-aivdom S*)) ≤ *length* (*get-clauses-wl-heur S*));
W ← *rewatch-heur-and-reorder* (*get-tvdom-aivdom* (*get-aivdom S*)) (*get-clauses-wl-heur S*) (*get-watched-wl-heur S*);
RETURN (*set-watched-wl-heur W S*)
})›

fun *remove-watched-wl* :: ‹'*v twl-st-wl* ⇒ -› **where**
‹*remove-watched-wl* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, -) = (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*)›

lemma *rewatch-st-correctness*:
assumes ‹*get-watched-wl S* = (λ-. [])› **and**
‹⋀*x*. *x* ∈# *dom-m* (*get-clauses-wl S*) ⟹
distinct ((*get-clauses-wl S*) ∝ *x*) ∧ 2 ≤ *length* ((*get-clauses-wl S*) ∝ *x*)›
shows ‹*rewatch-st S* ≤ *SPEC* (λ*T*. *remove-watched-wl S* = *remove-watched-wl T* ∧
correct-watching-init T)›
apply (*rule SPEC-rule-conjI*)
subgoal
using *rewatch-correctness*[*OF assms*]
unfolding *rewatch-st-def*
apply (*cases S*, *case-tac* ‹*rewatch* (*get-clauses-wl S*) (*get-watched-wl S*)›)
by (*auto simp*: *RES-RETURN-RES*)
subgoal
using *rewatch-correctness*[*OF assms*]
unfolding *rewatch-st-def*
apply (*cases S*, *case-tac* ‹*rewatch* (*get-clauses-wl S*) (*get-watched-wl S*)›)
by (*force simp*: *RES-RETURN-RES*)+
done

10.6 Fast to slow conversion

Setup to convert a list from *64 word* to *nat*.

definition *convert-wlists-to-nat-conv* :: ‹'*a list list* ⇒ '*a list list*› **where**
‹*convert-wlists-to-nat-conv* = *id*›

abbreviation *twl-st-heur''*
:: ‹*nat multiset* ⇒ *nat* ⇒ *clss-size* ⇒ (*isasat* × *nat twl-st-wl*) *set*›

where
‹*twl-st-heur''* $\mathcal{D}$ *r lcount* ≡ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur'* $\mathcal{D}$ ∧
length (*get-clauses-wl-heur S*) = *r* ∧ *get-learned-count S* = *lcount*}›

abbreviation *twl-st-heur-up''*
:: ‹*nat multiset* ⇒ *nat* ⇒ *nat* ⇒ *nat literal* ⇒ *clss-size* ⇒ (*isasat* × *nat twl-st-wl*) *set*›
where
‹*twl-st-heur-up''* $\mathcal{D}$ *r s L lcount* ≡ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur''* $\mathcal{D}$ *r lcount* ∧
length (*watched-by T L*) = *s* ∧ *s* ≤ *r*}›

lemma *length-watched-le*:
assumes
prop-inv: ‹*correct-watching x1*› **and**
xb-x'a: ‹(*x1a*, *x1*) ∈ *twl-st-heur''* $\mathcal{D}$*1 r lcount*› **and**
x2: ‹*x2* ∈# $\mathcal{L}_{all}$ (*all-atms-st x1*)›
shows ‹*length* (*watched-by x1 x2*) ≤ *r* − *MIN-HEADER-SIZE*›
proof −
have *dist*: ‹*distinct-watched* (*watched-by x1 x2*)›
using *prop-inv x2* **unfolding** *all-atms-def all-lits-def*
by (*cases x1*; *auto simp*: *correct-watching.simps ac-simps all-lits-st-alt-def*[*symmetric*])
then have *dist*: ‹*distinct-watched* (*watched-by x1 x2*)›
using *xb-x'a*
by (*cases x1*; *auto simp*: $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm correct-watching.simps*)
have *dist-vdom*: ‹*distinct* (*get-vdom x1a*)›
using *xb-x'a*
by (*cases x1*)
(*auto simp*: *twl-st-heur-def twl-st-heur'-def aivdom-inv-dec-alt-def Let-def*)
have *x2*: ‹*x2* ∈# $\mathcal{L}_{all}$ (*all-atms-st x1*)›
using *x2 xb-x'a* **unfolding** *all-atms-def*
by *auto*

have
valid: ‹*valid-arena* (*get-clauses-wl-heur x1a*) (*get-clauses-wl x1*) (*set* (*get-vdom x1a*))›
using *xb-x'a* **unfolding** *all-atms-def all-lits-def*
by (*cases x1*)
(*auto simp*: *twl-st-heur'-def twl-st-heur-def Let-def*)

have ‹*vdom-m* (*all-atms-st x1*) (*get-watched-wl x1*) (*get-clauses-wl x1*) ⊆ *set* (*get-vdom x1a*)›
using *xb-x'a*
by (*cases x1*)
(*auto simp*: *twl-st-heur-def twl-st-heur'-def ac-simps Let-def*)

then have *subset*: ‹*set* (*map fst* (*watched-by x1 x2*)) ⊆ *set* (*get-vdom x1a*)›
using *x2* **unfolding** *vdom-m-def*
by (*cases x1*)
(*force simp*: *twl-st-heur'-def twl-st-heur-def*
dest!: *multi-member-split*)
have *watched-incl*: ‹*mset* (*map fst* (*watched-by x1 x2*)) ⊆# *mset* (*get-vdom x1a*)›
by (*rule distinct-subseteq-iff*[*THEN iffD1*])
(*use dist*[*unfolded distinct-watched-alt-def*] *dist-vdom subset* **in**
‹*simp-all flip*: *distinct-mset-mset-distinct*›)
have *vdom-incl*: ‹*set* (*get-vdom x1a*) ⊆ {*MIN-HEADER-SIZE*..< *length* (*get-clauses-wl-heur x1a*)}›
using *valid-arena-in-vdom-le-arena*[*OF valid*] *arena-dom-status-iff*[*OF valid*] **by** *auto*

have ‹*length* (*get-vdom x1a*) ≤ *length* (*get-clauses-wl-heur x1a*) − *MIN-HEADER-SIZE*›
by (*subst distinct-card*[*OF dist-vdom*, *symmetric*])

```
    (use card-mono[OF - vdom-incl] in auto)
  then show ?thesis
    using size-mset-mono[OF watched-incl] xb-x'a
    by (auto intro!: order-trans[of ‹length (watched-by x1 x2)› ‹length (get-vdom x1a)›])
qed

lemma length-watched-le2:
  assumes
    prop-inv: ‹correct-watching-except i j L x1› and
    xb-x'a: ‹(x1a, x1) ∈ twl-st-heur'' 𝒟1 r lcount› and
    x2: ‹x2 ∈# all-lits-st x1› and diff: ‹L ≠ x2›
  shows ‹length (watched-by x1 x2) ≤ r − MIN-HEADER-SIZE›
proof −
  from prop-inv diff have dist: ‹distinct-watched (watched-by x1 x2)›
    using x2 unfolding all-atms-def all-lits-def
    by (cases x1; auto simp: ℒ_all-atm-of-all-lits-of-mm correct-watching-except.simps ac-simps)
  then have dist: ‹distinct-watched (watched-by x1 x2)›
    using xb-x'a
    by (cases x1; auto simp: ℒ_all-atm-of-all-lits-of-mm correct-watching.simps)
  have dist-vdom: ‹distinct (get-vdom x1a)›
    using xb-x'a
    by (cases x1)
      (auto simp: twl-st-heur-def twl-st-heur'-def aivdom-inv-dec-alt-def)

  have
      valid: ‹valid-arena (get-clauses-wl-heur x1a) (get-clauses-wl x1) (set (get-vdom x1a))›
    using xb-x'a unfolding all-atms-def all-lits-def
    by (cases x1)
     (auto simp: twl-st-heur'-def twl-st-heur-def)

  have ‹vdom-m (all-atms-st x1) (get-watched-wl x1) (get-clauses-wl x1) ⊆ set (get-vdom x1a)›
    using xb-x'a
    by (cases x1)
      (auto simp: twl-st-heur-def twl-st-heur'-def ac-simps simp flip: all-atms-def)
  then have subset: ‹set (map fst (watched-by x1 x2)) ⊆ set (get-vdom x1a)›
    using x2 unfolding vdom-m-def all-lits-st-alt-def[symmetric]
    by (cases x1)
      (force simp: twl-st-heur'-def twl-st-heur-def ac-simps simp flip: all-atms-def all-lits-alt-def2
        dest!: multi-member-split)
  have watched-incl: ‹mset (map fst (watched-by x1 x2)) ⊆# mset (get-vdom x1a)›
    by (rule distinct-subseteq-iff[THEN iffD1])
      (use dist[unfolded distinct-watched-alt-def] dist-vdom subset in
        ‹simp-all flip: distinct-mset-mset-distinct›)
  have vdom-incl: ‹set (get-vdom x1a) ⊆ {MIN-HEADER-SIZE..< length (get-clauses-wl-heur x1a)}›
    using valid-arena-in-vdom-le-arena[OF valid] arena-dom-status-iff[OF valid] by auto

  have ‹length (get-vdom x1a) ≤ length (get-clauses-wl-heur x1a) − MIN-HEADER-SIZE›
    by (subst distinct-card[OF dist-vdom, symmetric])
      (use card-mono[OF - vdom-incl] in auto)
  then show ?thesis
    using size-mset-mono[OF watched-incl] xb-x'a
    by (auto intro!: order-trans[of ‹length (watched-by x1 x2)› ‹length (get-vdom x1a)›])
qed

lemma atm-of-all-lits-of-m: ‹atm-of `# (all-lits-of-m C) = atm-of `# C + atm-of `# C›
  ‹atm-of ` set-mset (all-lits-of-m C) = atm-of `set-mset C ›
```

by (*induction C*; *auto simp*: *all-lits-of-m-add-mset*)+

find-theorems $\mathcal{L}_{all}$ *all-lits-st*

lemma *mop-watched-by-app-heur-mop-watched-by-at*:
‹(*uncurry2 mop-watched-by-app-heur*, *uncurry2 mop-watched-by-at*) ∈
twl-st-heur ×$_f$ *nat-lit-lit-rel* ×$_f$ *nat-rel* →$_f$ ⟨*Id*⟩*nres-rel*›
unfolding *mop-watched-by-app-heur-def mop-watched-by-at-def uncurry-def all-lits-def*[*symmetric*]
all-lits-alt-def[*symmetric*]
by (*intro frefI nres-relI*, *refine-rcg*)
(*auto simp*: *twl-st-heur-def map-fun-rel-def all-lits-st-alt-def*[*symmetric*])

lemma *mop-watched-by-app-heur-mop-watched-by-at''*:
‹(*uncurry2 mop-watched-by-app-heur*, *uncurry2 mop-watched-by-at*) ∈
twl-st-heur-up'' $\mathcal{D}$ *r s K lcount* ×$_f$ *nat-lit-lit-rel* ×$_f$ *nat-rel* →$_f$ ⟨*Id*⟩*nres-rel*›
by (*rule fref-mono*[*THEN set-mp*, *OF - - - mop-watched-by-app-heur-mop-watched-by-at*])
(*auto simp*: $\mathcal{L}_{all}$-*all-atms-all-lits twl-st-heur'-def map-fun-rel-def*)

definition *polarity-st-pre* :: ‹*nat twl-st-wl* × *nat literal* ⇒ *bool*› **where**
‹*polarity-st-pre* ≡ λ(*S*, *L*). *L* ∈# $\mathcal{L}_{all}$ (*all-atms-st S*)›

definition *mop-polarity-st-heur* :: ‹*isasat* ⇒ *nat literal* ⇒ *bool option nres*› **where**
‹*mop-polarity-st-heur S L* = *do* {
mop-polarity-pol (*get-trail-wl-heur S*) *L*
}›

lemma *mop-polarity-st-heur-mop-polarity-wl*:
‹(*uncurry mop-polarity-st-heur*, *uncurry mop-polarity-wl*) ∈
[λ-. *True*]$_f$ *twl-st-heur* ×$_r$ *Id* → ⟨⟨*bool-rel*⟩*option-rel*⟩*nres-rel*›
unfolding *mop-polarity-wl-def mop-polarity-st-heur-def uncurry-def mop-polarity-pol-def*
apply (*intro frefI nres-relI*)
apply (*refine-rcg polarity-pol-polarity*[*of* ‹*all-atms-st* -›, *THEN fref-to-Down-unRET-uncurry*])
apply (*auto simp*: *twl-st-heur-def* $\mathcal{L}_{all}$-*all-atms-all-lits ac-simps all-lits-st-alt-def*[*symmetric*]
intro!: *polarity-pol-pre simp flip*: *all-atms-def*)
done

lemma *mop-polarity-st-heur-mop-polarity-wl''*:
‹(*uncurry mop-polarity-st-heur*, *uncurry mop-polarity-wl*) ∈
[λ-. *True*]$_f$ *twl-st-heur-up''* $\mathcal{D}$ *r s K lcount* ×$_r$ *Id* → ⟨⟨*bool-rel*⟩*option-rel*⟩*nres-rel*›
by (*rule fref-mono*[*THEN set-mp*, *OF - - - mop-polarity-st-heur-mop-polarity-wl*])
(*auto simp*: $\mathcal{L}_{all}$-*all-atms-all-lits twl-st-heur'-def map-fun-rel-def*)

lemma [*simp*,*iff*]: ‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st S*) *S* ⟷ *blits-in-*$\mathcal{L}_{in}$ *S*›
unfolding *literals-are-*$\mathcal{L}_{in}$*-def is-*$\mathcal{L}_{all}$*-def* $\mathcal{L}_{all}$-*all-atms-all-lits all-lits-st-alt-def*[*symmetric*]
by *auto*

definition *length-avdom* :: ‹*isasat* ⇒ *nat*› **where**
‹*length-avdom S* = *length* (*get-avdom S*)›

definition *length-ivdom* :: ‹*isasat* ⇒ *nat*› **where**
‹*length-ivdom S* = *length* (*get-ivdom S*)›

definition *length-tvdom* :: ‹*isasat* ⇒ *nat*› **where**
‹*length-tvdom S* = *length* (*get-tvdom S*)›

definition *clause-is-learned-heur* :: ‹*isasat* ⇒ *nat* ⇒ *bool*›
where
‹*clause-is-learned-heur S C* ⟷ *arena-status* (*get-clauses-wl-heur S*) *C* = *LEARNED*›

definition *get-the-propagation-reason-heur*
:: ‹*isasat* ⇒ *nat literal* ⇒ *nat option nres*›
where
‹*get-the-propagation-reason-heur S* = *get-the-propagation-reason-pol* (*get-trail-wl-heur S*)›

definition *clause-lbd-heur* :: ‹*isasat* ⇒ *nat* ⇒ *nat*›
where
‹*clause-lbd-heur S C* = *arena-lbd* (*get-clauses-wl-heur S*) *C*›

definition (**in** −) *access-length-heur* **where**
‹*access-length-heur S i* = *arena-length* (*get-clauses-wl-heur S*) *i*›

definition *marked-as-used-st* **where**
‹*marked-as-used-st T C* =
marked-as-used (*get-clauses-wl-heur T*) *C*›

definition *access-avdom-at* :: ‹*isasat* ⇒ *nat* ⇒ *nat*› **where**
‹*access-avdom-at S i* = *get-avdom S* ! *i*›

definition *access-ivdom-at* :: ‹*isasat* ⇒ *nat* ⇒ *nat*› **where**
‹*access-ivdom-at S i* = *get-ivdom S* ! *i*›

definition *access-tvdom-at* :: ‹*isasat* ⇒ *nat* ⇒ *nat*› **where**
‹*access-tvdom-at S i* = *get-tvdom S* ! *i*›

definition *access-avdom-at-pre* **where**
‹*access-avdom-at-pre S i* ⟷ *i* < *length* (*get-avdom S*)›

definition *access-ivdom-at-pre* **where**
‹*access-ivdom-at-pre S i* ⟷ *i* < *length* (*get-ivdom S*)›

definition *access-tvdom-at-pre* **where**
‹*access-tvdom-at-pre S i* ⟷ *i* < *length* (*get-tvdom S*)›

definition *mark-garbage-heur* :: ‹*nat* ⇒ *nat* ⇒ *isasat* ⇒ *isasat*› **where**
‹*mark-garbage-heur C i* = (λ*S*.
let N′ = *extra-information-mark-to-delete* (*get-clauses-wl-heur S*) *C in*
let lcount = *clss-size-decr-lcount* (*get-learned-count S*) *in*
let vdom = *remove-inactive-aivdom i* (*get-aivdom S*) *in*
set-aivdom-wl-heur vdom (*set-clauses-wl-heur N′* (*set-learned-count-wl-heur lcount S*)))›

definition *mark-garbage-heur2* :: ‹*nat* ⇒ *isasat* ⇒ *isasat nres*› **where**
‹*mark-garbage-heur2 C* = (λ*S*. *do*{
ASSERT (*mark-garbage-pre* (*get-clauses-wl-heur S*, *C*));
let N′ = *get-clauses-wl-heur S*;
let st = *arena-status N′ C* = *IRRED*;
let N′ = *extra-information-mark-to-delete N′ C*;
let lcount = *get-learned-count S*;
ASSERT(¬*st* ⟶ *clss-size-lcount lcount* ≥ *1*);

```
    let lcount = (if st then lcount else clss-size-decr-lcount lcount);
    RETURN (set-clauses-wl-heur N' (set-learned-count-wl-heur lcount S))})›

definition mark-garbage-heur3 :: ‹nat ⇒ nat ⇒ isasat ⇒ isasat› where
  ‹mark-garbage-heur3 C i = (λS.
    let N' = get-clauses-wl-heur S in
    let N' = extra-information-mark-to-delete N' C in
    let lcount = get-learned-count S in
    let vdom = get-aivdom S in
    let vdom = remove-inactive-aivdom-tvdom i vdom in
    let lcount = clss-size-decr-lcount lcount in
    let S = set-clauses-wl-heur N' S in
    let S = set-learned-count-wl-heur lcount S in
    let S = set-aivdom-wl-heur vdom S in
    S)›

definition mark-garbage-heur4 :: ‹nat ⇒ isasat ⇒ isasat nres› where
  ‹mark-garbage-heur4 C S = (do{
    let N' = get-clauses-wl-heur S;
    let st = arena-status N' C = IRRED;
    let N' = extra-information-mark-to-delete (N') C;
    let lcount = get-learned-count S;
    ASSERT(¬st ⟶ clss-size-lcount lcount ≥ 1);
    let lcount = (if st then lcount else clss-size-incr-lcountUEk (clss-size-decr-lcount lcount));
    let stats = get-stats-heur S;
    let stats = (if st then decr-irred-clss stats else stats);
    let S = set-clauses-wl-heur N' S;
    let S = set-learned-count-wl-heur lcount S;
    let S = set-stats-wl-heur stats S;
    RETURN S
   })›

definition delete-index-vdom-heur :: ‹nat ⇒ isasat ⇒ isasat› where
  ‹delete-index-vdom-heur = (λi S.
    let vdom = get-aivdom S in
    let vdom = remove-inactive-aivdom-tvdom i vdom in
    let S = set-aivdom-wl-heur vdom S in
    S)›

lemma arena-act-pre-mark-used:
  ‹arena-act-pre arena C ⟹
  arena-act-pre (mark-unused arena C) C›
  unfolding arena-act-pre-def arena-is-valid-clause-idx-def
  apply clarify
  apply (rule-tac x=N in exI)
  apply (rule-tac x=vdom in exI)
  by (auto simp: arena-act-pre-def
    simp: valid-arena-mark-unused)

definition mop-mark-garbage-heur :: ‹nat ⇒ nat ⇒ isasat ⇒ isasat nres› where
  ‹mop-mark-garbage-heur C i = (λS. do {
     ASSERT(mark-garbage-pre (get-clauses-wl-heur S, C) ∧ clss-size-lcount (get-learned-count S) ≥ 1
∧ i < length (get-avdom S));
    RETURN (mark-garbage-heur C i S)
  })›
```

definition *mop-mark-garbage-heur3* :: ‹*nat* ⇒ *nat* ⇒ *isasat* ⇒ *isasat nres*› **where**
‹*mop-mark-garbage-heur3 C i* = (λ*S*. *do* {
ASSERT(*mark-garbage-pre* (*get-clauses-wl-heur S*, *C*) ∧ *clss-size-lcount* (*get-learned-count S*) ≥ *1* ∧ *i* < *length* (*get-tvdom S*));
RETURN (*mark-garbage-heur3 C i S*)
})›

definition *mark-unused-st-heur* :: ‹*nat* ⇒ *isasat* ⇒ *isasat*› **where**
‹*mark-unused-st-heur C* = (λ*S*.
let N' = *mark-unused* (*get-clauses-wl-heur S*) *C in*
let S = *set-clauses-wl-heur N' S in*
S)›

definition *mop-mark-unused-st-heur* :: ‹*nat* ⇒ *isasat* ⇒ *isasat nres*› **where**
‹*mop-mark-unused-st-heur C T* = *do* {
ASSERT(*arena-act-pre* (*get-clauses-wl-heur T*) *C*);
RETURN (*mark-unused-st-heur C T*)
}›

lemma *mark-unused-st-heur-simp*[*simp*]:
‹*get-avdom* (*mark-unused-st-heur C T*) = *get-avdom T*›
‹*get-vdom* (*mark-unused-st-heur C T*) = *get-vdom T*›
‹*get-ivdom* (*mark-unused-st-heur C T*) = *get-ivdom T*›
‹*get-tvdom* (*mark-unused-st-heur C T*) = *get-tvdom T*›
by (*cases T*; *auto simp*: *mark-unused-st-heur-def*; *fail*)+

fun *get-conflict-count-since-last-restart-heur* :: ‹*isasat* ⇒ *64 word*› **where**
‹*get-conflict-count-since-last-restart-heur S* = *get-conflict-count-since-last-restart* (*get-heur S*)›

definition *get-global-conflict-count* **where**
‹*get-global-conflict-count S* = *stats-conflicts* (*get-stats-heur S*)›

I also played with *ema-reinit fast-ema* and *ema-reinit slow-ema*. Currently removed, to test the performance, I remove it.

definition *incr-restart-stat* :: ‹*isasat* ⇒ *isasat nres*› **where**
‹*incr-restart-stat* = (λ*S*. *do*{
let heur = *get-heur S*;
let heur = *unset-fully-propagated-heur* (*heuristic-reluctant-untrigger* (*restart-info-restart-done-heur heur*));
let S = *set-heur-wl-heur heur S*;
let stats = *get-stats-heur S*;
let S = *set-stats-wl-heur* (*incr-restart* (*stats*)) *S*;
RETURN S
})›

definition *incr-reduction-stat* :: ‹*isasat* ⇒ *isasat nres*› **where**
‹*incr-reduction-stat* = (λ*S*. *do*{
let stats = *get-stats-heur S*;
let stats = *incr-reduction stats*;
let S = *set-stats-wl-heur stats S*;
RETURN S
})›

definition *incr-wasted-st* :: ‹*64 word* ⇒ *isasat* ⇒ *isasat*› **where**
‹*incr-wasted-st* = (λ*waste S*. *do*{

let heur = get-heur S in
let heur = incr-wasted waste heur in
let S = set-heur-wl-heur heur S in S
})›

definition *wasted-bytes-st* :: ‹*isasat ⇒ 64 word*› **where**
‹*wasted-bytes-st S = wasted-of* (*get-heur S*)›

definition *opts-restart-st* :: ‹*isasat ⇒ bool*› **where**
‹*opts-restart-st S = opts-restart* (*get-opts S*)›

definition *opts-reduction-st* :: ‹*isasat ⇒ bool*› **where**
‹*opts-reduction-st S = opts-reduce* (*get-opts S*)›

definition *opts-unbounded-mode-st* :: ‹*isasat ⇒ bool*› **where**
‹*opts-unbounded-mode-st S = opts-unbounded-mode* (*get-opts S*)›

definition *opts-minimum-between-restart-st* :: ‹*isasat ⇒ 64 word*› **where**
‹*opts-minimum-between-restart-st S = opts-minimum-between-restart* (*get-opts S*)›

definition *opts-restart-coeff1-st* :: ‹*isasat ⇒ 64 word*› **where**
‹*opts-restart-coeff1-st S = opts-restart-coeff1* (*get-opts S*)›

definition *opts-restart-coeff2-st* :: ‹*isasat ⇒ nat*› **where**
‹*opts-restart-coeff2-st S = opts-restart-coeff2* (*get-opts S*)›

definition *isasat-length-trail-st* :: ‹*isasat ⇒ nat*› **where**
‹*isasat-length-trail-st S = isa-length-trail* (*get-trail-wl-heur S*)›

definition *mop-isasat-length-trail-st* :: ‹*isasat ⇒ nat nres*› **where**
‹*mop-isasat-length-trail-st S = do* {
ASSERT(*isa-length-trail-pre* (*get-trail-wl-heur S*));
RETURN (*isa-length-trail* (*get-trail-wl-heur S*))
}›

definition *get-pos-of-level-in-trail-imp-st* :: ‹*isasat ⇒ nat ⇒ nat nres*› **where**
‹*get-pos-of-level-in-trail-imp-st S = get-pos-of-level-in-trail-imp* (*get-trail-wl-heur S*)›

definition *mop-clause-not-marked-to-delete-heur* :: ‹- ⇒ *nat ⇒ bool nres*›
where
‹*mop-clause-not-marked-to-delete-heur S C = do* {
ASSERT(*clause-not-marked-to-delete-heur-pre* (*S*, *C*));
RETURN (*clause-not-marked-to-delete-heur S C*)
}›

definition *mop-arena-lbd-st* **where**
‹*mop-arena-lbd-st S* =
mop-arena-lbd (*get-clauses-wl-heur S*)›

definition *mop-arena-status-st* :: ‹*isasat ⇒* -› **where**
‹*mop-arena-status-st S* =
mop-arena-status (*get-clauses-wl-heur S*)›

definition *mop-marked-as-used-st* :: ‹*isasat ⇒ nat ⇒ nat nres*› **where**

‹*mop-marked-as-used-st S* =
mop-marked-as-used (*get-clauses-wl-heur S*)›

definition *mop-arena-length-st* :: ‹*isasat* ⇒ *nat* ⇒ *nat nres*› **where**
‹*mop-arena-length-st S* =
mop-arena-length (*get-clauses-wl-heur S*)›

definition *full-arena-length-st* :: ‹*isasat* ⇒ *nat*› **where**
‹*full-arena-length-st S* = *length* (*get-clauses-wl-heur S*)›

definition (**in** −) *access-lit-in-clauses* **where**
‹*access-lit-in-clauses S i j* = (*get-clauses-wl S*) ∝ *i* ! *j*›

lemma *twl-st-heur-get-clauses-access-lit*[*simp*]:
‹(*S*, *T*) ∈ *twl-st-heur* ⟹ *C* ∈# *dom-m* (*get-clauses-wl T*) ⟹
i < *length* (*get-clauses-wl T* ∝ *C*) ⟹
get-clauses-wl T ∝ *C* ! *i* = *access-lit-in-clauses-heur S C i*›
for *S T C i*
by (*cases S*; *cases T*)
(*auto simp*: *arena-lifting twl-st-heur-def access-lit-in-clauses-heur-def*)

abbreviation *length-clauses-heur* **where**
‹*length-clauses-heur* ≡ *full-arena-length-st*›

lemmas *length-clauses-heur-def* = *full-arena-length-st-def*

In an attempt to avoid using *?a + ?b + ?c = ?a + (?b + ?c)*
?a + ?b = ?b + ?a
?b + (?a + ?c) = ?a + (?b + ?c)
*?a * ?b * ?c = ?a * (?b * ?c)*
*?a * ?b = ?b * ?a*
*?b * (?a * ?c) = ?a * (?b * ?c)*
((?a ∧ ?b) ∧ ?c) = (?a ∧ ?b ∧ ?c)
(?a ∧ ?b) = (?b ∧ ?a)
(?b ∧ ?a ∧ ?c) = (?a ∧ ?b ∧ ?c)
((?a ∨ ?b) ∨ ?c) = (?a ∨ ?b ∨ ?c)
(?a ∨ ?b) = (?b ∨ ?a)
(?b ∨ ?a ∨ ?c) = (?a ∨ ?b ∨ ?c)
inf (inf ?a ?b) ?c = inf ?a (inf ?b ?c)
inf ?a ?b = inf ?b ?a
inf ?b (inf ?a ?c) = inf ?a (inf ?b ?c)
sup (sup ?a ?b) ?c = sup ?a (sup ?b ?c)
sup ?a ?b = sup ?b ?a
sup ?b (sup ?a ?c) = sup ?a (sup ?b ?c)
min (min ?a ?b) ?c = min ?a (min ?b ?c)
min ?a ?b = min ?b ?a
min ?b (min ?a ?c) = min ?a (min ?b ?c)
max (max ?a ?b) ?c = max ?a (max ?b ?c)

$max\ ?a\ ?b = max\ ?b\ ?a$
$max\ ?b\ (max\ ?a\ ?c) = max\ ?a\ (max\ ?b\ ?c)$
$coprime\ ?b\ ?a = coprime\ ?a\ ?b$
$(?a\ dvd\ ?c - ?b) = (?a\ dvd\ ?b - ?c)$
$(?a\ @\ ?b)\ @\ ?c = ?a\ @\ ?b\ @\ ?c$
$(?a\ AND\ ?b)\ AND\ ?c = ?a\ AND\ ?b\ AND\ ?c$
$?a\ AND\ ?b = ?b\ AND\ ?a$
$?b\ AND\ ?a\ AND\ ?c = ?a\ AND\ ?b\ AND\ ?c$
$(?a\ OR\ ?b)\ OR\ ?c = ?a\ OR\ ?b\ OR\ ?c$
$?a\ OR\ ?b = ?b\ OR\ ?a$
$?b\ OR\ ?a\ OR\ ?c = ?a\ OR\ ?b\ OR\ ?c$
$(?a\ XOR\ ?b)\ XOR\ ?c = ?a\ XOR\ ?b\ XOR\ ?c$
$?a\ XOR\ ?b = ?b\ XOR\ ?a$
$?b\ XOR\ ?a\ XOR\ ?c = ?a\ XOR\ ?b\ XOR\ ?c$
$gcd\ (gcd\ ?a\ ?b)\ ?c = gcd\ ?a\ (gcd\ ?b\ ?c)$
$gcd\ ?a\ ?b = gcd\ ?b\ ?a$
$gcd\ ?b\ (gcd\ ?a\ ?c) = gcd\ ?a\ (gcd\ ?b\ ?c)$
$lcm\ (lcm\ ?a\ ?b)\ ?c = lcm\ ?a\ (lcm\ ?b\ ?c)$
$lcm\ ?a\ ?b = lcm\ ?b\ ?a$
$lcm\ ?b\ (lcm\ ?a\ ?c) = lcm\ ?a\ (lcm\ ?b\ ?c)$
$signed.min\ (signed.min\ ?a\ ?b)\ ?c = signed.min\ ?a\ (signed.min\ ?b\ ?c)$
$signed.min\ ?a\ ?b = signed.min\ ?b\ ?a$
$signed.min\ ?b\ (signed.min\ ?a\ ?c) = signed.min\ ?a\ (signed.min\ ?b\ ?c)$
$signed.max\ (signed.max\ ?a\ ?b)\ ?c = signed.max\ ?a\ (signed.max\ ?b\ ?c)$
$signed.max\ ?a\ ?b = signed.max\ ?b\ ?a$
$signed.max\ ?b\ (signed.max\ ?a\ ?c) = signed.max\ ?a\ (signed.max\ ?b\ ?c)$
$?a \cap\# \ ?b \cap\# \ ?c = ?a \cap\# \ (?b \cap\# \ ?c)$
$?a \cap\# \ ?b = ?b \cap\# \ ?a$
$?b \cap\# \ (?a \cap\# \ ?c) = ?a \cap\# \ (?b \cap\# \ ?c)$
$?a \cup\# \ ?b \cup\# \ ?c = ?a \cup\# \ (?b \cup\# \ ?c)$
$?a \cup\# \ ?b = ?b \cup\# \ ?a$
$?b \cup\# \ (?a \cup\# \ ?c) = ?a \cup\# \ (?b \cup\# \ ?c)$
$((?a \wedge* \ ?b) \wedge* \ ?c) = (?a \wedge* \ ?b \wedge* \ ?c)$
$(?a \wedge* \ ?b) = (?b \wedge* \ ?a)$
$(?b \wedge* \ ?a \wedge* \ ?c) = (?a \wedge* \ ?b \wedge* \ ?c)$ everywhere.

lemma *all-lits-simps*[*simp*]:
  ‹*all-lits N* ((*NE* + *UE*) + (*NS* + *US*)) = *all-lits N* (*NE* + *UE* + *NS* + *US*)›
  ‹*all-atms N* ((*NE* + *UE*) + (*NS* + *US*)) = *all-atms N* (*NE* + *UE* + *NS* + *US*)›
  by (*auto simp*: *ac-simps*)

lemma *learned-clss-count-twl-st-heur*: ‹(*T*, *Ta*) ∈ *twl-st-heur* ⟹
  learned-clss-count T=
  size (*get-learned-clss-wl Ta*) +
  size (*get-unit-learned-clss-wl Ta*) +

size (*get-subsumed-learned-clauses-wl Ta*) +
size (*get-learned-clauses0-wl Ta*)›
by (*auto simp*: *twl-st-heur-def clss-size-def learned-clss-count-def clss-size-corr-def clss-size-lcount-def clss-size-lcountUE-def clss-size-lcountUS-def clss-size-lcountUEk-def get-learned-clss-wl-def clss-size-lcountU0-def get-unit-learned-clss-wl-alt-def*)

lemma *clss-size-allcount-alt-def*:
‹*clss-size-allcount S* = *clss-size-lcountUS S* + *clss-size-lcountU0 S* + *clss-size-lcountUE S* + *clss-size-lcountUEk S* + *clss-size-lcount S*›
by (*cases S*) (*auto simp*: *clss-size-allcount-def clss-size-lcountUS-def clss-size-lcount-def clss-size-lcountUEk-def clss-size-lcountUE-def clss-size-lcountU0-def*)

definition *isasat-trail-nth-st* :: ‹*isasat* ⇒ *nat* ⇒ *nat literal nres*› **where**
‹*isasat-trail-nth-st S i* = *isa-trail-nth* (*get-trail-wl-heur S*) *i*›

definition *get-the-propagation-reason-pol-st* :: ‹*isasat*⇒ *nat literal* ⇒ *nat option nres*› **where**
‹*get-the-propagation-reason-pol-st S i* = *get-the-propagation-reason-pol* (*get-trail-wl-heur S*) *i*›

definition *empty-US-heur* :: ‹*isasat* ⇒ *isasat*› **where**
‹*empty-US-heur S* =
(*let lcount* = *get-learned-count S in*
let lcount = *clss-size-resetUS0 lcount in*
let S = *set-learned-count-wl-heur lcount S in S*
)›

lemma *get-clauses-wl-heur-empty-US*[*simp*]:
‹*get-clauses-wl-heur* (*empty-US-heur xc*) = *get-clauses-wl-heur xc*› **and**
get-vdom-empty-US[*simp*]:
‹*get-vdom* (*empty-US-heur xc*) = *get-vdom xc*›
‹*get-avdom* (*empty-US-heur xc*) = *get-avdom xc*›
‹*get-ivdom* (*empty-US-heur xc*) = *get-ivdom xc*›
‹*get-tvdom* (*empty-US-heur xc*) = *get-tvdom xc*›
by (*cases xc*; *auto simp*: *empty-US-heur-def*; *fail*)+

definition *empty-Q-wl* :: ‹'*v twl-st-wl* ⇒ '*v twl-st-wl*› **where**
‹*empty-Q-wl* = (λ(*M'*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, -, *W*). (*M'*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, {#}, *N0*, {#}, {#}, *W*))›

definition *empty-Q-wl2* :: ‹'*v twl-st-wl* ⇒ '*v twl-st-wl*› **where**
‹*empty-Q-wl2* = (λ(*M'*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, -, *W*). (*M'*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, {#}, *W*))›

definition *empty-US-heur-wl* :: ‹'*v twl-st-wl* ⇒ '*v twl-st-wl*› **where**
‹*empty-US-heur-wl* = (λ(*M'*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*). (*M'*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, {#}, *N0*, {#}, *Q*, *W*))›

lemma *restart-info-of-stats-simp* [*simp*]: ‹*restart-info-of-stats* (*incr-wasted-stats C heur*) = *restart-info-of-stats heur*›
by (*cases heur*; *auto*; *fail*)+

lemma *incr-wasted-st-twl-st*[*simp*]:
‹*get-aivdom* (*incr-wasted-st b T*) = *get-aivdom T*›
‹*get-avdom* (*incr-wasted-st w T*) = *get-avdom T*›
‹*get-vdom* (*incr-wasted-st w T*) = *get-vdom T*›
‹*get-ivdom* (*incr-wasted-st w T*) = *get-ivdom T*›
‹*get-tvdom* (*incr-wasted-st w T*) = *get-tvdom T*›

```
  ‹get-trail-wl-heur (incr-wasted-st w T) = get-trail-wl-heur T›
  ‹get-clauses-wl-heur (incr-wasted-st C T) = get-clauses-wl-heur T›
  ‹get-conflict-wl-heur (incr-wasted-st C T) = get-conflict-wl-heur T›
  ‹get-learned-count (incr-wasted-st C T) = get-learned-count T›
  ‹get-conflict-count-heur (incr-wasted-st C T) = get-conflict-count-heur T›
  ‹literals-to-update-wl-heur (incr-wasted-st C T) = literals-to-update-wl-heur T›
  ‹get-watched-wl-heur (incr-wasted-st C T) = get-watched-wl-heur T›
  ‹get-vmtf-heur (incr-wasted-st C T) = get-vmtf-heur T›
  ‹get-count-max-lvls-heur (incr-wasted-st C T) = get-count-max-lvls-heur T›
  ‹get-conflict-cach (incr-wasted-st C T) = get-conflict-cach T›
  ‹get-lbd (incr-wasted-st C T) = get-lbd T›
  ‹get-outlearned-heur (incr-wasted-st C T) = get-outlearned-heur T›
  ‹get-aivdom (incr-wasted-st C T) = get-aivdom T›
  ‹get-learned-count (incr-wasted-st C T) = get-learned-count T›
  ‹get-opts (incr-wasted-st C T) = get-opts T›
  ‹get-old-arena (incr-wasted-st C T) = get-old-arena T›
  by (cases T; auto simp: incr-wasted-st-def incr-wasted-def restart-info-of-def; fail)+

definition heuristic-reluctant-triggered2-st :: ‹isasat ⇒ bool› where
  ‹heuristic-reluctant-triggered2-st S = heuristic-reluctant-triggered2 (get-heur S)›

definition heuristic-reluctant-untrigger-st :: ‹isasat ⇒ isasat› where
  ‹heuristic-reluctant-untrigger-st S =
  (let heur = get-heur S;
    heur = heuristic-reluctant-untrigger heur;
    S = set-heur-wl-heur heur S in
  S)›

lemma twl-st-heur''D-twl-st-heurD:
  assumes H: ‹(⋀𝒟 r. f ∈ twl-st-heur'' 𝒟 r lcount →f ⟨twl-st-heur'' 𝒟 r lcount⟩ nres-rel)›
  shows ‹f ∈ {(S, T). (S, T) ∈ twl-st-heur ∧ get-learned-count S = lcount} →f
        ⟨{(S, T). (S, T) ∈ twl-st-heur ∧ get-learned-count S = lcount}⟩ nres-rel›  (is ‹- ∈ ?A B›)
proof −
  obtain f1 f2 where f: ‹f = (f1, f2)›
    by (cases f) auto
  show ?thesis
    unfolding f
    apply (simp only: fref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp)
    apply (intro conjI impI allI)
    subgoal for x y
      using assms[of ‹dom-m (get-clauses-wl y)› ‹length (get-clauses-wl-heur x)›,
        unfolded twl-st-heur'-def nres-rel-def in-pair-collect-simp f,
        rule-format] unfolding f
      apply (simp only: fref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp)
      apply (drule spec[of - x])
      apply (drule spec[of - y])
      apply simp
      apply (rule weaken-⇓'[of - ‹twl-st-heur'' (dom-m (get-clauses-wl y))
        (length (get-clauses-wl-heur x)) lcount›])
      apply (fastforce simp: twl-st-heur'-def)+
      done
    done
qed


lemma twl-st-heur'''D-twl-st-heurD:
```

```
  assumes H: ‹(⋀r. f ∈ twl-st-heur''' r →f ⟨twl-st-heur''' r⟩ nres-rel)›
  shows ‹f ∈ twl-st-heur →f ⟨twl-st-heur⟩ nres-rel› (is ‹- ∈ ?A B›)
proof −
  obtain f1 f2 where f: ‹f = (f1, f2)›
    by (cases f) auto
  show ?thesis
    unfolding f
    apply (simp only: fref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp)
    apply (intro conjI impI allI)
    subgoal for x y
      using assms[of ‹length (get-clauses-wl-heur x)›,
        unfolded twl-st-heur'-def nres-rel-def in-pair-collect-simp f,
        rule-format] unfolding f
      apply (simp only: fref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp)
      apply (drule spec[of - x])
      apply (drule spec[of - y])
      apply simp
      apply (rule weaken-⇓'[of - ‹twl-st-heur''' (length (get-clauses-wl-heur x))›])
      apply (fastforce simp: twl-st-heur'-def)+
      done
    done
qed


lemma twl-st-heur'''D-twl-st-heurD-prod:
  assumes H: ‹(⋀r. f ∈ twl-st-heur''' r →f ⟨A ×r twl-st-heur''' r⟩ nres-rel)›
  shows ‹f ∈ twl-st-heur →f ⟨A ×r twl-st-heur⟩ nres-rel› (is ‹- ∈ ?A B›)
proof −
  obtain f1 f2 where f: ‹f = (f1, f2)›
    by (cases f) auto
  show ?thesis
    unfolding f
    apply (simp only: fref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp)
    apply (intro conjI impI allI)
    subgoal for x y
      using assms[of ‹length (get-clauses-wl-heur x)›,
        unfolded twl-st-heur'-def nres-rel-def in-pair-collect-simp f,
        rule-format] unfolding f
      apply (simp only: fref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp)
      apply (drule spec[of - x])
      apply (drule spec[of - y])
      apply simp
      apply (rule weaken-⇓'[of - ‹A ×r twl-st-heur''' (length (get-clauses-wl-heur x))›])
      apply (fastforce simp: twl-st-heur'-def)+
      done
    done
qed

definition (in −) lit-of-hd-trail-st-heur :: ‹isasat ⇒ nat literal nres› where
  ‹lit-of-hd-trail-st-heur S = do {
    ASSERT (fst (get-trail-wl-heur S) ≠ []);
    RETURN (lit-of-last-trail-pol (get-trail-wl-heur S))
  }›
```

10.6.1 Lifting of Options

definition *get-target-opts* :: ‹*isasat* ⇒ *opts-target*› **where**
‹*get-target-opts S* = *opts-target* (*get-opts S*)›

definition *get-subsumption-opts* :: ‹*isasat* ⇒ *bool*› **where**
‹*get-subsumption-opts S* = *opts-subsumption* (*get-opts S*)›

definition *get-GC-units-opt* :: ‹*isasat* ⇒ *64 word*› **where**
‹*get-GC-units-opt S* = *opts-GC-units-lim* (*get-opts S*)›

definition *units-since-last-GC-st* :: ‹*isasat* ⇒ *64 word*› **where**
‹*units-since-last-GC-st S* = *units-since-last-GC* (*get-stats-heur S*)›

definition *units-since-beginning-st* :: ‹*isasat* ⇒ *64 word*› **where**
‹*units-since-beginning-st S* = *units-since-beginning* (*get-stats-heur S*)›

definition *reset-units-since-last-GC-st* :: ‹*isasat* ⇒ *isasat*› **where**
‹*reset-units-since-last-GC-st S* =
(*let stats* = *get-stats-heur S in*
let stats = *reset-units-since-last-GC stats in*
let S = *set-stats-wl-heur stats S in S*
)›

definition *clss-size-resetUS0-st* :: ‹*isasat* ⇒ *isasat*› **where**
‹*clss-size-resetUS0-st S* =
(*let lcount* = *get-learned-count S in*
let lcount = *clss-size-resetUS0 lcount in*
let S = *set-learned-count-wl-heur lcount S in S*
)›

definition *is-fully-propagated-heur-st* :: ‹*isasat* ⇒ *bool*› **where**
‹*is-fully-propagated-heur-st S* = *is-fully-propagated-heur* (*get-heur S*)›

definition *print-trail-st* :: ‹*isasat* ⇒ -› **where**
‹*print-trail-st* = (λ*S*. *print-trail* (*get-trail-wl-heur S*))›

definition *print-trail-st2* **where**
‹*print-trail-st2* - = ()›

lemma *print-trail-st-print-trail-st2*:
‹*print-trail-st S* ≤ ⇓*unit-rel* (*RETURN* (*print-trail-st2 S*))›
unfolding *print-trail-st2-def print-trail-st-def*
print-trail-def
apply (*refine-vcg WHILET-rule*[**where**
R = ‹*measure* (λ*i*. *Suc* (*length* (*fst* (*get-trail-wl-heur S*))) − *i*)› **and**
I = ‹λ*i*. *i* ≤ *length* (*fst* (*get-trail-wl-heur S*))›])
subgoal by *auto*
subgoal by *auto*
subgoal unfolding *print-literal-of-trail-def* **by** *auto*
subgoal unfolding *print-literal-of-trail-def* **by** *auto*
done

lemma *print-trail-st-print-trail-st2-rel*:
‹(*print-trail-st*, *RETURN o print-trail-st2*) ∈ *Id* →$_f$ (⟨*unit-rel*⟩*nres-rel*)›

using *print-trail-st-print-trail-st2* **by** (*force intro*!: *frefI nres-relI*)

named-theorems *isasat-state-simp*

lemma [*isasat-state-simp*]:
‹*learned-clss-count* (*Tuple17.set-q occs S*) = *learned-clss-count S*›
‹*learned-clss-count* (*Tuple17.set-p old-arena S*) = *learned-clss-count S*›
‹*learned-clss-count* (*Tuple17.set-o opts S*) = *learned-clss-count S*›
‹*learned-clss-count* (*Tuple17.set-n lcount S*) = *learned-clss-count-lcount lcount*›
‹*learned-clss-count* (*Tuple17.set-m aivdom S*) = *learned-clss-count S*›
‹*learned-clss-count* (*Tuple17.set-l heur S*) = *learned-clss-count S*›
‹*learned-clss-count* (*Tuple17.set-k stats S*) = *learned-clss-count S*›
‹*learned-clss-count* (*Tuple17.set-j outl S*) = *learned-clss-count S*›
‹*learned-clss-count* (*Tuple17.set-i lbd S*) = *learned-clss-count S*›
‹*learned-clss-count* (*Tuple17.set-h ccach S*) = *learned-clss-count S*›
‹*learned-clss-count* (*Tuple17.set-g count' S*) = *learned-clss-count S*›
‹*learned-clss-count* (*Tuple17.set-f vmtf' S*) = *learned-clss-count S*›
‹*learned-clss-count* (*Tuple17.set-e W S*) = *learned-clss-count S*›
‹*learned-clss-count* (*Tuple17.set-d j S*) = *learned-clss-count S*›
‹*learned-clss-count* (*Tuple17.set-c D S*) = *learned-clss-count S*›
‹*learned-clss-count* (*Tuple17.set-b N S*) = *learned-clss-count S*›
‹*learned-clss-count* (*Tuple17.set-a M S*) = *learned-clss-count S*›
‹*get-trail-wl-heur* (*set-learned-count-wl-heur lcount S*) = *get-trail-wl-heur S*›
‹*get-clauses-wl-heur* (*set-learned-count-wl-heur lcount S*) = *get-clauses-wl-heur S*›
‹*get-conflict-wl-heur* (*set-learned-count-wl-heur lcount S*) = *get-conflict-wl-heur S*›
‹*literals-to-update-wl-heur* (*set-learned-count-wl-heur lcount S*) = *literals-to-update-wl-heur S*›
‹*get-watched-wl-heur* (*set-learned-count-wl-heur lcount S*) = *get-watched-wl-heur S*›
‹*get-vmtf-heur* (*set-learned-count-wl-heur lcount S*) = *get-vmtf-heur S*›
‹*get-count-max-lvls-heur* (*set-learned-count-wl-heur lcount S*) = *get-count-max-lvls-heur S*›
‹*get-conflict-cach* (*set-learned-count-wl-heur lcount S*) = *get-conflict-cach S*›
‹*get-lbd* (*set-learned-count-wl-heur lcount S*) = *get-lbd S*›
‹*get-outlearned-heur* (*set-learned-count-wl-heur lcount S*) = *get-outlearned-heur S*›
‹*get-stats-heur* (*set-learned-count-wl-heur lcount S*) = *get-stats-heur S*›
‹*get-aivdom* (*set-learned-count-wl-heur lcount S*) = *get-aivdom S*›
‹*get-heur* (*set-learned-count-wl-heur lcount S*) = *get-heur S*›
‹*get-learned-count* (*set-learned-count-wl-heur lcount S*) = *lcount*›
‹*get-opts* (*set-learned-count-wl-heur lcount S*) = *get-opts S*›
‹*get-old-arena* (*set-learned-count-wl-heur lcount S*) = *get-old-arena S*›
‹*get-occs* (*set-learned-count-wl-heur lcount S*) = *get-occs S*›
by (*solves* ‹*cases S*; *auto simp*: *learned-clss-count-def*›)+

lemmas [*isasat-state-simp*] = *tuple17-state-simp*
lemmas [*simp*] = *isasat-state-simp*

definition (**in** −) *length-ll-fs-heur* :: ‹*isasat* ⇒ *nat literal* ⇒ *nat*› **where**
‹*length-ll-fs-heur S L* = *length* (*watched-by-int S L*)›

definition (**in** −) *length-watchlist* :: ‹*nat watcher list list* ⇒ *nat literal* ⇒ *nat*› **where**
‹*length-watchlist S L* = *length-ll S* (*nat-of-lit L*)›

definition *mop-length-watched-by-int* :: ‹*isasat* ⇒ *nat literal* ⇒ *nat nres*› **where**
‹*mop-length-watched-by-int S L* = *do* {
ASSERT(*nat-of-lit L* < *length* (*get-watched-wl-heur S*));
RETURN (*length* (*watched-by-int S L*))

}›

definition *end-of-rephasing-phase-st* :: ‹*isasat* ⇒ *64 word*› **where**
‹*end-of-rephasing-phase-st* = (λ*S*. *end-of-rephasing-phase-heur* (*get-heur S*))›

definition *end-of-restart-phase-st* :: ‹*isasat* ⇒ *64 word*› **where**
‹*end-of-restart-phase-st* = (λ*S*. *end-of-restart-phase* (*get-heur S*))›

definition *get-vmtf-heur-array* **where**
‹*get-vmtf-heur-array S* = (*fst* (*get-focused-heuristics* (*get-vmtf-heur S*)))›

definition *get-vmtf-heur-fst* **where**
‹*get-vmtf-heur-fst S* = (*fst o snd o snd*) (*get-focused-heuristics* (*get-vmtf-heur S*))›

definition *isa-vmtf-heur-fst* **where**
‹*isa-vmtf-heur-fst x* = (*case x of Bump-Heuristics hstable focused foc* - ⇒
RETURN (*vmtf-heur-fst focused*))›

definition *get-bump-heur-array-nth* **where**
‹*get-bump-heur-array-nth S i* = *get-vmtf-heur-array S* ! *i*›

definition *mop-mark-added-heur-st* :: ‹-› **where**
‹*mop-mark-added-heur-st L S* = *do* {
let heur = *get-heur S*;
heur ← *mop-mark-added-heur L True heur*;
RETURN (*set-heur-wl-heur heur S*)
} ›

definition *mark-added-clause-heur2* **where**
‹*mark-added-clause-heur2 S C* = *do* {
i ← *mop-arena-length-st S C*;
ASSERT (*i* ≤ *length* (*get-clauses-wl-heur S*));
(-, *S*) ← *WHILE*$_T$ (λ(*j*, *S*). *j* < *i*)
(λ(*j*, *S*). *do* {
ASSERT (*j*<*i*);
L ← *mop-access-lit-in-clauses-heur S C j*;
S ← *mop-mark-added-heur-st* (*atm-of L*) *S*;
RETURN (*j*+*1*, *S*)
})
(*0*, *S*);
RETURN S
}›

definition *mark-added-clause2* **where**
‹*mark-added-clause2 S C* = *do* {
i ← *RETURN* (*length* (*get-clauses-wl S* ∝ *C*));
(-, *S*) ← *WHILE*$_T$ $^{\lambda(j,\ T).\ j \le i \wedge T = S}$ (λ(*j*, *S*). *j* < *i*)
(λ(*j*, *S*). *do* {
ASSERT (*j*<*i*);
L ← *mop-clauses-at* (*get-clauses-wl S*) *C j*;
ASSERT (*L* ∈ *set* (*get-clauses-wl S* ∝ *C*));
let S = *S*;
RETURN (*j*+*1*, *S*)
})
(*0*, *S*);

RETURN S
}›

lemma *mop-mark-added-heur-st-it*:
assumes ‹(*S*,*T*) ∈ *twl-st-heur*› **and** ‹*A* ∈# *all-atms-st T*›
shows ‹*mop-mark-added-heur-st A S* ≤ *SPEC* (λ*c*. (*c*, *T*) ∈ {(*U*, *V*). (*U*, *V*) ∈ *twl-st-heur* ∧
(*get-clauses-wl-heur U*) = *get-clauses-wl-heur S* ∧
learned-clss-count U = *learned-clss-count S*})›
proof −
have *heur*: ‹*heuristic-rel* (*all-atms-st T*) (*get-heur S*)›
using *assms*(*1*)
by (*auto simp*: *twl-st-heur-def*)

show *?thesis*
unfolding *mop-mark-added-heur-st-def mop-mark-added-heur-def*
apply *refine-vcg*
subgoal
using *heur assms*(*2*)
unfolding *mark-added-heur-pre-def mark-added-heur-pre-stats-def*
by (*auto simp*: *heuristic-rel-def heuristic-rel-stats-def*
phase-saving-def $\mathcal{L}_{all}$*-all-atms-all-lits atms-of-def* $\mathcal{L}_{all}$*-add-mset*
dest!: *multi-member-split*)
subgoal by (*use assms* **in** ‹*auto simp add*: *twl-st-heur-def*›)
done
qed

lemma *mark-added-clause-heur2-id*:
assumes ‹(*S*,*T*) ∈ *twl-st-heur*› **and** ‹*C* ∈# *dom-m* (*get-clauses-wl T*)›
shows ‹*mark-added-clause-heur2 S C*
≤ ⇓{(*U*, *V*). (*U*, *V*) ∈ *twl-st-heur* ∧ (*get-clauses-wl-heur U*) = *get-clauses-wl-heur S* ∧
learned-clss-count U = *learned-clss-count S*} (*RETURN T*)› (**is** ‹- ≤⇓*?R* -›)
proof −
have *1*: ‹*mark-added-clause2 T C* ≤ ⇓*Id* (*RETURN T*)›
unfolding *mark-added-clause2-def mop-clauses-at-def nres-monad3*
apply (*refine-vcg WHILEIT-rule*[**where** *R* = ‹*measure* (λ(*i*,-). *length* (*get-clauses-wl T* ∝ *C*) −*i*)›])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*use assms* **in** *auto*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
have [*refine*]: ‹*y*′ ∈# *dom-m x*′ ⟹
((*x*, *y*), *x*′, *y*′) ∈ {(*N*, *N*′). *valid-arena N N*′ (*set* (*get-vdom S*))} ×*f nat-rel* ⟹
mop-arena-length x y ≤ *SPEC* (λ*y*. (*y*, *length* (*x*′ ∝ *y*′)) ∈ {(*a*,*b*). (*a*,*b*)∈*nat-rel* ∧ *a* = *length* (*x*′ ∝ *y*′)})› **for** *x y x*′ *y*′
apply (*rule mop-arena-length*[*THEN fref-to-Down-curry*, *of* - - - - ‹*set* (*get-vdom S*)›, *unfolded comp-def conc-fun-RETURN prod.simps*, *THEN order-trans*])
apply *assumption*
apply *assumption*

```
    by auto
  have [refine]: ‹((0, S), 0, T) ∈ nat-rel ×r ?R›
    using assms by auto
  have 2: ‹mark-added-clause-heur2 S C ≤ ⇓?R (mark-added-clause2 T C)›
    unfolding mark-added-clause-heur2-def mop-arena-length-st-def mop-access-lit-in-clauses-heur-def
      mark-added-clause2-def
    apply (refine-vcg mop-mark-added-heur-st-it)
    subgoal by (use assms in auto)
    subgoal by (use assms in ‹auto simp: twl-st-heur-def›)
    subgoal using assms by (auto simp: twl-st-heur-def dest: arena-lifting(10))
    subgoal by auto
    subgoal by auto
    apply (rule-tac vdom = ‹set (get-vdom (x2a))› in mop-arena-lit2)
    subgoal by (use assms in ‹auto simp: twl-st-heur-def›)
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by (use assms in ‹auto simp: all-atms-st-def all-atms-def all-lits-def ran-m-def
        all-lits-of-mm-add-mset image-Un atm-of-all-lits-of-m(2)
      dest!: multi-member-split›)
    subgoal by auto
    subgoal by auto
    done
  show ?thesis
    unfolding mop-arena-length-st-def mop-access-lit-in-clauses-heur-def
    apply (rule order-trans[OF 2])
    apply (rule ref-two-step')
    apply (rule 1[unfolded Down-id-eq])
    done
qed

definition mop-is-marked-added-heur-st where
  ‹mop-is-marked-added-heur-st S = mop-is-marked-added-heur (get-heur S)›

lemma is-marked-added-heur-st-it:
  assumes ‹(S,T) ∈ twl-st-heur› and ‹A ∈# all-atms-st T›
  shows ‹mop-is-marked-added-heur-st S A ≤ SPEC(λc. (c, d) ∈ (UNIV :: (bool × bool) set))›
proof −
  have heur: ‹heuristic-rel (all-atms-st T) (get-heur S)›
    using assms(1)
    by (auto simp: twl-st-heur-def)
  then have ‹is-marked-added-heur-pre (get-heur S) A›
    using assms
    unfolding is-marked-added-heur-pre-def
    by (auto simp: heuristic-rel-def is-marked-added-heur-pre-stats-def
      heuristic-rel-stats-def phase-saving-def atms-of-ℒall-𝒜in)
  then show ?thesis
    unfolding mop-is-marked-added-heur-st-def mop-is-marked-added-heur-def
    by auto
qed

definition schedule-next-pure-lits-st :: ‹isasat ⇒ isasat› where
  ‹schedule-next-pure-lits-st S = set-heur-wl-heur (schedule-next-pure-lits (get-heur S)) S›

definition next-pure-lits-schedule-st :: ‹isasat ⇒ -› where
  ‹next-pure-lits-schedule-st S = next-pure-lits-schedule (get-heur S)›
```

definition *schedule-info-of-st* :: ‹*isasat* ⇒ -› **where**
‹*schedule-info-of-st S* = *schedule-info-of* (*get-heur S*)›

definition *schedule-next-reduce-st* :: ‹*64 word* ⇒ *isasat* ⇒ *isasat*› **where**
‹*schedule-next-reduce-st b S* = *set-heur-wl-heur* (*schedule-next-reduce b* (*get-heur S*)) *S*›

definition *next-reduce-schedule-st* :: ‹*isasat* ⇒ -› **where**
‹*next-reduce-schedule-st S* = *next-reduce-schedule* (*get-heur S*)›

definition *schedule-next-subsume-st* :: ‹*64 word* ⇒ *isasat* ⇒ *isasat*› **where**
‹*schedule-next-subsume-st b S* = *set-heur-wl-heur* (*schedule-next-subsume b* (*get-heur S*)) *S*›

definition *next-subsume-schedule-st* :: ‹*isasat* ⇒ -› **where**
‹*next-subsume-schedule-st S* = *next-subsume-schedule* (*get-heur S*)›

lemma *avdom-delete-index-vdom-heur*[*simp*]:
‹*get-avdom* (*delete-index-vdom-heur i S*) = (*get-avdom S*)›
‹*get-tvdom* (*delete-index-vdom-heur i S*) = *delete-index-and-swap* (*get-tvdom S*) *i*›
by (*cases S*; *auto simp*: *delete-index-vdom-heur-def*; *fail*)+
lemma [*simp*]:
‹*learned-clss-count* (*delete-index-vdom-heur C T*) = *learned-clss-count T*›
‹*learned-clss-count* (*mark-unused-st-heur C T*) = *learned-clss-count T*›
by (*cases T*; *auto simp*: *learned-clss-count-def delete-index-vdom-heur-def*
mark-unused-st-heur-def; *fail*)+

lemma *get-vdom-mark-garbage*[*simp*]:
‹*get-vdom* (*mark-garbage-heur C i S*) = *get-vdom S*›
‹*get-avdom* (*mark-garbage-heur C i S*) = *delete-index-and-swap* (*get-avdom S*) *i*›
‹*get-ivdom* (*mark-garbage-heur C i S*) = *get-ivdom S*›
‹*get-tvdom* (*mark-garbage-heur C i S*) = *get-tvdom S*›
‹*get-tvdom* (*mark-garbage-heur3 C i S*) = *delete-index-and-swap* (*get-tvdom S*) *i*›
‹*get-ivdom* (*mark-garbage-heur3 C i S*) = *get-ivdom S*›
‹*get-vdom* (*mark-garbage-heur3 C i S*) = *get-vdom S*›
‹*learned-clss-count* (*mark-garbage-heur3 C i* (*S*)) ≤ *learned-clss-count S*›
‹*learned-clss-count* (*mark-garbage-heur3 C i* (*incr-wasted-st b S*)) ≤ *learned-clss-count S*›
by (*cases S*; *auto simp*: *mark-garbage-heur-def mark-garbage-heur3-def*
learned-clss-count-def incr-wasted-st-def; *fail*)+

fun *get-reductions-count* :: ‹*isasat* ⇒ *64 word*› **where**
‹*get-reductions-count S* = *get-reduction-count* (*get-stats-heur S*)›

abbreviation *get-irredundant-count* **where**
‹*get-irredundant-count* ≡ *irredundant-clss*›

definition *get-irredundant-count-st* :: ‹*isasat* ⇒ *64 word*› **where**
‹*get-irredundant-count-st S* = *get-irredundant-count* (*get-stats-heur S*)›

lemma [*simp*]:
‹*get-avdom-aivdom* (*push-to-tvdom C aivdom*) = *get-avdom-aivdom aivdom*›
‹*get-vdom-aivdom* (*push-to-tvdom C aivdom*) = *get-vdom-aivdom aivdom*›
‹*get-ivdom-aivdom* (*push-to-tvdom C aivdom*) = *get-ivdom-aivdom aivdom*›

‹*get-tvdom-aivdom* (*push-to-tvdom C aivdom*) = *get-tvdom-aivdom aivdom* @ [*C*]›
by (*cases aivdom*; *auto simp*: *push-to-tvdom-def push-to-tvdom-int-def*; *fail*)+

lemma *aivdom-inv-dec-empty-tvdom*[*intro*!]:
‹*aivdom-inv-dec aivdom d* ⟹ *aivdom-inv-dec* (*empty-tvdom aivdom*) *d*›
by (*cases aivdom*) (*auto simp*: *aivdom-inv-dec-alt-def empty-tvdom-def*)

lemma [*simp*]:
‹*get-avdom-aivdom* (*empty-tvdom aivdom*) = *get-avdom-aivdom aivdom*›
‹*get-vdom-aivdom* (*empty-tvdom aivdom*) = *get-vdom-aivdom aivdom*›
‹*get-ivdom-aivdom* (*empty-tvdom aivdom*) = *get-ivdom-aivdom aivdom*›
‹*get-tvdom-aivdom* (*empty-tvdom aivdom*) = []›
by (*cases aivdom*; *auto simp*: *empty-tvdom-def*; *fail*)+

definition *isasat-fast-relaxed* :: ‹*isasat* ⇒ *bool*› **where**
‹*isasat-fast-relaxed S* ⟷ *length* (*get-clauses-wl-heur S*) ≤ *snat64-max* ∧ *learned-clss-count S* ≤ *unat64-max*›

definition *isasat-fast-relaxed2* :: ‹*isasat* ⇒ *nat* ⇒ *bool*› **where**
‹*isasat-fast-relaxed2 S n* ⟷ *isasat-fast-relaxed S* ∧ *n* < *unat64-max*›

definition *mop-arena-promote-st* **where**
‹*mop-arena-promote-st S C* = *do* {
let N' = *get-clauses-wl-heur S*;
let lcount = *get-learned-count S*;
ASSERT(*clss-size-lcount lcount* ≥ *1*);
let lcount = *clss-size-decr-lcount lcount*;
N' ← *mop-arena-set-status N' C IRRED*;
RETURN (*set-clauses-wl-heur N'* (*set-learned-count-wl-heur lcount S*))
}›

definition *set-stats-size-limit-st* **where**
‹*set-stats-size-limit-st lbd sze T* = (
let stats = *get-stats-heur T*;
stats = *set-stats-size-limit lbd sze stats*
in set-stats-wl-heur stats T
)›

definition *get-lsize-limit-stats-st* :: ‹-› **where**
‹*get-lsize-limit-stats-st T* = *get-lsize-limit-stats* (*get-stats-heur T*)›

definition *maybe-mark-added-clause-heur2* **where**
‹*maybe-mark-added-clause-heur2 S C* = *do* {
let (*lbd-limit*, *size-limit*) = *get-lsize-limit-stats-st S*;
lbd ← *mop-arena-lbd-st S C*;
sze ← *mop-arena-length-st S C*;
st ← *mop-arena-status-st S C*;
if (*st* = *IRRED* ∨ (*st* = *LEARNED* ∧ *lbd* ≤ *lbd-limit* ∧ *sze* ≤ *size-limit*))
then mark-added-clause-heur2 S C
else RETURN S
}›

lemma *maybe-mark-added-clause-heur2-id*:
assumes ‹(*S*,*T*) ∈ *twl-st-heur*› **and** ‹*C* ∈# *dom-m* (*get-clauses-wl T*)›
shows ‹*maybe-mark-added-clause-heur2 S C*

```
    ≤ ⇓{(U, V). (U, V) ∈ twl-st-heur ∧ (get-clauses-wl-heur U) = get-clauses-wl-heur S ∧
      learned-clss-count U = learned-clss-count S} (RETURN T)› (is ‹- ≤⇓?R -›)
proof −
  have
      valid: ‹valid-arena (get-clauses-wl-heur S) (get-clauses-wl T) (set (get-vdom S))›
    using assms unfolding all-atms-def all-lits-def
    by (auto simp: twl-st-heur'-def twl-st-heur-def)
  show ?thesis
    using assms unfolding maybe-mark-added-clause-heur2-def mop-arena-lbd-st-def
      mop-arena-lbd-def mop-arena-length-st-def mop-arena-length-def nres-monad3
      mop-arena-status-st-def mop-arena-status-def
    apply (refine-vcg mark-added-clause-heur2-id[OF assms(1), THEN order-trans])
    subgoal using valid by (auto simp: get-clause-LBD-pre-def
      arena-is-valid-clause-idx-def)
    subgoal using valid by (auto simp: arena-is-valid-clause-idx-def)
    subgoal using valid by (auto simp: arena-is-valid-clause-vdom-def)
    subgoal by (auto simp: Refine-Basic.RETURN-def conc-fun-RES)
    subgoal by auto
    done
qed

definition stats-forward-rounds-st :: ‹isasat ⇒ 64 word› where
  ‹stats-forward-rounds-st S = stats-forward-rounds (get-stats-heur S)›

definition incr-purelit-rounds-st :: ‹-› where
  ‹incr-purelit-rounds-st S = set-stats-wl-heur (incr-purelit-rounds (get-stats-heur S)) S›

lemma all-count-learned[simp]: ‹clss-size-allcount (get-learned-count S) = learned-clss-count S›
    by (auto simp: twl-st-heur'-def clss-size-allcount-def learned-clss-count-def clss-size-lcountU0-def
      clss-size-lcount-def clss-size-lcountUE-def clss-size-lcountUS-def clss-size-lcountUEk-def
      split: prod.splits)
end
theory IsaSAT-Sorting
  imports IsaSAT-Setup
begin
```

Chapter 11

Sorting of clauses

We use the sort function developped by Peter Lammich.

For the ordering, we prefer low lbds. If equal we take lower size. Then for tie, clauses derived later are preferred because they are not redundant. The last condition ensures that we do not depend on the order of the clauses in the array.

definition *clause-score-ordering* **where**
‹*clause-score-ordering* = (λ(*lbd*, *size*, *idx*) (*lbd'*, *size'*, *idx'*). *lbd* < *lbd'* ∨ (*lbd* = *lbd'* ∧ (*size* < *size'* ∨ (*size* = *size'* ∧ *idx* > *idx'*))))›

definition (**in** −) *clause-score-extract* :: ‹*arena* ⇒ *nat* ⇒ *nat* × *nat* × *nat*› **where**
‹*clause-score-extract arena C* = (
  if arena-status arena C = *DELETED*
  then (*unat32-max*, *snat64-max*, *snat64-max*) — deleted elements are the largest possible
  else
    let lbd = *arena-lbd arena C*;
      len = *arena-length arena C in*
    (*lbd*, *len*, *C*)
)›

definition *valid-sort-clause-score-pre-at* **where**
‹*valid-sort-clause-score-pre-at arena C* ⟷
  (∃ *i vdom*. *C* = *vdom* ! *i* ∧ *arena-is-valid-clause-vdom arena* (*vdom*!*i*) ∧
      (*arena-status arena* (*vdom*!*i*) ≠ *DELETED* ⟶
        (*get-clause-LBD-pre arena* (*vdom*!*i*) ∧ *arena-act-pre arena* (*vdom*!*i*)))
      ∧ *i* < *length vdom*)›

definition (**in** −)*valid-sort-clause-score-pre* **where**
‹*valid-sort-clause-score-pre arena vdom* ⟷
  (∀ *C* ∈ *set vdom*. *arena-is-valid-clause-vdom arena C* ∧
     (*arena-status arena C* ≠ *DELETED* ⟶
       (*get-clause-LBD-pre arena C* ∧ *arena-act-pre arena C*)))›

definition *clause-score-less* :: ‹*arena* ⇒ *nat* ⇒ *nat* ⇒ *bool*› **where**
clause-score-less arena i j ⟷
  clause-score-ordering (*clause-score-extract arena i*) (*clause-score-extract arena j*)

definition *idx-cdom* :: ‹*arena* ⇒ *nat set*› **where**
‹*idx-cdom arena* ≡ {*i*. *valid-sort-clause-score-pre-at arena i*}›

definition *mop-clause-score-less* **where**
‹*mop-clause-score-less arena i j = do* {
ASSERT(*valid-sort-clause-score-pre-at arena i*);
ASSERT(*valid-sort-clause-score-pre-at arena j*);
RETURN (*clause-score-ordering* (*clause-score-extract arena i*) (*clause-score-extract arena j*))
}›

end
theory *IsaSAT-Sorting-LLVM*
imports *IsaSAT-Sorting*
Examples.Sorting-Ex-Array-Idxs
IsaSAT-Literals-LLVM
begin

declare *α-butlast*[*simp del*]
hide-const (**open**) *NEMonad.RETURN NEMonad.ASSERT*

All the weird proofs comes from the fact that, while very useful, *vcg* enjoys instantiating schematic variables by true, rendering proofs impossible.

locale *pure-eo-adapter* =
fixes *elem-assn* :: ‹*'a* ⇒ *'ai*::*llvm-rep* ⇒ *assn*›
and *wo-assn* :: ‹*'a list* ⇒ *'oi*::*llvm-rep* ⇒ *assn*›
and *wo-get-impl* :: ‹*'oi* ⇒ *'size*::*len2 word* ⇒ *'ai llM*›
and *wo-set-impl* :: ‹*'oi* ⇒ *'size*::*len2 word* ⇒ *'ai* ⇒ *'oi llM*›
assumes *pure*[*safe-constraint-rules*]: ‹*is-pure elem-assn*›
and *get-hnr*: ‹(*uncurry wo-get-impl*,*uncurry mop-list-get*) ∈ *wo-assn*^{k} $*_a$ *snat-assn*^{k} $\to_a$ *elem-assn*›
and *set-hnr*: ‹(*uncurry2 wo-set-impl*,*uncurry2 mop-list-set*) ∈ [λ-.*True*]$_c$ *wo-assn*^{d} $*_a$ *snat-assn*^{k} $*_a$ *elem-assn*^{k} → *wo-assn* [λ((*ai*,-),-) *r*. *r*=*ai*]$_c$›
begin

lemmas [*sepref-fr-rules*] = *get-hnr set-hnr*

definition ‹*only-some-rel* ≡ {(*a*, *Some a*) | *a*. *True*} ∪ {(*x*, *None*) | *x*. *True*}›

definition ‹*eo-assn* ≡ *hr-comp wo-assn* (⟨*only-some-rel*⟩*list-rel*)›

definition ‹*eo-extract1 p i* ≡ *doN* { *r* ← *mop-list-get p i*; *RETURN* (*r*,*p*) }›
sepref-definition *eo-extract-impl* **is** ‹*uncurry eo-extract1*›
:: ‹*wo-assn*^{d} $*_a$ (*snat-assn'* *TYPE*(*'size*))^k $\to_a$ *elem-assn* $\times_a$ *wo-assn*›
unfolding *eo-extract1-def*
by *sepref*

lemma *mop-eo-extract-aux*: ‹*mop-eo-extract p i = doN* { *r* ← *mop-list-get p i*; *ASSERT* (*r*≠*None* ∧ *i*<*length p*); *RETURN* (*the r*, *p*[*i*:=*None*]) }›
by (*auto simp*: *pw-eq-iff refine-pw-simps assert-true-bind-conv summarize-ASSERT-conv intro*!: *bind-cong arg-cong*[*of - - ASSERT*])

lemma *assign-none-only-some-list-rel*:
assumes *SR*[*param*]: ‹(*a*, *a'*) ∈ ⟨*only-some-rel*⟩*list-rel*› **and** *L*: ‹*i* < *length a'*›
shows ‹(*a*, *a'*[*i* := *None*]) ∈ ⟨*only-some-rel*⟩*list-rel*›
proof −
have ‹(*a*[*i* := *a*!*i*], *a'*[*i* := *None*]) ∈ ⟨*only-some-rel*⟩*list-rel*›
apply (*parametricity*)
by (*auto simp*: *only-some-rel-def*)
also from *L list-rel-imp-same-length*[*OF SR*] **have** ‹*a*[*i* := *a*!*i*] = *a*› **by** *auto*

finally show *?thesis* **.**
qed

lemma *eo-extract1-refine*: ‹(*eo-extract1*, *mop-eo-extract*) ∈ ⟨*only-some-rel*⟩*list-rel* → *nat-rel* → ⟨*Id* ×$_r$ ⟨*only-some-rel*⟩*list-rel*⟩*nres-rel*›
unfolding *eo-extract1-def mop-eo-extract-aux*
supply *R* = *mop-list-get.fref*[*THEN frefD*, *OF TrueI prod-relI*, *unfolded uncurry-apply*, *THEN nres-relD*]
apply (*refine-rcg R*)
apply *assumption*
apply (*clarsimp simp*: *assign-none-only-some-list-rel*)
by (*auto simp*: *only-some-rel-def*)

lemma *eo-list-set-refine*: ‹(*mop-list-set*, *mop-eo-set*) ∈ ⟨*only-some-rel*⟩*list-rel* → *Id* → *Id* → ⟨⟨*only-some-rel*⟩*list-rel*⟩*nres*
unfolding *mop-list-set-alt mop-eo-set-alt*
apply *refine-rcg*
apply (*simp add*: *list-rel-imp-same-length*)
apply *simp*
apply *parametricity*
apply (*auto simp*: *only-some-rel-def*)
done

lemma *set-hnr'*: ‹(*uncurry2 wo-set-impl*,*uncurry2 mop-list-set*) ∈ *wo-assn*^{d} *∗*$_a$ *snat-assn*^{k} *∗*$_a$ *elem-assn*^{k} →$_a$ *wo-assn*›
apply (*rule hfref-cons*[*OF set-hnr*])
apply (*auto simp*: *entails-lift-extract-simps sep-algebra-simps*)
done

context
notes [*fcomp-norm-unfold*] = *eo-assn-def*[*symmetric*]
begin
lemmas *eo-extract-refine-aux* = *eo-extract-impl.refine*[*FCOMP eo-extract1-refine*]

lemma *eo-extract-refine*: (*uncurry eo-extract-impl*, *uncurry mop-eo-extract*)
∈ [λ-. *True*]$_c$ *eo-assn*^{d} *∗*$_a$ *snat-assn*^{k} → (*elem-assn* ×$_a$ *eo-assn*) [λ(*ai*,-) (-,*r*). *r*=*ai*]$_c$
apply (*sepref-to-hnr*)
apply (*rule hn-refine-nofailI*)
apply (*rule hnr-ceq-assnI*)
supply *R* = *eo-extract-refine-aux*[*to-hnr*, *unfolded APP-def*]
apply (*rule hn-refine-cons*[*OF - R*])
subgoal by (*auto simp*: *sep-algebra-simps entails-lift-extract-simps hn-ctxt-def pure-def invalid-assn-def*)
subgoal by (*auto simp*: *sep-algebra-simps entails-lift-extract-simps hn-ctxt-def pure-def invalid-assn-def*)
subgoal by (*auto simp*: *sep-algebra-simps entails-lift-extract-simps hn-ctxt-def pure-def invalid-assn-def*)
unfolding *eo-extract-impl-def mop-eo-extract-def hn-ctxt-def eo-assn-def hr-comp-def*
apply (*subst* (*3*) *sep.add-commute*)
supply *R* = *get-hnr*[*to-hnr*, *THEN hn-refineD*, *unfolded APP-def hn-ctxt-def*]
thm *R*
apply *vcg*
supply [*vcg-rules*] = *R*

apply (*vcg*)
supply [*simp*] = *refine-pw-simps list-rel-imp-same-length*

apply *vcg*[]
apply (*auto simp*: *POSTCOND-def EXTRACT-def*)
apply (*rule STATE-monoI*)
apply *assumption*
apply (*auto simp*: *entails-def*)
by (*simp add*: *pure-true-conv*)

lemmas *eo-set-refine-aux* = *set-hnr'*[*FCOMP eo-list-set-refine*]

lemma *pure-entails-empty*: ‹*is-pure A* ⟹ *A a c* ⊢ □›
by (*auto simp*: *is-pure-def sep-algebra-simps entails-lift-extract-simps*)

lemma *eo-set-refine*: ‹(*uncurry2 wo-set-impl*, *uncurry2 mop-eo-set*) ∈ [λ-. *True*]$_c$ *eo-assn*^{d} ∗$_a$ *snat-assn*^{k} ∗$_a$ *elem-assn*^{d} → (*eo-assn*) [λ((*ai*,-),-) *r*. *r*=*ai*]$_c$›
apply (*sepref-to-hnr*)
apply (*rule hn-refine-nofailI*)
apply (*rule hnr-ceq-assnI*)
supply *R* = *eo-set-refine-aux*[*to-hnr*, *unfolded APP-def*]
apply (*rule hn-refine-cons*[*OF - R*])
subgoal by (*auto simp*: *sep-algebra-simps entails-lift-extract-simps hn-ctxt-def pure-def invalid-assn-def pure-entails-empty*[*OF pure*])
subgoal by (*auto simp*: *sep-algebra-simps entails-lift-extract-simps hn-ctxt-def pure-def invalid-assn-def pure-entails-empty*[*OF pure*])
subgoal by (*auto simp*: *sep-algebra-simps entails-lift-extract-simps hn-ctxt-def pure-def invalid-assn-def pure-entails-empty*[*OF pure*])
unfolding *hn-ctxt-def eo-assn-def hr-comp-def*
supply *R* = *set-hnr*[*to-hnr*, *THEN hn-refineD*, *unfolded APP-def hn-ctxt-def*]
supply [*vcg-rules*] = *R*
apply (*vcg*)
supply [*simp*] = *refine-pw-simps list-rel-imp-same-length*
apply (*vcg*)[]
apply (*auto simp*: *POSTCOND-def EXTRACT-def*)
apply (*rule STATE-monoI*)
apply *assumption*
apply (*auto simp*: *entails-def*)
by (*simp add*: *pure-true-conv*)

end

lemma *id-Some-only-some-rel*: ‹(*id*, *Some*) ∈ *Id* → *only-some-rel*›
by (*auto simp*: *only-some-rel-def*)

lemma *map-some-only-some-rel-iff*: ‹(*xs*, *map Some ys*) ∈ ⟨*only-some-rel*⟩*list-rel* ⟷ *xs*=*ys*›
apply (*rule iffI*)
subgoal
apply (*induction xs* ‹*map Some ys*› *arbitrary*: *ys rule*: *list-rel-induct*)
apply (*auto simp*: *only-some-rel-def*)
done
subgoal
apply (*rewrite* **in** ‹(¤,-)› *list.map-id*[*symmetric*])
apply (*parametricity add*: *id-Some-only-some-rel*)
by *simp*
done

lemma *wo-assn-conv*: ‹*wo-assn xs ys* = *eo-assn* (*map Some xs*) *ys*›
unfolding *eo-assn-def hr-comp-def*
by (*auto simp*: *pred-lift-extract-simps sep-algebra-simps fun-eq-iff map-some-only-some-rel-iff*)

lemma *to-eo-conv-refine*: ‹(*Mreturn*, *mop-to-eo-conv*) ∈ [λ-. *True*]$_c$ *wo-assn*^{d} → (*eo-assn*) [λ(*ai*) (*r*). *r*=*ai*]$_c$›
unfolding *mop-to-eo-conv-def*
apply *sepref-to-hoare*
apply (*rewrite wo-assn-conv*)
apply *vcg*
done

lemma ‹*None* ∉ *set xs* ⟷ (∃ *ys*. *xs* = *map Some ys*)›
using *None-not-in-set-conv* **by** *auto*

lemma *to-wo-conv-refine*: ‹(*Mreturn*, *mop-to-wo-conv*) ∈ [λ-. *True*]$_c$ *eo-assn*^{d} → (*wo-assn*) [λ(*ai*) (*r*). *r*=*ai*]$_c$›
unfolding *mop-to-wo-conv-def eo-assn-def hr-comp-def*
apply *sepref-to-hoare*
apply (*auto simp add*: *refine-pw-simps map-some-only-some-rel-iff elim*!: *None-not-in-set-conv*)
by *vcg*

lemma *random-access-iterator*: *random-access-iterator wo-assn eo-assn elem-assn*
Mreturn Mreturn
eo-extract-impl
wo-set-impl
apply *unfold-locales*
using *to-eo-conv-refine to-wo-conv-refine eo-extract-refine eo-set-refine*
apply *blast*+
done

sublocale *random-access-iterator wo-assn eo-assn elem-assn*
Mreturn Mreturn
eo-extract-impl
wo-set-impl
by (*rule random-access-iterator*)

end

lemma *is-pureE-abs*:
assumes *is-pure P*
obtains *P'* **where** *P* = (λ*x x'*. ↑(*P' x x'*))
using *assms* **unfolding** *is-pure-def* **by** *blast*

lemma *al-pure-eo*: ‹*is-pure A* ⟹ *pure-eo-adapter A* (*al-assn A*) *arl-nth arl-upd*›
apply *unfold-locales*
apply *assumption*
apply (*rule al-nth-hnr-mop*; *simp*)
subgoal
apply (*sepref-to-hnr*)
apply (*rule hn-refine-nofailI*)
apply (*rule hnr-ceq-assnI*)
supply *R* = *al-upd-hnr-mop*[*to-hnr*, *unfolded APP-def*, *of A*]
apply (*rule hn-refine-cons*[*OF* - *R*])
subgoal by (*auto simp*: *hn-ctxt-def pure-def invalid-assn-def sep-algebra-simps entails-lift-extract-simps*)

subgoal by (*auto simp*: *hn-ctxt-def pure-def invalid-assn-def sep-algebra-simps entails-lift-extract-simps*)
subgoal by (*auto simp*: *hn-ctxt-def pure-def invalid-assn-def sep-algebra-simps entails-lift-extract-simps*)
subgoal by (*auto simp*: *hn-ctxt-def pure-def invalid-assn-def sep-algebra-simps entails-lift-extract-simps*)
unfolding *hn-ctxt-def al-assn-def hr-comp-def pure-def in-snat-rel-conv-assn*

apply (*erule is-pureE*)
apply (*auto simp add*: *refine-pw-simps*)[]
apply *vcg*
apply (*rule wpa-monoI*)
apply (*rule arl-upd-rule*[*unfolded htriple-def*, *rule-format*])
apply (*rule STATE-monoI*)
apply *assumption*
apply (*auto simp flip*: *in-snat-rel-conv-assn simp*: *fri-basic-extract-simps*
dr-assn-pure-asm-prefix-def entails-def pred-lift-extract-simps
SOLVE-AUTO-DEFER-def list-rel-imp-same-length)[]
apply (*simp add*: *POSTCOND-def EXTRACT-def*)
apply (*subst*(*asm*) *sep-algebra-class.sep-conj-commute*)
apply (*subst STATE-monoI*)
apply *assumption*
apply (*rule conj-entails-mono*[*OF frame-rem1*])
apply *simp-all*
done
done

end
theory *IsaSAT-Arena-Sorting-LLVM*
imports *IsaSAT-Sorting-LLVM IsaSAT-Arena-LLVM*
begin

type-synonym *vdom-fast-assn* = ‹*64 word array-list64*›
abbreviation *vdom-fast-assn* :: ‹*vdom* ⇒ *vdom-fast-assn* ⇒ *assn*› **where**
‹*vdom-fast-assn* ≡ *arl64-assn sint64-nat-assn*›

sepref-def *delete-index-and-swap-code2*
is ‹*uncurry* (*RETURN oo delete-index-and-swap*)›
:: ‹[λ(*xs*, *i*). *i* < *length xs*]$_a$
vdom-fast-assn^d *$*_a$ *sint64-nat-assn*^{k} → *vdom-fast-assn*›
unfolding *delete-index-and-swap.simps*
by *sepref*

definition *idx-cdom* :: ‹*arena* ⇒ *nat set*› **where**
‹*idx-cdom arena* ≡ {*i*. *valid-sort-clause-score-pre-at arena i*}›

sepref-register *clause-score-extract arena-status arena-lbd unat32-max DELETED*

lemma *valid-sort-clause-score-pre-at-alt-def*:
‹*valid-sort-clause-score-pre-at arena C* ⟷
(∃ *i vdom*. *C* = *vdom* ! *i* ∧ *arena-is-valid-clause-vdom arena* (*vdom*!*i*) ∧
(*arena-status arena* (*vdom*!*i*) ≠ *DELETED* ⟶
(*get-clause-LBD-pre arena* (*vdom*!*i*) ∧ *arena-act-pre arena* (*vdom*!*i*)) ∧
arena-is-valid-clause-idx arena C))
∧ *i* < *length vdom*)›
unfolding *valid-sort-clause-score-pre-at-def arena-is-valid-clause-idx-def*
arena-act-pre-def arena-is-valid-clause-idx-def **by** *auto*

sepref-def (**in** −) *clause-score-extract-code*

is ‹*uncurry* (*RETURN oo clause-score-extract*)›
:: ‹[*uncurry valid-sort-clause-score-pre-at*]$_a$
arena-fast-assn^k $*_a$ *sint64-nat-assn*^{k} $\to$ *uint32-nat-assn* $\times_a$ *sint64-nat-assn* $\times_a$ *sint64-nat-assn*›
supply [[*goals-limit* = *1*]]
unfolding *clause-score-extract-def valid-sort-clause-score-pre-at-alt-def unat64-max-def*[*simplified*]
by *sepref*

sepref-def (**in** −) *clause-score-ordering-code*
is ‹*uncurry* (*RETURN oo clause-score-ordering*)›
:: ‹(*uint32-nat-assn* $\times_a$ *sint64-nat-assn* $\times_a$ *sint64-nat-assn*)^k $*_a$ (*uint32-nat-assn* $\times_a$ *sint64-nat-assn* $\times_a$ *sint64-nat-assn*)^k $\to_a$ *bool1-assn*›
supply [[*goals-limit* = *1*]]
unfolding *clause-score-ordering-def*
by *sepref*

sepref-register *mop-clause-score-less clause-score-less clause-score-ordering*
sepref-def *mop-clause-score-less-impl*
is ‹*uncurry2 mop-clause-score-less*›
:: ‹*arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\to_a$ *bool1-assn*›
unfolding *mop-clause-score-less-def*
by *sepref*

interpretation *LBD*: *weak-ordering-on-lt* **where**
C = ‹*idx-cdom vs*› **and**
less = ‹*clause-score-less vs*›
by *unfold-locales*
(*auto simp*: *clause-score-less-def clause-score-extract-def*
clause-score-ordering-def split: *if-splits*)

interpretation *LBD*: *parameterized-weak-ordering idx-cdom clause-score-less*
mop-clause-score-less
by *unfold-locales*
(*auto simp*: *mop-clause-score-less-def*
idx-cdom-def clause-score-less-def)

global-interpretation *LBD*: *parameterized-sort-impl-context*
‹*woarray-assn snat-assn*› ‹*eoarray-assn snat-assn*› *snat-assn*
Mreturn Mreturn
eo-extract-impl
array-upd
idx-cdom clause-score-less mop-clause-score-less mop-clause-score-less-impl
‹*arena-fast-assn*›
defines
LBD-is-guarded-insert-impl = *LBD.is-guarded-param-insert-impl*
and *LBD-is-unguarded-insert-impl* = *LBD.is-unguarded-param-insert-impl*
and *LBD-unguarded-insertion-sort-impl* = *LBD.unguarded-insertion-sort-param-impl*
and *LBD-guarded-insertion-sort-impl* = *LBD.guarded-insertion-sort-param-impl*
and *LBD-final-insertion-sort-impl* = *LBD.final-insertion-sort-param-impl*

and *LBD-pcmpo-idxs-impl* = *LBD.pcmpo-idxs-impl*
and *LBD-pcmpo-v-idx-impl* = *LBD.pcmpo-v-idx-impl*
and *LBD-pcmpo-idx-v-impl* = *LBD.pcmpo-idx-v-impl*
and *LBD-pcmp-idxs-impl* = *LBD.pcmp-idxs-impl*

and *LBD-mop-geth-impl* = *LBD.mop-geth-impl*
and *LBD-mop-seth-impl* = *LBD.mop-seth-impl*
and *LBD-sift-down-impl* = *LBD.sift-down-impl*
and *LBD-heapify-btu-impl* = *LBD.heapify-btu-impl*
and *LBD-heapsort-impl* = *LBD.heapsort-param-impl*
and *LBD-qsp-next-l-impl* = *LBD.qsp-next-l-impl*
and *LBD-qsp-next-h-impl* = *LBD.qsp-next-h-impl*
and *LBD-qs-partition-impl* = *LBD.qs-partition-impl*

and *LBD-partition-pivot-impl* = *LBD.partition-pivot-impl*
and *LBD-introsort-aux-impl* = *LBD.introsort-aux-param-impl*
and *LBD-introsort-impl* = *LBD.introsort-param-impl*
and *LBD-move-median-to-first-impl* = *LBD.move-median-to-first-param-impl*

apply *unfold-locales*
apply (*rule eo-hnr-dep*)+
unfolding *GEN-ALGO-def refines-param-relp-def*
by (*rule mop-clause-score-less-impl.refine*)

global-interpretation
LBD-it: *pure-eo-adapter sint64-nat-assn vdom-fast-assn arl-nth arl-upd*
defines *LBD-it-eo-extract-impl* = *LBD-it.eo-extract-impl*
apply (*rule al-pure-eo*)
by *simp*

global-interpretation *LBD-it*: *parameterized-sort-impl-context*
vdom-fast-assn ‹*LBD-it.eo-assn*› *sint64-nat-assn*
Mreturn Mreturn
LBD-it-eo-extract-impl
arl-upd
idx-cdom clause-score-less mop-clause-score-less mop-clause-score-less-impl
‹*arena-fast-assn*›
defines
LBD-it-is-guarded-insert-impl = *LBD-it.is-guarded-param-insert-impl*
and *LBD-it-is-unguarded-insert-impl* = *LBD-it.is-unguarded-param-insert-impl*
and *LBD-it-unguarded-insertion-sort-impl* = *LBD-it.unguarded-insertion-sort-param-impl*
and *LBD-it-guarded-insertion-sort-impl* = *LBD-it.guarded-insertion-sort-param-impl*
and *LBD-it-final-insertion-sort-impl* = *LBD-it.final-insertion-sort-param-impl*

and *LBD-it-pcmpo-idxs-impl* = *LBD-it.pcmpo-idxs-impl*
and *LBD-it-pcmpo-v-idx-impl* = *LBD-it.pcmpo-v-idx-impl*
and *LBD-it-pcmpo-idx-v-impl* = *LBD-it.pcmpo-idx-v-impl*
and *LBD-it-pcmp-idxs-impl* = *LBD-it.pcmp-idxs-impl*

and *LBD-it-mop-geth-impl* = *LBD-it.mop-geth-impl*
and *LBD-it-mop-seth-impl* = *LBD-it.mop-seth-impl*
and *LBD-it-sift-down-impl* = *LBD-it.sift-down-impl*
and *LBD-it-heapify-btu-impl* = *LBD-it.heapify-btu-impl*
and *LBD-it-heapsort-impl* = *LBD-it.heapsort-param-impl*
and *LBD-it-qsp-next-l-impl* = *LBD-it.qsp-next-l-impl*
and *LBD-it-qsp-next-h-impl* = *LBD-it.qsp-next-h-impl*
and *LBD-it-qs-partition-impl* = *LBD-it.qs-partition-impl*

and *LBD-it-partition-pivot-impl* = *LBD-it.partition-pivot-impl*
and *LBD-it-introsort-aux-impl* = *LBD-it.introsort-aux-param-impl*
and *LBD-it-introsort-impl* = *LBD-it.introsort-param-impl*
and *LBD-it-move-median-to-first-impl* = *LBD-it.move-median-to-first-param-impl*

apply *unfold-locales*
unfolding *GEN-ALGO-def refines-param-relp-def*
apply (*rule mop-clause-score-less-impl.refine*)
done

definition *idx-clause-cdom* :: ‹*arena* ⇒ *nat set*› **where**
‹*idx-clause-cdom arena* ≡ {*i*. *arena-is-valid-clause-idx arena i*}›

sepref-def (**in** −) *arena-length-code*
is ‹*uncurry* (*RETURN oo arena-length*)›
:: ‹[*uncurry arena-is-valid-clause-idx*]$_a$
arena-fast-assn^k $*_a$ *sint64-nat-assn*^{k} → *sint64-nat-assn*›
supply [[*goals-limit* = *1*]]
unfolding *clause-score-extract-def valid-sort-clause-score-pre-at-alt-def unat64-max-def*[*simplified*]
by *sepref*

sepref-def (**in** −) *clause-size-ordering-code*
is ‹*uncurry* (*RETURN oo* (≤))›
:: ‹(*sint64-nat-assn*)^k $*_a$ (*sint64-nat-assn*)^k $\to_a$ *bool1-assn*›
supply [[*goals-limit* = *1*]]
unfolding *clause-score-ordering-def*
by *sepref*

definition *clause-size-less* **where**
‹*clause-size-less arena i j* = (*arena-length arena i* < *arena-length arena j*)›

definition *mop-clause-size-less* **where**
‹*mop-clause-size-less arena i j* = *do* {
ASSERT(*arena-is-valid-clause-idx arena i*);
ASSERT(*arena-is-valid-clause-idx arena j*);
RETURN (*clause-size-less arena i j*)
}›

sepref-def *mop-clause-size-less-impl*
is ‹*uncurry2 mop-clause-size-less*›
:: ‹*arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\to_a$ *bool1-assn*›
unfolding *mop-clause-size-less-def clause-size-less-def*
by *sepref*

interpretation *Size-Ordering*: *weak-ordering-on-lt* **where**
C = ‹*idx-clause-cdom vs*› **and**
less = ‹*clause-size-less vs*›
by *unfold-locales*
(*auto simp*: *clause-size-less-def clause-score-extract-def*
clause-score-ordering-def split: *if-splits*)

interpretation *Size-Ordering*: *parameterized-weak-ordering idx-clause-cdom clause-size-less*
mop-clause-size-less

by *unfold-locales*
(*auto simp*: *mop-clause-size-less-def*
idx-clause-cdom-def clause-size-less-def)

global-interpretation *Size-Ordering*: *parameterized-sort-impl-context*
‹*woarray-assn snat-assn*› ‹*eoarray-assn snat-assn*› *snat-assn*
Mreturn Mreturn
eo-extract-impl
array-upd
idx-clause-cdom clause-size-less mop-clause-size-less mop-clause-size-less-impl
‹*arena-fast-assn*›
defines
Size-Ordering-is-guarded-insert-impl = *Size-Ordering.is-guarded-param-insert-impl*
and *Size-Ordering-is-unguarded-insert-impl* = *Size-Ordering.is-unguarded-param-insert-impl*
and *Size-Ordering-unguarded-insertion-sort-impl* = *Size-Ordering.unguarded-insertion-sort-param-impl*
and *Size-Ordering-guarded-insertion-sort-impl* = *Size-Ordering.guarded-insertion-sort-param-impl*
and *Size-Ordering-final-insertion-sort-impl* = *Size-Ordering.final-insertion-sort-param-impl*

and *Size-Ordering-pcmpo-idxs-impl* = *Size-Ordering.pcmpo-idxs-impl*
and *Size-Ordering-pcmpo-v-idx-impl* = *Size-Ordering.pcmpo-v-idx-impl*
and *Size-Ordering-pcmpo-idx-v-impl* = *Size-Ordering.pcmpo-idx-v-impl*
and *Size-Ordering-pcmp-idxs-impl* = *Size-Ordering.pcmp-idxs-impl*

and *Size-Ordering-mop-geth-impl* = *Size-Ordering.mop-geth-impl*
and *Size-Ordering-mop-seth-impl* = *Size-Ordering.mop-seth-impl*
and *Size-Ordering-sift-down-impl* = *Size-Ordering.sift-down-impl*
and *Size-Ordering-heapify-btu-impl* = *Size-Ordering.heapify-btu-impl*
and *Size-Ordering-heapsort-impl* = *Size-Ordering.heapsort-param-impl*
and *Size-Ordering-qsp-next-l-impl* = *Size-Ordering.qsp-next-l-impl*
and *Size-Ordering-qsp-next-h-impl* = *Size-Ordering.qsp-next-h-impl*
and *Size-Ordering-qs-partition-impl* = *Size-Ordering.qs-partition-impl*

and *Size-Ordering-partition-pivot-impl* = *Size-Ordering.partition-pivot-impl*
and *Size-Ordering-introsort-aux-impl* = *Size-Ordering.introsort-aux-param-impl*
and *Size-Ordering-introsort-impl* = *Size-Ordering.introsort-param-impl*
and *Size-Ordering-move-median-to-first-impl* = *Size-Ordering.move-median-to-first-param-impl*

apply *unfold-locales*
unfolding *GEN-ALGO-def refines-param-relp-def*
by (*rule mop-clause-size-less-impl.refine*)

global-interpretation *Size-Ordering-it*: *parameterized-sort-impl-context*
vdom-fast-assn ‹*LBD-it.eo-assn*› *sint64-nat-assn*
Mreturn Mreturn
LBD-it-eo-extract-impl
arl-upd
idx-clause-cdom clause-size-less mop-clause-size-less mop-clause-size-less-impl
‹*arena-fast-assn*›
defines
Size-Ordering-it-is-guarded-insert-impl = *Size-Ordering-it.is-guarded-param-insert-impl*
and *Size-Ordering-it-is-unguarded-insert-impl* = *Size-Ordering-it.is-unguarded-param-insert-impl*
and *Size-Ordering-it-unguarded-insertion-sort-impl* = *Size-Ordering-it.unguarded-insertion-sort-param-impl*
and *Size-Ordering-it-guarded-insertion-sort-impl* = *Size-Ordering-it.guarded-insertion-sort-param-impl*
and *Size-Ordering-it-final-insertion-sort-impl* = *Size-Ordering-it.final-insertion-sort-param-impl*

```
    and Size-Ordering-it-pcmpo-idxs-impl = Size-Ordering-it.pcmpo-idxs-impl
    and Size-Ordering-it-pcmpo-v-idx-impl = Size-Ordering-it.pcmpo-v-idx-impl
    and Size-Ordering-it-pcmpo-idx-v-impl = Size-Ordering-it.pcmpo-idx-v-impl
    and Size-Ordering-it-pcmp-idxs-impl = Size-Ordering-it.pcmp-idxs-impl

    and Size-Ordering-it-mop-geth-impl = Size-Ordering-it.mop-geth-impl
    and Size-Ordering-it-mop-seth-impl = Size-Ordering-it.mop-seth-impl
    and Size-Ordering-it-sift-down-impl = Size-Ordering-it.sift-down-impl
    and Size-Ordering-it-heapify-btu-impl = Size-Ordering-it.heapify-btu-impl
    and Size-Ordering-it-heapsort-impl = Size-Ordering-it.heapsort-param-impl
    and Size-Ordering-it-qsp-next-l-impl = Size-Ordering-it.qsp-next-l-impl
    and Size-Ordering-it-qsp-next-h-impl = Size-Ordering-it.qsp-next-h-impl
    and Size-Ordering-it-qs-partition-impl = Size-Ordering-it.qs-partition-impl

    and Size-Ordering-it-partition-pivot-impl = Size-Ordering-it.partition-pivot-impl
    and Size-Ordering-it-introsort-aux-impl = Size-Ordering-it.introsort-aux-param-impl
    and Size-Ordering-it-introsort-impl = Size-Ordering-it.introsort-param-impl
  and Size-Ordering-it-move-median-to-first-impl = Size-Ordering-it.move-median-to-first-param-impl

  apply unfold-locales
  unfolding GEN-ALGO-def refines-param-relp-def
  apply (rule mop-clause-size-less-impl.refine)
  done



lemmas [llvm-inline] = LBD-it.eo-extract-impl-def[THEN meta-fun-cong, THEN meta-fun-cong]

export-llvm
  ‹LBD-heapsort-impl :: - ⇒ - ⇒ -›
  ‹LBD-introsort-impl :: - ⇒ - ⇒ -›
  ‹Size-Ordering-heapsort-impl :: - ⇒ - ⇒ -›
  ‹Size-Ordering-introsort-impl :: - ⇒ - ⇒ -›


type-synonym virtual-vdom-fast-assn = ‹64 word›
definition virtual-vdom-fast-assn :: ‹vdom ⇒ virtual-vdom-fast-assn ⇒ -› where
  ‹virtual-vdom-fast-assn = hr-comp sint64-nat-assn {(a,b). a= 0}›

definition qqq where ‹qqq xs - = Mreturn xs›
lemmas [llvm-inline] = qqq-def

lemma [unfolded qqq-def, sepref-fr-rules]:
  ‹(uncurry (qqq), uncurry (RETURN oo op-list-append))
  ∈ virtual-vdom-fast-assn^k *_a sint64-nat-assn^k →_a virtual-vdom-fast-assn›
proof −
  have [iff]: ‹x = snat u ∧ snat-invar u ∧ x = 0 ⟷ x = 0 ∧ u= 0› for x u
    by (auto intro: snat-invar-0)
     (auto simp add: snat-eq-unat-aux2 unat-arith-simps(3))

  have [simp]: ⋀ns u. virtual-vdom-fast-assn (ns::nat list) (u) = ↑(u=0)
    by (auto simp add: Sepref-Basic.pure-def virtual-vdom-fast-assn-def hr-comp-def snat-rel-def
      snat.rel-def br-def
      simp flip: import-param-3
      simp del: import-param-3)
```

```
    (auto simp: import-param-3 exists-eq-star-conv)

  have [simp]: ‹(∃x. (↑True ∧* ↑(x = a @ [b])) s) = □ s› for a b s
    by (auto simp: Exists-eq-simp, simp-all add: pure-true-conv)
  show ?thesis
    unfolding qqq-def
    apply sepref-to-hoare
    by (vcg)
qed


definition empty-virtual-vdom :: ‹nat list› where
  ‹empty-virtual-vdom = []›

sepref-register empty-virtual-vdom
lemma [sepref-fr-rules]:
  ‹(uncurry0 (Mreturn 0), uncurry0 (RETURN empty-virtual-vdom))
  ∈ unit-assn^k →_a virtual-vdom-fast-assn›
proof −
  have [iff]: ‹x = snat u ∧ snat-invar u ∧ x = 0 ⟷ x = 0 ∧ u= 0› for x u
    by (auto intro: snat-invar-0)
     (auto simp add: snat-eq-unat-aux2 unat-arith-simps(3))

  have [simp]: ⋀ns u. virtual-vdom-fast-assn (ns::nat list) (u) = ↑(u=0)
    by (auto simp add: Sepref-Basic.pure-def virtual-vdom-fast-assn-def hr-comp-def snat-rel-def
      snat.rel-def br-def
      simp flip: import-param-3
      simp del: import-param-3)
     (auto simp: import-param-3 exists-eq-star-conv)

  have [simp]: ‹(∃x. (↑True ∧* ↑(x = a @ [b])) s) = □ s› for a b s
    by (auto simp: Exists-eq-simp, simp-all add: pure-true-conv)
  show ?thesis
    unfolding qqq-def
    apply sepref-to-hoare
    by (vcg)
qed

schematic-goal mk-free-ghost-assn[sepref-frame-free-rules]: ‹MK-FREE virtual-vdom-fast-assn ?fr›
  unfolding virtual-vdom-fast-assn-def
  by synthesize-free

experiment
begin
definition ‹test0 ≡ (λ xs C. RETURN (xs @ [C]))›
sepref-def test
  is ‹uncurry test0›
  :: ‹virtual-vdom-fast-assn^k  *_a sint64-nat-assn^k →_a virtual-vdom-fast-assn›
  supply [[goals-limit=1]]
  unfolding test0-def
  by sepref

   export-llvm test
 end
end
theory IsaSAT-Watch-List-LLVM
```

imports *IsaSAT-Watch-List IsaSAT-Literals-LLVM IsaSAT-Arena-Sorting-LLVM*
begin

type-synonym *watched-wl-uint32* = ‹(*64*,(*64 word* × *32 word* × *1 word*),*64*)*array-array-list*›

abbreviation ‹*watcher-fast-assn* ≡ *sint64-nat-assn* ×$_a$ *unat-lit-assn* ×$_a$ *bool1-assn* ›
abbreviation ‹*watchlist-fast-assn* ≡ *aal-assn′ TYPE*(*64*) *TYPE*(*64*) *watcher-fast-assn*›

lemma [*def-pat-rules*]: ‹*append-ll* ≡ *op-list-list-push-back*›
by (*rule eq-reflection*) (*auto simp*: *append-ll-def fun-eq-iff*)

sepref-register *mop-append-ll mop-arena-length*

sepref-def *mop-append-ll-impl*
is ‹*uncurry2 mop-append-ll*›
:: ‹[λ((*W*, *i*), -). *length* (*W* ! (*nat-of-lit i*)) < *snat64-max*]$_a$
watchlist-fast-assn^d *$*_a$ *unat-lit-assn*^{k} *$*_a$ *watcher-fast-assn*^{k} → *watchlist-fast-assn*›
unfolding *mop-append-ll-def*
by *sepref*

sepref-def *rewatch-heur-fast-code*
is ‹*uncurry2* (*rewatch-heur*)›
:: ‹[λ((*vdom*, *arena*), *W*). (∀ *x* ∈ *set vdom*. *x* ≤ *snat64-max*) ∧ *length arena* ≤ *snat64-max* ∧
length vdom ≤ *snat64-max*]$_a$
vdom-fast-assn^k *$*_a$ *arena-fast-assn*^{k} *$*_a$ *watchlist-fast-assn*^{d} → *watchlist-fast-assn*›
supply [[*goals-limit=1*]]
arena-lit-pre-le-snat64-max[*dest*] *arena-is-valid-clause-idx-le-unat64-max*[*dest*]
supply [*simp*] = *append-ll-def*
supply [*dest*] = *arena-lit-implI*(*1*)
unfolding *rewatch-heur-alt-def Let-def PR-CONST-def*
unfolding *while-eq-nfoldli*[*symmetric*]
apply (*subst while-upt-while-direct*, *simp*)
unfolding *if-not-swap*
FOREACH-cond-def FOREACH-body-def
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *aal-gen-swap*:
‹*W*[*L* := *More-List.swap* (*W* ! *L*) *i j*] =
(*let x* = *W* ! *L* ! *i*; *y* = *W* ! *L* ! *j*; *W* = *op-list-list-upd W L j x*; *W* = *op-list-list-upd W L i y in W*)›
apply (*auto simp*: *convert-swap More-List.swap-def*)
by (*smt* (*verit*, *best*) *list-update-id′ list-update-overwrite list-update-swap nth-list-update′*)

sepref-register *watchlist-put-binaries-first-one watchlist-put-binaries-first*
sepref-def *watchlist-put-binaries-first-one-impl*
is ‹*uncurry watchlist-put-binaries-first-one*›
:: ‹*watchlist-fast-assn*^{d} *$*_a$ *sint64-nat-assn*^{k} →$_a$ *watchlist-fast-assn*›
unfolding *watchlist-put-binaries-first-one-def*
unfolding *if-not-swap convert-swap fold-op-list-list-llen*
unfolding
convert-swap aal-gen-swap fold-op-list-list-idx op-list-get-def
op-list-set-def fold-op-list-list-upd
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *watchlist-put-binaries-first-impl*
is ‹*watchlist-put-binaries-first*›
:: ‹*watchlist-fast-assn*^{d} →$_a$ *watchlist-fast-assn*›
unfolding *watchlist-put-binaries-first-def*
unfolding *if-not-swap convert-swap fold-op-list-list-llen*
unfolding
convert-swap aal-gen-swap fold-op-list-list-idx op-list-get-def
op-list-set-def op-list-list-len-def[*symmetric*]
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

end
theory *IsaSAT-Mark-LLVM*
imports *IsaSAT-Mark*
IsaSAT-Literals-LLVM
begin

type-synonym *mark* = ‹*bool option*›

definition *mark-rel-aux* :: ‹(*int* × *mark*) *set*› **where**
‹*mark-rel-aux* = {(*0*, *None*), (*1*, *Some True*), (−*1*, *Some False*)}›

definition *mark-rel* :: ‹(*3 word* × *bool option*) *set*› **where**
‹*mark-rel* = *sint-rel*′ *TYPE*(*3*) *O mark-rel-aux*›

abbreviation *mark-assn* :: ‹*mark* ⇒ *3 word* ⇒ *assn*› **where**
‹*mark-assn* ≡ *pure mark-rel*›

definition *marked-struct-rel* :: ‹(- × *lookup-clause-rel*) *set*› **where**
‹*marked-struct-rel* = *Id* ×$_r$ ⟨*mark-rel-aux*⟩*list-rel*›

type-synonym *mark-assn* = ‹*32 word* × *3 word ptr*›
definition *marked-struct-assn* :: ‹*lookup-clause-rel* ⇒ *mark-assn* ⇒ *assn*› **where**
‹*marked-struct-assn* = *uint32-nat-assn* ×$_a$ *array-assn* (*pure mark-rel*)›

definition *Mark* :: ‹*bool* ⇒ *bool option*› **where** [*simp*]: ‹*Mark* = *Some*›
definition *NoMark* :: ‹*bool option*› **where** [*simp*]: ‹*NoMark* = *None*›

lemmas *mark-defs* = *Mark-def*[*symmetric*] *NoMark-def*[*symmetric*]

lemmas [*fcomp-norm-unfold*] = *mark-rel-aux-def*[*symmetric*] *mark-rel-def*[*symmetric*]

sepref-def *Mark-impl* [*llvm-inline*]
is [] ‹*RETURN o* (λ*b*. *if b then 1 else* −*1*)› :: ‹*bool1-assn*^{k} →$_a$ *sint-assn*′ *TYPE*(*3*)›
apply (*annot-sint-const* ‹*TYPE*(*3*)›)
by *sepref*

sepref-def *NoMark-impl* [*llvm-inline*]
is [] ‹*uncurry0* (*RETURN 0*)› :: ‹*unit-assn*^{k} →$_a$ *sint-assn*′ *TYPE*(*3*)›
apply (*annot-sint-const* ‹*TYPE*(*3*)›)
by *sepref*

sepref-register *Mark NoMark*

lemma *Mark-rel-aux*: ‹(λ*b*. *if b then 1* :: *int else* −*1*, *Mark*) ∈ *bool-rel* → *mark-rel-aux*›
by (*auto intro*: *frefI split*: *if-splits simp*: *mark-rel-aux-def*)

lemmas [*sepref-fr-rules*] =
Mark-impl.refine[*FCOMP Mark-rel-aux*]

lemma *NoMark-rel-aux*: ‹(*0*, *NoMark*) ∈ *mark-rel-aux*›
by (*auto intro*: *frefI split*: *if-splits simp*: *mark-rel-aux-def*)

lemmas [*sepref-fr-rules*] =
NoMark-impl.refine[*FCOMP NoMark-rel-aux*]

lemma *mark-rel-aux-eq*: ‹((=), (=)) ∈ *mark-rel-aux* → *mark-rel-aux* → *bool-rel*›
by (*auto intro*!: *frefI simp*: *mark-rel-aux-def*)

sepref-def *mark-eq-impl* **is**
‹*uncurry* (*RETURN oo* (=))› :: ‹(*sint-assn'* *TYPE*(*3*))^d *$*_a$* (*sint-assn'* *TYPE*(*3*))^d →$_a$ *bool1-assn*›
unfolding *mark-rel-aux-def*
by *sepref*

lemmas [*sepref-fr-rules*] = *mark-eq-impl.refine*[*FCOMP mark-rel-aux-eq*]

sepref-register ‹(=) :: *mark* ⇒ - ⇒ -› *lit-is-in-lookup delete-from-lookup-conflict unmark-clause*
add-to-lookup-conflict pre-simplify-clause-lookup

sepref-def *lit-is-in-lookup-impl*
is ‹*uncurry lit-is-in-lookup*›
:: ‹*unat-lit-assn*^{k} *$*_a$* *marked-struct-assn*^{k} →$_a$ *bool1-assn*›
unfolding *lit-is-in-lookup-def mark-defs marked-struct-assn-def*
array-fold-custom-replicate
apply (*annot-all-atm-idxs*)
by *sepref*

sepref-def *delete-from-lookup-conflict-impl*
is ‹*uncurry delete-from-lookup-conflict*›
:: ‹*unat-lit-assn*^{k} *$*_a$* *marked-struct-assn*^{d} →$_a$ *marked-struct-assn*›
unfolding *delete-from-lookup-conflict-def mark-defs marked-struct-assn-def*
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
apply (*annot-all-atm-idxs*)
by *sepref*
abbreviation *clause-ll-assn* **where**
‹*clause-ll-assn* ≡ *al-assn'* *TYPE*(*64*) *unat-lit-assn*›

sepref-def *unmark-clause-impl*
is ‹*uncurry unmark-clause*›
:: ‹*clause-ll-assn*^{k} *$*_a$* *marked-struct-assn*^{d} →$_a$ *marked-struct-assn*›
unfolding *unmark-clause-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *add-to-lookup-conflict*
is ‹*uncurry* (*RETURN oo add-to-lookup-conflict*)›
:: ‹[λ(*L*, *n*, *D*). *atm-of L* < *length D* ∧ *n* < *unat32-max*]$_a$
unat-lit-assn^k *$*_a$* *marked-struct-assn*^{d} → *marked-struct-assn*›
unfolding *mark-defs marked-struct-assn-def add-to-lookup-conflict-def NOTIN-def ISIN-def*
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
apply (*annot-all-atm-idxs*)
by *sepref*

sepref-def *pre-simplify-clause-lookup-impl*
  is ‹*uncurry2 pre-simplify-clause-lookup*›
   :: ‹*clause-ll-assn*^{k} $*_a$ *clause-ll-assn*^{d} $*_a$ *marked-struct-assn*^{d} $\rightarrow_a$ *bool1-assn* $\times_a$ *clause-ll-assn* $\times_a$ *marked-struct-assn*›
  unfolding *pre-simplify-clause-lookup-def*
  supply [[*goals-limit=1*]]
  apply (*annot-snat-const* ‹*TYPE*(*64*)›)
  by *sepref*

end
theory *IsaSAT-Show*
  imports
    Show.Show-Instances
    IsaSAT-Setup
begin

Chapter 12

Printing information about progress

We provide a function to print some information about the state. This is mostly meant to ease extracting statistics and printing information during the run. Remark that this function is basically an FFI (to follow Andreas Lochbihler words) and is not unsafe (since printing has not side effects), but we do not need any correctness theorems.

However, it seems that the PolyML as targeted by *export-code checking* does not support that print function. Therefore, we cannot provide the code printing equations by default.

For the LLVM version code equations are not supported and hence we replace the function by hand.

definition *println-string* :: ‹*String.literal* ⇒ *unit*› **where**
‹*println-string* - = ()›

definition *print-c* :: ‹*64 word* ⇒ *unit*› **where**
‹*print-c* - = ()›

definition *print-char* :: ‹*64 word* ⇒ *unit*› **where**
‹*print-char* - = ()›

definition *print-uint64* :: ‹*64 word* ⇒ *unit*› **where**
‹*print-uint64* - = ()›

12.0.1 Print Information for IsaSAT

Printing the information slows down the solver by a huge factor.

definition *isasat-banner-content* **where**
‹*isasat-banner-content* =
''c conflicts decisions restarts uset avg-lbd
'' @
''c propagations reductions GC Learnt
'' @
''c clauses ''›

definition *isasat-information-banner* :: ‹- ⇒ *unit nres*› **where**
‹*isasat-information-banner* - =
RETURN (*println-string* (*String.implode* (*show isasat-banner-content*)))›

definition *isasat-current-information-stats* :: ‹*64 word* ⇒ *isasat-stats* ⇒ *clss-size* ⇒ *isasat-stats*› **where**
‹*isasat-current-information-stats* =

```
  (λcurr-phase stats (lcount, lcountUE, lcountUEk, lcountUS, -).
    if (stats-conflicts stats) AND 8191 = 8191 — (8191::'a) = (8192::'a) − (1::'a), i.e., we print when
all first bits are 1.
    then do{
      let - = print-c (stats-propagations stats);
        - = if curr-phase = 1 then print-open-colour 33 else ();
        - = print-char 126;
        - = print-uint64 (stats-propagations stats);
        - = print-uint64 (stats-conflicts stats);
        - = print-uint64 (of-nat lcount);
        - = print-uint64 (irredundant-clss stats);
        - = print-uint64 (stats-restarts stats);
        - = print-uint64 (stats-reductions stats);
        - = print-uint64 (stats-fixed stats);
        - = print-uint64 (stats-gcs stats);
        - = print-uint64 (ema-extract-value (stats-avg-lbd stats));
        - = print-uint64 (of-nat lcountUE);
        - = print-uint64 (of-nat lcountUEk);
        - = print-uint64 (of-nat lcountUS);
        - = print-close-colour 0
      in
        stats}
     else stats
   )›

definition isasat-current-information :: ‹64 word ⇒ isasat-stats ⇒ clss-size ⇒ isasat-stats› where
‹isasat-current-information =
  (λcurr-phase stats count. (isasat-current-information-stats curr-phase stats count)
    )›



definition isasat-current-status :: ‹isasat ⇒ isasat nres› where
‹isasat-current-status =
   (λS.
     let curr-phase = current-restart-phase (get-heur S);
        stats = (isasat-current-information curr-phase (get-stats-heur S) (get-learned-count S))
     in RETURN (set-stats-wl-heur stats S))›

lemma isasat-current-status-id:
  ‹(isasat-current-status, RETURN o id) ∈
  {(S, T). (S, T) ∈ twl-st-heur ∧ length (get-clauses-wl-heur S) ≤ r ∧ learned-clss-count S ≤ u} →f
    ⟨{(S, T). (S, T) ∈ twl-st-heur ∧ length (get-clauses-wl-heur S) ≤ r ∧ learned-clss-count S ≤
u}⟩nres-rel›
  by (intro frefI nres-relI)
    (auto simp: twl-st-heur-def isasat-current-status-def learned-clss-count-def)

definition isasat-print-progress :: ‹64 word ⇒ 64 word ⇒ isasat-stats ⇒ clss-size ⇒ unit› where
‹isasat-print-progress c curr-phase =
  (λstats (lcount, lcountUE, lcountUEk, lcountUS, -).
     let - = print-c (stats-propagations stats);
         - = if curr-phase = 1 then print-open-colour 33 else ();
         - = print-char c;
         - = print-uint64 (stats-propagations stats);
         - = print-uint64 (stats-conflicts stats);
         - = print-uint64 (of-nat lcount);
```

```
      - = print-uint64 (irredundant-clss stats);
      - = print-uint64 (stats-restarts stats);
      - = print-uint64 (stats-reductions stats);
      - = print-uint64 (stats-fixed stats);
      - = print-uint64 (stats-gcs stats);
      - = print-uint64 (ema-extract-value (stats-avg-lbd stats));
      - = print-uint64 (of-nat lcountUE);
      - = print-uint64 (of-nat lcountUEk);
      - = print-uint64 (of-nat lcountUS);
      - = print-close-colour 0
    in
      ())›

definition isasat-current-progress :: ‹64 word ⇒ isasat ⇒ unit nres› where
‹isasat-current-progress =
  (λc S.
    let
      curr-phase = current-restart-phase (get-heur S);
      - = isasat-print-progress c curr-phase (get-stats-heur S) (get-learned-count S)
    in RETURN ())›


end
theory IsaSAT-Trail-LLVM
imports IsaSAT-Literals-LLVM IsaSAT-Trail
begin

hide-const (open) NEMonad.ASSERT NEMonad.RETURN

type-synonym trail-pol-fast-assn =
  ‹32 word array-list64 × tri-bool-assn larray64 × 32 word larray64 ×
    64 word larray64 × 32 word ×
    32 word array-list64 × 64 word›

sepref-def DECISION-REASON-impl is ‹uncurry0 (RETURN DECISION-REASON)›
  :: ‹unit-assn^k →_a sint64-nat-assn›
  unfolding DECISION-REASON-def apply (annot-snat-const ‹TYPE(64)›) by sepref


definition trail-pol-fast-assn :: ‹trail-pol ⇒ trail-pol-fast-assn ⇒ assn› where
  ‹trail-pol-fast-assn ≡
    arl64-assn unat-lit-assn ×_a larray64-assn (tri-bool-assn) ×_a
    larray64-assn uint32-nat-assn ×_a
    larray64-assn sint64-nat-assn ×_a uint32-nat-assn ×_a
    arl64-assn uint32-nat-assn ×_a sint64-nat-assn›
```

Code generation

Conversion between incomplete and complete mode

```
sepref-def count-decided-pol-impl is
‹RETURN o count-decided-pol› :: ‹trail-pol-fast-assn^k →_a uint32-nat-assn›
  unfolding trail-pol-fast-assn-def count-decided-pol-def
  by sepref


sepref-def get-level-atm-fast-code
```

is ‹*uncurry* (*RETURN oo get-level-atm-pol*)›
:: ‹[*get-level-atm-pol-pre*]$_a$
trail-pol-fast-assn^k **$_a$ atom-assn*^{k} → *uint32-nat-assn*›
unfolding *get-level-atm-pol-def nat-shiftr-div2*[*symmetric*]
get-level-atm-pol-pre-def trail-pol-fast-assn-def
supply [[*eta-contract* = *false*, *show-abbrevs*=*false*]]
apply (*rewrite at* ‹*nth* -› *eta-expand*)
apply (*rewrite at* ‹*nth* - -› *annot-index-of-atm*)
by *sepref*

sepref-def *get-level-fast-code*
is ‹*uncurry* (*RETURN oo get-level-pol*)›
:: ‹[*get-level-pol-pre*]$_a$
trail-pol-fast-assn^k **$_a$ unat-lit-assn*^{k} → *uint32-nat-assn*›
unfolding *get-level-get-level-atm nat-shiftr-div2*[*symmetric*]
get-level-pol-pre-def get-level-pol-def
supply [[*goals-limit* = *1*]] *image-image*[*simp*] *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*[*simp*]
get-level-atm-pol-pre-def[*simp*]
by *sepref*

sepref-def *polarity-pol-fast-code*
is ‹*uncurry* (*RETURN oo polarity-pol*)›
:: ‹[*uncurry polarity-pol-pre*]$_a$ *trail-pol-fast-assn*^{k} **$_a$ unat-lit-assn*^{k} → *tri-bool-assn*›
unfolding *polarity-pol-def option.case-eq-if polarity-pol-pre-def*
trail-pol-fast-assn-def
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-def *polarity-pol-fast*
is ‹*uncurry* (*mop-polarity-pol*)›
:: ‹*trail-pol-fast-assn*^{k} **$_a$ unat-lit-assn*^{k} →$_a$ *tri-bool-assn*›
unfolding *mop-polarity-pol-def trail-pol-fast-assn-def*
polarity-pol-def polarity-pol-pre-def
by *sepref*

sepref-register *isa-length-trail*
sepref-def *isa-length-trail-fast-code*
is ‹*RETURN o isa-length-trail*›
:: ‹[*λ-. True*]$_a$ *trail-pol-fast-assn*^{k} → *snat-assn' TYPE*(*64*)›
unfolding *isa-length-trail-def isa-length-trail-pre-def length-uint32-nat-def*
trail-pol-fast-assn-def
by *sepref*

sepref-def *mop-isa-length-trail-fast-code*
is ‹*mop-isa-length-trail*›
:: ‹*trail-pol-fast-assn*^{k} →$_a$ *snat-assn' TYPE*(*64*)›
unfolding *mop-isa-length-trail-def isa-length-trail-pre-def length-uint32-nat-def*
by *sepref*

sepref-def *cons-trail-Propagated-tr-fast-code*
is ‹*uncurry2* (*cons-trail-Propagated-tr*)›
:: ‹*unat-lit-assn*^{k} **$_a$ sint64-nat-assn*^{k} **$_a$ trail-pol-fast-assn*^{d} →$_a$ *trail-pol-fast-assn*›

unfolding *cons-trail-Propagated-tr-def cons-trail-Propagated-tr-def*
SET-TRUE-def[*symmetric*] *SET-FALSE-def*[*symmetric*] *cons-trail-Propagated-tr-pre-def*
unfolding *trail-pol-fast-assn-def prod.case*
apply (*subst* (*3*)*annot-index-of-atm*)
apply (*subst* (*4*)*annot-index-of-atm*)

supply [[*goals-limit* = *1*]]

by *sepref*

sepref-def *tl-trail-tr-fast-code*
is ‹*RETURN o tl-trailt-tr*›
:: ‹[*tl-trailt-tr-pre*]$_a$
trail-pol-fast-assn^d → *trail-pol-fast-assn*›
supply *if-splits*[*split*] *option.splits*[*split*]
unfolding *tl-trailt-tr-def UNSET-def*[*symmetric*] *tl-trailt-tr-pre-def*
unfolding *trail-pol-fast-assn-def*
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
supply [[*goals-limit* = *1*]]
unfolding *fold-tuple-optimizations*
by *sepref*

sepref-def *tl-trail-proped-tr-fast-code*
is ‹*RETURN o tl-trail-propedt-tr*›
:: ‹[*tl-trail-propedt-tr-pre*]$_a$
trail-pol-fast-assn^d → *trail-pol-fast-assn*›
supply *if-splits*[*split*] *option.splits*[*split*]
unfolding *tl-trail-propedt-tr-def UNSET-def*[*symmetric*]
tl-trail-propedt-tr-pre-def
unfolding *trail-pol-fast-assn-def*
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-def *lit-of-last-trail-fast-code*
is ‹*RETURN o lit-of-last-trail-pol*›
:: ‹[λ(*M*, -). *M* ≠ []]$_a$ *trail-pol-fast-assn*^{k} → *unat-lit-assn*›
unfolding *lit-of-last-trail-pol-def trail-pol-fast-assn-def*
by *sepref*

sepref-def *cons-trail-Decided-tr-fast-code*
is ‹*uncurry* (*RETURN oo cons-trail-Decided-tr*)›
:: ‹[*cons-trail-Decided-tr-pre*]$_a$
unat-lit-assn^k *$*$_a$ *trail-pol-fast-assn*^{d} → *trail-pol-fast-assn*›
unfolding *cons-trail-Decided-tr-def cons-trail-Decided-tr-def trail-pol-fast-assn-def*
SET-TRUE-def[*symmetric*] *SET-FALSE-def*[*symmetric*] *cons-trail-Decided-tr-pre-def*

apply (*annot-unat-const* ‹*TYPE*(*32*)›)
apply (*rewrite at* ‹-@[⌑]› **in** ‹(-,⌑)› *annot-snat-unat-downcast*[**where** *'l*=‹*32*›])
supply [[*goals-limit* = *1*]]
unfolding *fold-tuple-optimizations*

by *sepref*

sepref-def *defined-atm-fast-code*
is ‹*uncurry* (*RETURN oo defined-atm-pol*)›
:: ‹[*uncurry defined-atm-pol-pre*]$_a$ *trail-pol-fast-assn*^{k} $*_a$ *atom-assn*^{k} $\to$ *bool1-assn*›
unfolding *defined-atm-pol-def UNSET-def*[*symmetric*] *tri-bool-eq-def*[*symmetric*]
defined-atm-pol-pre-def trail-pol-fast-assn-def Pos-rel-def[*symmetric*]
unfolding *ins-idx-upcast64*
supply *Pos-impl.refine*[*sepref-fr-rules*]
supply *UNSET-def*[*simp del*]
by *sepref*

sepref-register *get-propagation-reason-raw-pol*
sepref-def *get-propagation-reason-fast-code*
is ‹*uncurry get-propagation-reason-raw-pol*›
:: ‹*trail-pol-fast-assn*^{k} $*_a$ *unat-lit-assn*^{k} $\to_a$ *sint64-nat-assn*›
unfolding *get-propagation-reason-raw-pol-def trail-pol-fast-assn-def*

by *sepref*

sepref-register *isa-trail-nth trail-height-before-conflict*

sepref-def *isa-trail-nth-fast-code*
is ‹*uncurry isa-trail-nth*›
:: ‹*trail-pol-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\to_a$ *unat-lit-assn*›
unfolding *isa-trail-nth-def trail-pol-fast-assn-def*
by *sepref*

sepref-def *tl-trail-tr-no-CS-fast-code*
is ‹*RETURN o tl-trailt-tr-no-CS*›
:: ‹[*tl-trailt-tr-no-CS-pre*]$_a$
trail-pol-fast-assn^d $\to$ *trail-pol-fast-assn*›
supply *if-splits*[*split*] *option.splits*[*split*]
unfolding *tl-trailt-tr-no-CS-def UNSET-def*[*symmetric*] *tl-trailt-tr-no-CS-pre-def*
unfolding *trail-pol-fast-assn-def*
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-def *trail-conv-back-imp-fast-code*
is ‹*uncurry trail-conv-back-imp*›
:: ‹*uint32-nat-assn*^{k} $*_a$ *trail-pol-fast-assn*^{d} $\to_a$ *trail-pol-fast-assn*›
supply [[*goals-limit=1*]]
unfolding *trail-conv-back-imp-def trail-pol-fast-assn-def*
apply (*rewrite at* ‹*take* ⌑› *annot-unat-snat-upcast*[**where** *'l=64*])
by *sepref*

sepref-def *get-pos-of-level-in-trail-imp-fast-code*
is ‹*uncurry get-pos-of-level-in-trail-imp*›
:: ‹*trail-pol-fast-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $\to_a$ *uint32-nat-assn*›

unfolding *get-pos-of-level-in-trail-imp-def trail-pol-fast-assn-def*
apply (*rewrite at ‹- ! ⌑›* *annot-unat-snat-upcast*[**where** *'l=64*])
by *sepref*

sepref-def *get-the-propagation-reason-fast-code*
is ‹*uncurry get-the-propagation-reason-pol*›
:: ‹*trail-pol-fast-assn*^{k} $*_a$ *unat-lit-assn*^{k} $\rightarrow_a$ *snat-option-assn' TYPE*(*64*)›
unfolding *get-the-propagation-reason-pol-def trail-pol-fast-assn-def*
tri-bool-eq-def[*symmetric*]
by *sepref*

sepref-def *trail-height-before-conflict-impl*
is ‹*trail-height-before-conflict*›
:: ‹*trail-pol-fast-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
unfolding *trail-height-before-conflict-def trail-pol-fast-assn-def*
apply (*annot-unat-const ‹TYPE*(*32*)›)
apply (*rewrite at ‹- ! ⌑›* *annot-unat-snat-upcast*[**where** *'l=64*])
supply [[*goals-limit = 1*]]
by *sepref*

sepref-def *trail-zeroed-until-impl*
is ‹*RETURN o trail-zeroed-until*›
:: ‹*trail-pol-fast-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*›
unfolding *trail-zeroed-until-def trail-pol-fast-assn-def*
by *sepref*

sepref-def *trail-set-zeroed-until-impl*
is ‹*uncurry* (*RETURN oo trail-set-zeroed-until*)›
:: ‹*sint64-nat-assn*^{k} $*_a$ *trail-pol-fast-assn*^{d} $\rightarrow_a$ *trail-pol-fast-assn*›
unfolding *trail-set-zeroed-until-def trail-pol-fast-assn-def*
by *sepref*

schematic-goal *mk-free-trail-pol-fast-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE trail-pol-fast-assn ?fr*›
unfolding *trail-pol-fast-assn-def*
by *synthesize-free*

experiment begin

export-llvm
tri-bool-UNSET-impl
tri-bool-SET-TRUE-impl
tri-bool-SET-FALSE-impl
DECISION-REASON-impl
count-decided-pol-impl
get-level-atm-fast-code
get-level-fast-code
polarity-pol-fast-code
isa-length-trail-fast-code
cons-trail-Propagated-tr-fast-code
tl-trail-tr-fast-code
tl-trail-proped-tr-fast-code
lit-of-last-trail-fast-code
cons-trail-Decided-tr-fast-code
defined-atm-fast-code

get-propagation-reason-fast-code
isa-trail-nth-fast-code
tl-trail-tr-no-CS-fast-code
trail-conv-back-imp-fast-code
get-pos-of-level-in-trail-imp-fast-code
get-the-propagation-reason-fast-code
trail-height-before-conflict-impl
trail-set-zeroed-until-impl
trail-zeroed-until-impl
end

end
theory *IsaSAT-Profile-LLVM*
imports *IsaSAT-Profile IsaSAT-Literals-LLVM*
begin

sepref-register *IsaSAT-Profile.start*
IsaSAT-Profile.stop
IsaSAT-Profile.PROPAGATE
IsaSAT-Profile.ANALYZE
IsaSAT-Profile.GC
IsaSAT-Profile.REDUCE
IsaSAT-Profile.INITIALISATION
IsaSAT-Profile.MINIMIZATION

sepref-def *start-profile*
is ‹*RETURN o IsaSAT-Profile.start*›
:: ‹*word-assn*^{k} →$_a$ *unit-assn*›
unfolding *IsaSAT-Profile.start-def*
by *sepref*

sepref-def *stop-profile*
is ‹*RETURN o IsaSAT-Profile.stop*›
:: ‹*word-assn*^{k} →$_a$ *unit-assn*›
unfolding *IsaSAT-Profile.stop-def*
by *sepref*

sepref-def *IsaSAT-Profile-PROPAGATE*
is ‹*uncurry0* (*RETURN IsaSAT-Profile.PROPAGATE*)›
:: ‹*unit-assn*^{k} →$_a$ *word-assn*›
unfolding *IsaSAT-Profile.PROPAGATE-def*
by *sepref*

sepref-def *IsaSAT-Profile-ANALYZE*
is ‹*uncurry0* (*RETURN IsaSAT-Profile.ANALYZE*)›
:: ‹*unit-assn*^{k} →$_a$ *word-assn*›
unfolding *IsaSAT-Profile.ANALYZE-def*
by *sepref*

sepref-def *IsaSAT-Profile-GC*
is ‹*uncurry0* (*RETURN IsaSAT-Profile.GC*)›
:: ‹*unit-assn*^{k} →$_a$ *word-assn*›
unfolding *IsaSAT-Profile.GC-def*
by *sepref*

sepref-def *IsaSAT-Profile-REDUCE*
  is ‹*uncurry0* (*RETURN IsaSAT-Profile.REDUCE*)›
  :: ‹*unit-assn*^{k} →$_a$ *word-assn*›
  unfolding *IsaSAT-Profile.REDUCE-def*
  by *sepref*

sepref-def *IsaSAT-Profile-MINIMIZATION*
  is ‹*uncurry0* (*RETURN IsaSAT-Profile.MINIMIZATION*)›
  :: ‹*unit-assn*^{k} →$_a$ *word-assn*›
  unfolding *IsaSAT-Profile.MINIMIZATION-def*
  by *sepref*

sepref-def *IsaSAT-Profile-INITIALISATION*
  is ‹*uncurry0* (*RETURN IsaSAT-Profile.INITIALISATION*)›
  :: ‹*unit-assn*^{k} →$_a$ *word-assn*›
  unfolding *IsaSAT-Profile.INITIALISATION-def*
  by *sepref*

sepref-def *IsaSAT-Profile-PURE-LITERAL*
  is ‹*uncurry0* (*RETURN IsaSAT-Profile.PURE-LITERAL*)›
  :: ‹*unit-assn*^{k} →$_a$ *word-assn*›
  unfolding *IsaSAT-Profile.PURE-LITERAL-def*
  by *sepref*

sepref-def *IsaSAT-Profile-BINARY-SIMP*
  is ‹*uncurry0* (*RETURN IsaSAT-Profile.BINARY-SIMP*)›
  :: ‹*unit-assn*^{k} →$_a$ *word-assn*›
  unfolding *IsaSAT-Profile.BINARY-SIMP-def*
  by *sepref*

sepref-def *IsaSAT-Profile-SUBSUMPTION*
  is ‹*uncurry0* (*RETURN IsaSAT-Profile.SUBSUMPTION*)›
  :: ‹*unit-assn*^{k} →$_a$ *word-assn*›
  unfolding *IsaSAT-Profile.SUBSUMPTION-def*
  by *sepref*

experiment begin

export-llvm
  IsaSAT-Profile-PROPAGATE **is** ‹*PROFILE-CST IsaSAT-Profile-PROPAGATE*()›
  IsaSAT-Profile-REDUCE **is** ‹*PROFILE-CST IsaSAT-Profile-REDUCE*()›
  IsaSAT-Profile-GC **is** ‹*PROFILE-CST IsaSAT-Profile-GC*()›
  IsaSAT-Profile-ANALYZE **is** ‹*PROFILE-CST IsaSAT-Profile-ANALYZE*()›
  IsaSAT-Profile-MINIMIZATION **is** ‹*PROFILE-CST IsaSAT-Profile-MINIMIZATION*()›
  IsaSAT-Profile-INITIALISATION **is** ‹*PROFILE-CST IsaSAT-Profile-INITIALISATION*()›
  IsaSAT-Profile-SUBSUMPTION **is** ‹*PROFILE-CST IsaSAT-Profile-SUBSUMPTION*()›
  IsaSAT-Profile-PURE-LITERAL **is** ‹*PROFILE-CST IsaSAT-Profile-PURE-LITERAL*()›
  IsaSAT-Profile-BINARY-SIMP **is** ‹*PROFILE-CST IsaSAT-Profile-BINARY-SIMP*()›
  defines ‹
    typedef int8-t PROFILE-CST;
  ›
end
end
theory *IsaSAT-Lookup-Conflict-LLVM*
imports

IsaSAT-Lookup-Conflict
IsaSAT-Trail-LLVM
IsaSAT-Clauses-LLVM
LBD-LLVM
IsaSAT-Profile-LLVM
begin
hide-const (**open**) *NEMonad.ASSERT NEMonad.RETURN*

sepref-register *set-lookup-conflict-aa*
type-synonym *lookup-clause-assn* = ‹*32 word* × (*1 word*) *ptr*›

type-synonym (**in** −) *option-lookup-clause-assn* = ‹*1 word* × *lookup-clause-assn*›

type-synonym (**in** −) *out-learned-assn* = ‹*32 word array-list64*›

abbreviation (**in** −) *out-learned-assn* :: ‹*out-learned* ⇒ *out-learned-assn* ⇒ *assn*› **where**
‹*out-learned-assn* ≡ *arl64-assn unat-lit-assn*›

definition *minimize-status-int-rel* :: ‹(*nat* × *minimize-status*) *set*› **where**
‹*minimize-status-int-rel* = {(*0*, *SEEN-UNKNOWN*), (*1*, *SEEN-FAILED*), (*2*, *SEEN-REMOVABLE*)}›

abbreviation *minimize-status-ref-rel* **where**
‹*minimize-status-ref-rel* ≡ *snat-rel′ TYPE*(*8*)›

abbreviation *minimize-status-ref-assn* **where**
‹*minimize-status-ref-assn* ≡ *pure minimize-status-ref-rel*›

definition *minimize-status-rel* :: ‹-› **where**
‹*minimize-status-rel* = *minimize-status-ref-rel O minimize-status-int-rel*›

abbreviation *minimize-status-assn* :: ‹-› **where**
‹*minimize-status-assn* ≡ *pure minimize-status-rel*›

lemma *minimize-status-assn-alt-def*:
‹*minimize-status-assn* = *pure* (*snat-rel O minimize-status-int-rel*)›
unfolding *minimize-status-rel-def* **..**

lemmas [*fcomp-norm-unfold*] = *minimize-status-assn-alt-def*[*symmetric*]

definition *minimize-status-rel-eq* :: ‹*minimize-status* ⇒ *minimize-status* ⇒ *bool*› **where**
[*simp*]: ‹*minimize-status-rel-eq* = (=)›

lemma *minimize-status-rel-eq*:
‹((=), *minimize-status-rel-eq*) ∈ *minimize-status-int-rel* → *minimize-status-int-rel* → *bool-rel*›
by (*auto simp*: *minimize-status-int-rel-def*)

sepref-def *minimize-status-rel-eq-impl*
is [] ‹*uncurry* (*RETURN oo* (=))›
:: ‹*minimize-status-ref-assn*^{k} *$*_a$ *minimize-status-ref-assn*^{k} →$_a$ *bool1-assn*›
supply [[*goals-limit=1*]]
by *sepref*

sepref-register *minimize-status-rel-eq*

lemmas [*sepref-fr-rules*] = *minimize-status-rel-eq-impl.refine*[*FCOMP minimize-status-rel-eq*]

lemma
SEEN-FAILED-rel: ‹(*1*, *SEEN-FAILED*) ∈ *minimize-status-int-rel*› **and**
SEEN-UNKNOWN-rel: ‹(*0*, *SEEN-UNKNOWN*) ∈ *minimize-status-int-rel*› **and**
SEEN-REMOVABLE-rel: ‹(*2*, *SEEN-REMOVABLE*) ∈ *minimize-status-int-rel*›
by (*auto simp*: *minimize-status-int-rel-def*)

sepref-def *SEEN-FAILED-impl*
is [] ‹*uncurry0* (*RETURN 1*)›
:: ‹*unit-assn*^{k} →$_a$ *minimize-status-ref-assn*›
supply [[*goals-limit=1*]]
apply (*annot-snat-const* ‹*TYPE*(*8*)›)
by *sepref*

sepref-def *SEEN-UNKNOWN-impl*
is [] ‹*uncurry0* (*RETURN 0*)›
:: ‹*unit-assn*^{k} →$_a$ *minimize-status-ref-assn*›
supply [[*goals-limit=1*]]
apply (*annot-snat-const* ‹*TYPE*(*8*)›)
by *sepref*

sepref-def *SEEN-REMOVABLE-impl*
is [] ‹*uncurry0* (*RETURN 2*)›
:: ‹*unit-assn*^{k} →$_a$ *minimize-status-ref-assn*›
supply [[*goals-limit=1*]]
apply (*annot-snat-const* ‹*TYPE*(*8*)›)
by *sepref*

lemmas [*sepref-fr-rules*] = *SEEN-FAILED-impl.refine*[*FCOMP SEEN-FAILED-rel*]
SEEN-UNKNOWN-impl.refine[*FCOMP SEEN-UNKNOWN-rel*]
SEEN-REMOVABLE-impl.refine[*FCOMP SEEN-REMOVABLE-rel*]

definition *option-bool-impl-rel* **where**
‹*option-bool-impl-rel* = *bool1-rel O option-bool-rel*›

abbreviation *option-bool-impl-assn* :: ‹-› **where**
‹*option-bool-impl-assn* ≡ *pure* (*option-bool-impl-rel*)›

lemma *option-bool-impl-assn-alt-def*:
‹*option-bool-impl-assn* = *hr-comp bool1-assn option-bool-rel*›
unfolding *option-bool-impl-rel-def* **by** (*simp add*: *hr-comp-pure*)

lemmas [*fcomp-norm-unfold*] = *option-bool-impl-assn-alt-def*[*symmetric*]
option-bool-impl-rel-def[*symmetric*]

lemma *Some-rel*: ‹(λ-. *True*, *ISIN*) ∈ *bool-rel* → *option-bool-rel*›
by (*auto simp*: *option-bool-rel-def*)

sepref-def *Some-impl*
is [] ‹*RETURN o* (λ-. *True*)›
:: ‹*bool1-assn*^{k} →$_a$ *bool1-assn*›
by *sepref*

lemmas [*sepref-fr-rules*] = *Some-impl.refine*[*FCOMP Some-rel*]

lemma *is-Notin-rel*: ‹$(\lambda x.\ \neg x,\ \textit{is-NOTIN}) \in \textit{option-bool-rel} \to \textit{bool-rel}$›
  by (*auto simp*: *option-bool-rel-def*)

sepref-def *is-Notin-impl*
  is [] ‹$\textit{RETURN}\ o\ (\lambda x.\ \neg x)$›
  :: ‹$\textit{bool1-assn}^k \to_a \textit{bool1-assn}$›
  by *sepref*

lemmas [*sepref-fr-rules*] = *is-Notin-impl.refine*[*FCOMP is-Notin-rel*]

lemma *NOTIN-rel*: ‹$(\textit{False},\ \textit{NOTIN}) \in \textit{option-bool-rel}$›
  by (*auto simp*: *option-bool-rel-def*)

sepref-def *NOTIN-impl*
  is [] ‹$\textit{uncurry0}\ (\textit{RETURN False})$›
  :: ‹$\textit{unit-assn}^k \to_a \textit{bool1-assn}$›
  by *sepref*

lemmas [*sepref-fr-rules*] = *NOTIN-impl.refine*[*FCOMP NOTIN-rel*]

definition (**in** −) *lookup-clause-rel-assn*
  :: ‹*lookup-clause-rel* ⇒ *lookup-clause-assn* ⇒ *assn*›
where
 ‹$\textit{lookup-clause-rel-assn} \equiv (\textit{uint32-nat-assn} \times_a \textit{array-assn option-bool-impl-assn})$›

definition (**in** −)*conflict-option-rel-assn*
  :: ‹*conflict-option-rel* ⇒ *option-lookup-clause-assn* ⇒ *assn*›
where
 ‹$\textit{conflict-option-rel-assn} \equiv (\textit{bool1-assn} \times_a \textit{lookup-clause-rel-assn})$›

lemmas [*fcomp-norm-unfold*] = *conflict-option-rel-assn-def*[*symmetric*]
  lookup-clause-rel-assn-def[*symmetric*]

definition (**in** −)*ana-refinement-fast-rel* **where**
  ‹$\textit{ana-refinement-fast-rel} \equiv \textit{snat-rel}'\ \textit{TYPE}(64) \times_r \textit{unat-rel}'\ \textit{TYPE}(32) \times_r \textit{bool1-rel}$›

abbreviation (**in** −)*ana-refinement-fast-assn* **where**
  ‹$\textit{ana-refinement-fast-assn} \equiv \textit{sint64-nat-assn} \times_a \textit{uint32-nat-assn} \times_a \textit{bool1-assn}$›

lemma *ana-refinement-fast-assn-def*:
  ‹$\textit{ana-refinement-fast-assn} = \textit{pure ana-refinement-fast-rel}$›
  by (*auto simp*: *ana-refinement-fast-rel-def*)

abbreviation (**in** −)*analyse-refinement-fast-assn* **where**
  ‹$\textit{analyse-refinement-fast-assn} \equiv$
    $\textit{arl64-assn ana-refinement-fast-assn}$›

lemma *lookup-clause-assn-is-None-alt-def*:
  ‹$\textit{RETURN}\ o\ \textit{lookup-clause-assn-is-None} = (\lambda(b,\ \text{-},\ \text{-}).\ \textit{RETURN}\ b)$›
  unfolding *lookup-clause-assn-is-None-def* **by** *auto*

sepref-def *lookup-clause-assn-is-None-impl*

is ‹*RETURN o lookup-clause-assn-is-None*›
:: ‹*conflict-option-rel-assn*^{k} $\rightarrow_a$ *bool1-assn*›
unfolding *lookup-clause-assn-is-None-alt-def conflict-option-rel-assn-def*
lookup-clause-rel-assn-def
by *sepref*

lemma *size-lookup-conflict-alt-def*:
‹*RETURN o size-lookup-conflict* = (λ(-, *b*, -). *RETURN b*)›
unfolding *size-lookup-conflict-def* **by** *auto*

sepref-def *size-lookup-conflict-impl*
is ‹*RETURN o size-lookup-conflict*›
:: ‹*conflict-option-rel-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
unfolding *size-lookup-conflict-alt-def conflict-option-rel-assn-def*
lookup-clause-rel-assn-def
by *sepref*

sepref-def *is-in-conflict-code*
is ‹*uncurry* (*RETURN oo is-in-lookup-conflict*)›
:: ‹[λ((*n*, *xs*), *L*). *atm-of L* < *length xs*]$_a$
lookup-clause-rel-assn^k $*_a$ *unat-lit-assn*^{k} $\rightarrow$ *bool1-assn*›
supply [[*goals-limit=1*]]
unfolding *is-in-lookup-conflict-def is-NOTIN-alt-def*[*symmetric*]
lookup-clause-rel-assn-def
by *sepref*

lemma *lookup-clause-assn-is-empty-alt-def*:
‹*lookup-clause-assn-is-empty* = (λ*S*. *size-lookup-conflict S* = *0*)›
by (*auto simp*: *size-lookup-conflict-def lookup-clause-assn-is-empty-def fun-eq-iff*)

sepref-def *lookup-clause-assn-is-empty-impl*
is ‹*RETURN o lookup-clause-assn-is-empty*›
:: ‹*conflict-option-rel-assn*^{k} $\rightarrow_a$ *bool1-assn*›
unfolding *lookup-clause-assn-is-empty-alt-def*
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

definition *the-lookup-conflict* :: ‹*conflict-option-rel* $\Rightarrow$ -› **where**
‹*the-lookup-conflict* = *snd*›

lemma *the-lookup-conflict-alt-def*:
‹*RETURN o the-lookup-conflict* = (λ(-, (*n*, *xs*)). *RETURN* (*n*, *xs*))›
by (*auto simp*: *the-lookup-conflict-def*)

sepref-def *the-lookup-conflict-impl*
is ‹*RETURN o the-lookup-conflict*›
:: ‹*conflict-option-rel-assn*^{d} $\rightarrow_a$ *lookup-clause-rel-assn*›
unfolding *the-lookup-conflict-alt-def conflict-option-rel-assn-def*
lookup-clause-rel-assn-def
by *sepref*

definition *Some-lookup-conflict* :: ‹- $\Rightarrow$ *conflict-option-rel*› **where**

‹*Some-lookup-conflict xs* = (*False*, *xs*)›

lemma *Some-lookup-conflict-alt-def*:
‹*RETURN o Some-lookup-conflict* = (λ*xs*. *RETURN* (*False*, *xs*))›
by (*auto simp*: *Some-lookup-conflict-def*)

sepref-def *Some-lookup-conflict-impl*
is ‹*RETURN o Some-lookup-conflict*›
:: ‹*lookup-clause-rel-assn*^{d} →$_a$ *conflict-option-rel-assn*›
unfolding *Some-lookup-conflict-alt-def conflict-option-rel-assn-def*
lookup-clause-rel-assn-def
by *sepref*
sepref-register *Some-lookup-conflict*

type-synonym *cach-refinement-l-assn* = ‹*8 word ptr* × *32 word array-list64*›

definition (**in** −) *cach-refinement-l-assn* :: ‹- ⇒ *cach-refinement-l-assn* ⇒ -› **where**
‹*cach-refinement-l-assn* ≡ *array-assn minimize-status-assn* ×$_a$ *arl64-assn atom-assn*›

sepref-register *conflict-min-cach-l*
sepref-def *delete-from-lookup-conflict-code*
is ‹*uncurry delete-from-lookup-conflict*›
:: ‹*unat-lit-assn*^{k} *$*_a$ *lookup-clause-rel-assn*^{d} →$_a$ *lookup-clause-rel-assn*›
unfolding *delete-from-lookup-conflict-def NOTIN-def*[*symmetric*]
conflict-option-rel-assn-def
lookup-clause-rel-assn-def
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

lemma *arena-is-valid-clause-idx-le-unat64-max*:
‹*arena-is-valid-clause-idx be bd* ⟹
length be ≤ *snat64-max* ⟹
bd + *arena-length be bd* ≤ *snat64-max*›
‹*arena-is-valid-clause-idx be bd* ⟹ *length be* ≤ *snat64-max* ⟹
bd ≤ *snat64-max*›
using *arena-lifting*(*10*)[*of be - - bd*]
by (*fastforce simp*: *arena-lifting arena-is-valid-clause-idx-def*)+

lemma *add-to-lookup-conflict-alt-def*:
‹*RETURN oo add-to-lookup-conflict* = (λ*L* (*n*, *xs*). *RETURN* (*if xs* ! *atm-of L* = *NOTIN then n* + *1 else n*,
xs[*atm-of L* := *ISIN* (*is-pos L*)]))›
unfolding *add-to-lookup-conflict-def* **by** (*auto simp*: *fun-eq-iff*)

sepref-register *ISIN NOTIN atm-of add-to-lookup-conflict*

sepref-def *add-to-lookup-conflict-impl*
is ‹*uncurry* (*RETURN oo add-to-lookup-conflict*)›
:: ‹[λ(*L*, (*n*, *xs*)). *atm-of L* < *length xs* ∧ *n* + *1* ≤ *unat32-max*]$_a$
unat-lit-assn^k *$*_a$ (*lookup-clause-rel-assn*)^d → *lookup-clause-rel-assn*›
unfolding *add-to-lookup-conflict-alt-def lookup-clause-rel-assn-def*
is-NOTIN-alt-def[*symmetric*] *fold-is-None NOTIN-def*
apply (*rewrite at* ‹- + ⌑› *unat-const-fold*[**where** *'a* = ‹*32*›])
by *sepref*

lemma *isa-lookup-conflict-merge-alt-def*:
 ‹*isa-lookup-conflict-merge i0* = (λ*M N i zs clvls outl*.
 do {
  let xs = *the-lookup-conflict zs*;
  ASSERT(*arena-is-valid-clause-idx N i*);
  (-, *clvls*, *zs*, *outl*) ← *WHILE*$_T$λ(*i*::*nat*, *clvls* :: *nat*, *zs*, *outl*). *length* (*snd zs*) = *length* (*snd xs*) ∧ *Suc* (*fst zs*)
   (λ(*j* :: *nat*, *clvls*, *zs*, *outl*). *j* < *i* + *arena-length N i*)
   (λ(*j* :: *nat*, *clvls*, *zs*, *outl*). *do* {
    ASSERT(*j* < *length N*);
    ASSERT(*arena-lit-pre N j*);
    ASSERT(*get-level-pol-pre* (*M*, *arena-lit N j*));
  ASSERT(*get-level-pol M* (*arena-lit N j*) ≤ *Suc* (*unat32-max div 2*));
    ASSERT(*atm-of* (*arena-lit N j*) < *length* (*snd zs*));
    ASSERT(¬*is-in-lookup-conflict zs* (*arena-lit N j*) ⟶ *length outl* < *unat32-max*);
    let outl = *isa-outlearned-add M* (*arena-lit N j*) *zs outl*;
    let clvls = *isa-clvls-add M* (*arena-lit N j*) *zs clvls*;
    let zs = *add-to-lookup-conflict* (*arena-lit N j*) *zs*;
    RETURN(*Suc j*, *clvls*, *zs*, *outl*)
   })
   (*i* + *i0*, *clvls*, *xs*, *outl*);
  RETURN (*Some-lookup-conflict zs*, *clvls*, *outl*)
 })›
 unfolding *isa-lookup-conflict-merge-def Some-lookup-conflict-def*
  the-lookup-conflict-def
 by (*auto simp*: *fun-eq-iff*)

sepref-def *resolve-lookup-conflict-merge-fast-code*
 is ‹*uncurry5 isa-set-lookup-conflict-aa*›
 :: ‹[λ(((((*M*, *N*), *i*), (-, *xs*)), -), *out*).
     length N ≤ *snat64-max*]$_a$
   trail-pol-fast-assn^k *$*_a$ *arena-fast-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} *$*_a$ *conflict-option-rel-assn*^{d} *$*_a$
    uint32-nat-assn^k *$*_a$ *out-learned-assn*^{d} →
   conflict-option-rel-assn ×$_a$ *uint32-nat-assn* ×$_a$ *out-learned-assn*›
 supply
  literals-are-in-$\mathcal{L}_{in}$*-trail-get-level-unat32-max*[*dest*]
  arena-is-valid-clause-idx-le-unat64-max[*dest*]
 unfolding *isa-set-lookup-conflict-aa-def lookup-conflict-merge-def*
  PR-CONST-def nth-rll-def[*symmetric*]
  isa-outlearned-add-def isa-clvls-add-def
  isa-lookup-conflict-merge-alt-def
  fmap-rll-u-def[*symmetric*]
  fmap-rll-def[*symmetric*]
  is-NOTIN-def[*symmetric*] *add-0-right*
 apply (*rewrite at* ‹*RETURN* (¤, - ,-, -)› *Suc-eq-plus1*)
 apply (*rewrite at* ‹*RETURN* (- + ¤, - ,-, -)› *snat-const-fold*[**where** *'a* = ‹*64*›])
 apply (*rewrite* **in** ‹*If* - ¤› *unat-const-fold*[**where** *'a* = ‹*32*›])
 supply [[*goals-limit* = *1*]]
 unfolding *fold-tuple-optimizations*
 by *sepref*

sepref-register *isa-resolve-merge-conflict-gt2*

lemma *arena-is-valid-clause-idx-le-unat64-max2*:
 ‹*arena-is-valid-clause-idx be bd* ⟹

length be ≤ *snat64-max* ⟹
bd + *arena-length be bd* ≤ *snat64-max*›
‹*arena-is-valid-clause-idx be bd* ⟹ *length be* ≤ *snat64-max* ⟹
bd < *snat64-max*›
using *arena-lifting*(*10*)[*of be - - bd*]
apply (*fastforce simp*: *arena-lifting arena-is-valid-clause-idx-def*)
using *arena-lengthI*(*2*) *less-le-trans* **by** *blast*

sepref-def *resolve-merge-conflict-fast-code*
is ‹*uncurry5 isa-resolve-merge-conflict-gt2*›
:: ‹[*uncurry5* (λ*M N i* (*b*, *xs*) *clvls outl*. *length N* ≤ *snat64-max*)]$_a$
trail-pol-fast-assn^k *∗$_a$ arena-fast-assn*^{k} *∗$_a$ sint64-nat-assn*^{k} *∗$_a$ conflict-option-rel-assn*^{d} *∗$_a$*
uint32-nat-assn^k *∗$_a$ out-learned-assn*^{d} →
conflict-option-rel-assn ×$_a$ *uint32-nat-assn* ×$_a$ *out-learned-assn*›
supply
literals-are-in-$\mathcal{L}_{in}$*-trail-get-level-unat32-max*[*dest*]
fmap-length-rll-u-def[*simp*]
arena-is-valid-clause-idx-le-unat64-max[*intro*]
arena-is-valid-clause-idx-le-unat64-max2[*dest*]
unfolding *isa-resolve-merge-conflict-gt2-def lookup-conflict-merge-def*
PR-CONST-def nth-rll-def[*symmetric*]
isa-outlearned-add-def isa-clvls-add-def
isa-lookup-conflict-merge-alt-def
fmap-rll-u-def[*symmetric*]
fmap-rll-def[*symmetric*]
is-NOTIN-def[*symmetric*] *add-0-right*
apply (*rewrite at* ‹*RETURN* (¤, - ,-, -)› *Suc-eq-plus1*)
apply (*rewrite at* ‹*WHILEIT* - - - (- + ¤, - ,-, -)› *snat-const-fold*[**where** *'a* = ‹*64*›])
apply (*rewrite at* ‹*RETURN* (- + ¤, - ,-, -)› *snat-const-fold*[**where** *'a* = ‹*64*›])
apply (*rewrite* **in** ‹*If* - ¤› *unat-const-fold*[**where** *'a* = ‹*32*›])
supply [[*goals-limit* = *1*]]
unfolding *fold-tuple-optimizations*
by *sepref*

sepref-def *atm-in-conflict-code*
is ‹*uncurry* (*RETURN oo atm-in-conflict-lookup*)›
:: ‹[*uncurry atm-in-conflict-lookup-pre*]$_a$
atom-assn^k *∗$_a$ lookup-clause-rel-assn*^{k} → *bool1-assn*›
unfolding *atm-in-conflict-lookup-def atm-in-conflict-lookup-pre-def*
is-NOTIN-alt-def[*symmetric*] *fold-is-None NOTIN-def lookup-clause-rel-assn-def*
apply (*rewrite at* ‹ - ! -› *annot-index-of-atm*)
by *sepref*

sepref-def *conflict-min-cach-l-code*
is ‹*uncurry* (*RETURN oo conflict-min-cach-l*)›
:: ‹[*conflict-min-cach-l-pre*]$_a$ *cach-refinement-l-assn*^{k} *∗$_a$ atom-assn*^{k} → *minimize-status-assn*›
unfolding *conflict-min-cach-l-def conflict-min-cach-l-pre-def cach-refinement-l-assn-def*
apply (*rewrite at* ‹*nth* -› *eta-expand*)
apply (*rewrite at* ‹ - ! -› *annot-index-of-atm*)
by *sepref*

lemma *conflict-min-cach-set-failed-l-alt-def*:
‹*conflict-min-cach-set-failed-l* = (λ(*cach*, *sup*) *L*. *do* {
ASSERT(*L* < *length cach*);

ASSERT(*length sup* ≤ *1* + *unat32-max div 2*);
let b = (*cach* ! *L* = *SEEN-UNKNOWN*);
RETURN (*cach*[*L* := *SEEN-FAILED*], *if b then sup* @ [*L*] *else sup*)
})›
unfolding *conflict-min-cach-set-failed-l-def Let-def* **by** *auto*

lemma *le-unat32-max-div2-le-unat32-max*: ‹*a2'* ≤ *Suc* (*unat32-max div 2*) ⟹ *a2'* < *unat32-max*›
by (*auto simp*: *unat32-max-def*)

sepref-def *conflict-min-cach-set-failed-l-code*
is ‹*uncurry conflict-min-cach-set-failed-l*›
:: ‹*cach-refinement-l-assn*^{d} *∗*$_a$ *atom-assn*^{k} →$_a$ *cach-refinement-l-assn*›
supply [[*goals-limit=1*]] *le-unat32-max-div2-le-unat32-max*[*dest*]
unfolding *conflict-min-cach-set-failed-l-alt-def*
minimize-status-rel-eq-def[*symmetric*] *cach-refinement-l-assn-def*

apply (*rewrite at* ‹ - ! -› *annot-index-of-atm*)
apply (*rewrite at* ‹*list-update* - - -› *annot-index-of-atm*)
by *sepref*

lemma *conflict-min-cach-set-removable-l-alt-def*:
‹*conflict-min-cach-set-removable-l* = (λ(*cach*, *sup*) *L*. *do* {
ASSERT(*L* < *length cach*);
ASSERT(*length sup* ≤ *1* + *unat32-max div 2*);
let b = (*cach* ! *L* = *SEEN-UNKNOWN*);
RETURN (*cach*[*L* := *SEEN-REMOVABLE*], *if b then sup* @ [*L*] *else sup*)
})›
unfolding *conflict-min-cach-set-removable-l-def* **by** *auto*

sepref-def *conflict-min-cach-set-removable-l-code*
is ‹*uncurry conflict-min-cach-set-removable-l*›
:: ‹*cach-refinement-l-assn*^{d} *∗*$_a$ *atom-assn*^{k} →$_a$ *cach-refinement-l-assn*›
unfolding *conflict-min-cach-set-removable-l-alt-def*
minimize-status-rel-eq-def[*symmetric*] *cach-refinement-l-assn-def*
apply (*rewrite at* ‹ - ! -› *annot-index-of-atm*)
apply (*rewrite at* ‹*list-update* - - -› *annot-index-of-atm*)
by *sepref*

lemma *lookup-conflict-size-impl-alt-def*:
‹*RETURN o* (λ(*n*, *xs*). *n*) = (λ(*n*, *xs*). *RETURN n*)›
by *auto*

lemma *single-replicate*: ‹[*C*] = *op-list-append* [] *C*›
by *auto*

sepref-register *lookup-conflict-remove1*

sepref-register *isa-lit-redundant-rec-wl-lookup*

sepref-register *isa-mark-failed-lits-stack*

sepref-register *lit-redundant-rec-wl-lookup conflict-min-cach-set-removable-l*
get-propagation-reason-pol lit-redundant-reason-stack-wl-lookup

sepref-register *isa-minimize-and-extract-highest-lookup-conflict isa-literal-redundant-wl-lookup*

lemma *set-lookup-empty-conflict-to-none-alt-def*:
‹*RETURN o set-lookup-empty-conflict-to-none* = (λ(*n*, *xs*). *RETURN* (*True*, *n*, *xs*))›
by (*auto simp*: *set-lookup-empty-conflict-to-none-def*)

sepref-def *set-lookup-empty-conflict-to-none-imple*
is ‹*RETURN o set-lookup-empty-conflict-to-none*›
:: ‹*lookup-clause-rel-assn*^{d} →$_a$ *conflict-option-rel-assn*›
unfolding *set-lookup-empty-conflict-to-none-alt-def*
lookup-clause-rel-assn-def conflict-option-rel-assn-def
by *sepref*

lemma *isa-mark-failed-lits-stackI*:
assumes
‹*length ba* ≤ *Suc* (*unat32-max div 2*)› **and**
‹*a1'* < *length ba*›
shows ‹*Suc a1'* ≤ *unat32-max*›
using *assms* **by** (*auto simp*: *unat32-max-def*)

sepref-register *conflict-min-cach-set-failed-l*
sepref-def *isa-mark-failed-lits-stack-fast-code*
is ‹*uncurry2* (*isa-mark-failed-lits-stack*)›
:: ‹[λ((*N*, -), -). *length N* ≤ *snat64-max*]$_a$
arena-fast-assn^k *$*_a$ *analyse-refinement-fast-assn*^{k} *$*_a$ *cach-refinement-l-assn*^{d} →
cach-refinement-l-assn›
supply [[*goals-limit* = *1*]] *neq-Nil-revE*[*elim*!] *image-image*[*simp*]
mark-failed-lits-stack-inv-helper1[*dest*] *mark-failed-lits-stack-inv-helper2*[*dest*]
fmap-length-rll-u-def[*simp*] *isa-mark-failed-lits-stackI*[*intro*]
arena-is-valid-clause-idx-le-unat64-max[*intro*] *le-unat32-max-div2-le-unat32-max*[*intro*]
unfolding *isa-mark-failed-lits-stack-def PR-CONST-def*
conflict-min-cach-set-failed-def[*symmetric*]
conflict-min-cach-def[*symmetric*]
get-literal-and-remove-of-analyse-wl-def
nth-rll-def[*symmetric*]
fmap-rll-def[*symmetric*]
arena-lit-def[*symmetric*]
minimize-status-rel-eq-def[*symmetric*]
apply (*rewrite at 1* **in** ‹*conflict-min-cach-set-failed-l* - ¤› *snat-const-fold*[**where** *'a* = ‹*64*›])
apply (*rewrite* **in** ‹*RETURN* (- + ¤, -)› *snat-const-fold*[**where** *'a* = ‹*64*›])
apply (*rewrite at 0* **in** ‹(¤, -)› *snat-const-fold*[**where** *'a* = ‹*64*›])
apply (*rewrite at* ‹*arena-lit* - (- + ¤ − -)› *annot-unat-snat-upcast*[**where** *'l* = *64*])
by *sepref*

sepref-def *isa-get-literal-and-remove-of-analyse-wl-fast-code*
is ‹*uncurry* (*RETURN oo isa-get-literal-and-remove-of-analyse-wl*)›
:: ‹[λ(*arena*, *analyse*). *isa-get-literal-and-remove-of-analyse-wl-pre arena analyse* ∧
length arena ≤ *snat64-max*]$_a$
arena-fast-assn^k *$*_a$ *analyse-refinement-fast-assn*^{d} →
unat-lit-assn ×$_a$ *analyse-refinement-fast-assn*›
supply [[*goals-limit=1*]] *arena-lit-pre-le2*[*dest*]
and [*dest*] = *arena-lit-implI*
unfolding *isa-get-literal-and-remove-of-analyse-wl-pre-def*
isa-get-literal-and-remove-of-analyse-wl-def

apply (*rewrite at ‹length - − ¤›* *snat-const-fold*[**where** *'a=64*])
apply (*rewrite at ‹arena-lit - (- + ¤)›* *annot-unat-snat-upcast*[**where** *'l = 64*])
apply (*annot-unat-const ‹TYPE(32)›*)
by *sepref*

sepref-def *ana-lookup-conv-lookup-fast-code*
is ‹*uncurry* (*RETURN oo ana-lookup-conv-lookup*)›
:: ‹[*uncurry ana-lookup-conv-lookup-pre*]$_a$ *arena-fast-assn*^{k} $*_a$
(*ana-refinement-fast-assn*)^k
$\to$ *sint64-nat-assn* $\times_a$ *sint64-nat-assn* $\times_a$ *sint64-nat-assn* $\times_a$ *sint64-nat-assn*›
unfolding *ana-lookup-conv-lookup-pre-def ana-lookup-conv-lookup-def*
apply (*rewrite at ‹(-, -, ¤, -)›* *annot-unat-snat-upcast*[**where** *'l = 64*])
apply (*annot-snat-const ‹TYPE(64)›*)
by *sepref*

sepref-register *arena-lit*
sepref-def *lit-redundant-reason-stack-wl-lookup-fast-code*
is ‹*uncurry2* (*RETURN ooo lit-redundant-reason-stack-wl-lookup*)›
:: ‹[*uncurry2 lit-redundant-reason-stack-wl-lookup-pre*]$_a$
unat-lit-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\to$
ana-refinement-fast-assn›
unfolding *lit-redundant-reason-stack-wl-lookup-def lit-redundant-reason-stack-wl-lookup-pre-def*
apply (*rewrite at ‹¤ < -›* *snat-const-fold*[**where** *'a=64*])
apply (*annot-unat-const ‹TYPE(32)›*)
by *sepref*

lemma *isa-lit-redundant-rec-wl-lookupI*:
assumes
‹*length ba* $\leq$ *Suc* (*unat32-max div 2*)›
shows ‹*length ba* < *unat32-max*›
using *assms* **by** (*auto simp*: *unat32-max-def*)

lemma *arena-lit-pre-le*: ‹
arena-lit-pre a i $\Longrightarrow$ *length a* $\leq$ *snat64-max* $\Longrightarrow$ *i* $\leq$ *snat64-max*›
using *arena-lifting*(*7*)[*of a - -*] **unfolding** *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
by *fastforce*

lemma *get-propagation-reason-pol-get-propagation-reason-pol-raw*: ‹*do* {
C ← *get-propagation-reason-pol M* (−*L*);
case C of
Some C ⇒ *f C*
| *None* ⇒ *g*
} = *do* {
C ← *get-propagation-reason-raw-pol M* (−*L*);
if C ≠ *DECISION-REASON then f C else g*
}›
by (*cases M*) (*auto simp*: *get-propagation-reason-pol-def get-propagation-reason-raw-pol-def*)

sepref-register *atm-in-conflict-lookup*
sepref-def *lit-redundant-rec-wl-lookup-fast-code*
is ‹*uncurry5* (*isa-lit-redundant-rec-wl-lookup*)›
:: ‹[λ(((((*M*, *NU*), *D*), *cach*), *analysis*), *lbd*). *length NU* ≤ *snat64-max*]$_a$
trail-pol-fast-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ (*lookup-clause-rel-assn*)^k $*_a$
cach-refinement-l-assn^d $*_a$ *analyse-refinement-fast-assn*^{d} $*_a$ *lbd-assn*^{k} $\to$

cach-refinement-l-assn $\times_a$ *analyse-refinement-fast-assn* $\times_a$ *bool1-assn*›
supply [[*goals-limit* = *1*]] *neq-Nil-revE*[*elim*] *image-image*[*simp*]
literals-are-in-$\mathcal{L}_{in}$*-trail-uminus-in-lits-of-l*[*intro*]
literals-are-in-$\mathcal{L}_{in}$*-trail-in-lits-of-l-atms*[*intro*]
literals-are-in-$\mathcal{L}_{in}$*-trail-uminus-in-lits-of-l-atms*[*intro*] *nth-rll-def*[*simp*]
fmap-length-rll-u-def[*simp*]
isa-lit-redundant-rec-wl-lookupI[*intro*]
arena-lit-pre-le[*dest*] *isa-mark-failed-lits-stackI*[*intro*]

unfolding *isa-lit-redundant-rec-wl-lookup-def*
conflict-min-cach-set-removable-def[*symmetric*]
conflict-min-cach-def[*symmetric*]
get-literal-and-remove-of-analyse-wl-def
nth-rll-def[*symmetric*] *PR-CONST-def*
fmap-rll-u-def[*symmetric*] *minimize-status-rel-eq-def*[*symmetric*]
fmap-rll-def[*symmetric*] *length-0-conv*[*symmetric*]
apply (*subst get-propagation-reason-pol-get-propagation-reason-pol-raw*)
apply (*rewrite at* ‹*get-level-pol - - =* ¤› *unat-const-fold*[**where** *'a=32*])
apply (*rewrite at* ‹(-, ¤, -)› *annotate-assn*[**where** *A=analyse-refinement-fast-assn*])
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
unfolding *nth-rll-def*[*symmetric*]
fmap-rll-def[*symmetric*]
fmap-length-rll-def[*symmetric*]
unfolding *nth-rll-def*[*symmetric*]
fmap-rll-def[*symmetric*]
fmap-length-rll-def[*symmetric*]
fmap-rll-u-def[*symmetric*]
by *sepref*

sepref-def *delete-index-and-swap-code*
is ‹*uncurry* (*RETURN oo delete-index-and-swap*)›
:: ‹[*λ*(*xs*, *i*). *i* < *length xs*]$_a$
(*arl64-assn unat-lit-assn*)^d *$*_a$ *sint64-nat-assn*^{k} → *arl64-assn unat-lit-assn*›
unfolding *delete-index-and-swap.simps*
by *sepref*

sepref-def *lookup-conflict-upd-None-code*
is ‹*uncurry* (*RETURN oo lookup-conflict-upd-None*)›
:: ‹[*λ*((*n*, *xs*), *i*). *i* < *length xs* ∧ *n* > *0*]$_a$
lookup-clause-rel-assn^d *$*_a$ *sint32-nat-assn*^{k} → *lookup-clause-rel-assn*›
unfolding *lookup-conflict-upd-None-RETURN-def lookup-clause-rel-assn-def*
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

lemma *unat32-max-ge0*: ‹*0* < *unat32-max*› **by** (*auto simp*: *unat32-max-def*)

sepref-def *literal-redundant-wl-lookup-fast-code*
is ‹*uncurry5 isa-literal-redundant-wl-lookup*›
:: ‹[*λ*(((((*M*, *NU*), *D*), *cach*), *L*), *lbd*). *length NU* ≤ *snat64-max*]$_a$
trail-pol-fast-assn^k *$*_a$ *arena-fast-assn*^{k} *$*_a$ *lookup-clause-rel-assn*^{k} *$*_a$
cach-refinement-l-assn^d *$*_a$ *unat-lit-assn*^{k} *$*_a$ *lbd-assn*^{k} →
cach-refinement-l-assn $\times_a$ *analyse-refinement-fast-assn* $\times_a$ *bool1-assn*›
supply [[*goals-limit=1*]]
literals-are-in-$\mathcal{L}_{in}$*-trail-uminus-in-lits-of-l*[*intro*] *unat32-max-ge0*[*intro*!]

literals-are-in-$\mathcal{L}_{in}$-trail-uminus-in-lits-of-l-atms[*intro*]
unfolding *isa-literal-redundant-wl-lookup-def PR-CONST-def*
minimize-status-rel-eq-def[*symmetric*]
apply (*rewrite at* ‹(-, ¤, -)› *al-fold-custom-empty*[**where** *'l=64*])+
unfolding *single-replicate*
apply (*rewrite at* ‹*get-level-pol - - = ¤*› *unat-const-fold*[**where** *'a=32*])
unfolding *al-fold-custom-empty*[**where** *'l=64*]
apply (*subst get-propagation-reason-pol-get-propagation-reason-pol-raw*)
by *sepref*

sepref-def *conflict-remove1-code*
is ‹*uncurry* (*RETURN oo lookup-conflict-remove1*)›
:: ‹[*lookup-conflict-remove1-pre*]$_a$ *unat-lit-assn*^{k} *$*_a$ *lookup-clause-rel-assn*^{d} → *lookup-clause-rel-assn*›
supply [[*goals-limit=2*]]
unfolding *lookup-conflict-remove1-def lookup-conflict-remove1-pre-def lookup-clause-rel-assn-def*
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

sepref-def *minimize-and-extract-highest-lookup-conflict-fast-code*
is ‹*uncurry5 isa-minimize-and-extract-highest-lookup-conflict*›
:: ‹[λ(((((*M*, *NU*), *D*), *cach*), *lbd*), *outl*). *length NU* ≤ *snat64-max*]$_a$
trail-pol-fast-assn^k *$*_a$ *arena-fast-assn*^{k} *$*_a$ *lookup-clause-rel-assn*^{d} *$*_a$
cach-refinement-l-assn^d *$*_a$ *lbd-assn*^{k} *$*_a$ *out-learned-assn*^{d} →
lookup-clause-rel-assn ×$_a$ *cach-refinement-l-assn* ×$_a$ *out-learned-assn*›
supply [[*goals-limit=1*]]
literals-are-in-$\mathcal{L}_{in}$-trail-uminus-in-lits-of-l[*intro*]
minimize-and-extract-highest-lookup-conflict-inv-def[*simp*]
in-$\mathcal{L}_{all}$-less-unat32-max'[*intro*]
unfolding *isa-minimize-and-extract-highest-lookup-conflict-def*
PR-CONST-def
minimize-and-extract-highest-lookup-conflict-inv-def
apply (*rewrite at* ‹(-, ¤, -, -)› *snat-const-fold*[**where** *'a = 64*])
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *isasat-lookup-merge-eq2-alt-def*:
‹*isasat-lookup-merge-eq2 L M N C* = (λ*zs clvls outl*. *do* {
let zs = *the-lookup-conflict zs*;
ASSERT(*arena-lit-pre N C*);
ASSERT(*arena-lit-pre N* (*C*+*1*));
let L0 = *arena-lit N C*;
let L' = (*if L0* = *L then arena-lit N* (*C* + *1*) *else L0*);
ASSERT(*get-level-pol-pre* (*M*, *L'*));
ASSERT(*get-level-pol M L'* ≤ *Suc* (*unat32-max div 2*));
ASSERT(*atm-of L'* < *length* (*snd zs*));
ASSERT(*length outl* < *unat32-max*);
let outl = *isa-outlearned-add M L' zs outl*;
ASSERT(*clvls* < *unat32-max*);
ASSERT(*fst zs* < *unat32-max*);
let clvls = *isa-clvls-add M L' zs clvls*;
let zs = *add-to-lookup-conflict L' zs*;
RETURN(*Some-lookup-conflict zs*, *clvls*, *outl*)

})›
by (*auto simp*: *the-lookup-conflict-def Some-lookup-conflict-def Let-def*
isasat-lookup-merge-eq2-def fun-eq-iff)

sepref-def *isasat-lookup-merge-eq2-fast-code*
is ‹*uncurry6 isasat-lookup-merge-eq2*›
:: ‹[λ((((((*L*, *M*), *NU*), -), -), -), -). *length NU* ≤ *snat64-max*]$_a$
unat-lit-assn^k *$*_a$ *trail-pol-fast-assn*^{k} *$*_a$ *arena-fast-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} *$*_a$
conflict-option-rel-assn^d *$*_a$ *uint32-nat-assn*^{k} *$*_a$ *out-learned-assn*^{d} →
conflict-option-rel-assn ×$_a$ *uint32-nat-assn* ×$_a$ *out-learned-assn*›
supply [[*goals-limit* = *1*]]
unfolding *isasat-lookup-merge-eq2-alt-def*
isa-outlearned-add-def isa-clvls-add-def
is-NOTIN-def[*symmetric*]
supply
image-image[*simp*] *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$[*simp*]
literals-are-in-$\mathcal{L}_{in}$*-trail-get-level-unat32-max*[*dest*]
fmap-length-rll-u-def[*simp*] *the-lookup-conflict-def*[*simp*]
arena-is-valid-clause-idx-le-unat64-max[*dest*]
arena-lit-pre-le2[*dest*] *arena-lit-pre-le*[*dest*]
apply (*rewrite* **in** ‹*if* - *then* - + ⌑ *else* -› *unat-const-fold*[**where** *'a=32*])
apply (*rewrite* **in** ‹*if* - *then arena-lit* - (- + ⌑) *else* -› *snat-const-fold*[**where** *'a=64*])
by *sepref*

sepref-def *is-in-option-lookup-conflict-code*
is ‹*uncurry* (*RETURN oo is-in-option-lookup-conflict*)›
:: ‹[λ(*L*, (*c*, *n*, *xs*)). *atm-of L* < *length xs*]$_a$
unat-lit-assn^k *$*_a$ *conflict-option-rel-assn*^{k} → *bool1-assn*›
unfolding *is-in-option-lookup-conflict-alt-def is-in-lookup-conflict-def PROTECT-def*
is-NOTIN-alt-def[*symmetric*] *conflict-option-rel-assn-def lookup-clause-rel-assn-def*
by *sepref*

definition *None-lookup-conflict* :: ‹- ⇒ - ⇒ *conflict-option-rel*› **where**
‹*None-lookup-conflict b xs* = (*b*, *xs*)›

sepref-def *None-lookup-conflict-impl*
is ‹*uncurry* (*RETURN oo None-lookup-conflict*)›
:: ‹*bool1-assn*^{k} *$*_a$ *lookup-clause-rel-assn*^{d} →$_a$ *conflict-option-rel-assn*›
unfolding *None-lookup-conflict-def conflict-option-rel-assn-def*
lookup-clause-rel-assn-def
by *sepref*

sepref-register *None-lookup-conflict*
declare *None-lookup-conflict-impl.refine*[*sepref-fr-rules*]

schematic-goal *mk-free-lookup-clause-rel-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE lookup-clause-rel-assn ?fr*›
unfolding *conflict-option-rel-assn-def lookup-clause-rel-assn-def*
by *synthesize-free*

schematic-goal *mk-free-cach-refinement-l-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE cach-refinement-l-assn ?fr*›
unfolding *cach-refinement-l-assn-def*
by *synthesize-free*

schematic-goal *mk-free-trail-pol-fast-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE conflict-option-rel-assn ?fr*›
  unfolding *conflict-option-rel-assn-def*
  by *synthesize-free*

experiment begin

export-llvm
  nat-lit-eq-impl
  minimize-status-rel-eq-impl
  SEEN-FAILED-impl
  SEEN-UNKNOWN-impl
  SEEN-REMOVABLE-impl
  Some-impl
  is-Notin-impl
  NOTIN-impl
  lookup-clause-assn-is-None-impl
  size-lookup-conflict-impl
  is-in-conflict-code
  lookup-clause-assn-is-empty-impl
  the-lookup-conflict-impl
  Some-lookup-conflict-impl
  delete-from-lookup-conflict-code
  add-to-lookup-conflict-impl
  resolve-lookup-conflict-merge-fast-code
  resolve-merge-conflict-fast-code
  atm-in-conflict-code
  conflict-min-cach-l-code
  conflict-min-cach-set-failed-l-code
  conflict-min-cach-set-removable-l-code
  set-lookup-empty-conflict-to-none-imple
  isa-mark-failed-lits-stack-fast-code
  isa-get-literal-and-remove-of-analyse-wl-fast-code
  ana-lookup-conv-lookup-fast-code
  lit-redundant-reason-stack-wl-lookup-fast-code
  lit-redundant-rec-wl-lookup-fast-code
  delete-index-and-swap-code
  lookup-conflict-upd-None-code
  literal-redundant-wl-lookup-fast-code
  conflict-remove1-code
  minimize-and-extract-highest-lookup-conflict-fast-code
  isasat-lookup-merge-eq2-fast-code

end

end
theory *IsaSAT-VMTF-Setup-LLVM*
imports *IsaSAT-Setup IsaSAT-Literals-LLVM*
begin

type-synonym *vmtf-node-assn* = ‹(*64 word* × *32 word* × *32 word*)›

definition ‹*vmtf-node1-rel* ≡ { ((*a*,*b*,*c*),(*VMTF-Node a b c*)) | *a b c*. *True*}›
definition ‹*vmtf-node2-assn* ≡ *uint64-nat-assn* ×$_a$ *atom.option-assn* ×$_a$ *atom.option-assn*›

definition ‹*vmtf-node-assn* ≡ *hr-comp vmtf-node2-assn vmtf-node1-rel*›
lemmas [*fcomp-norm-unfold*] = *vmtf-node-assn-def*[*symmetric*]

lemma *vmtf-node-assn-pure*[*safe-constraint-rules*]: ‹*CONSTRAINT is-pure vmtf-node-assn*›
  unfolding *vmtf-node-assn-def vmtf-node2-assn-def*
  by *solve-constraint*

lemmas [*sepref-frame-free-rules*] = *mk-free-is-pure*[*OF vmtf-node-assn-pure*[*unfolded CONSTRAINT-def*]]

lemma
    vmtf-Node-refine1: ‹(λ*a b c*. (*a*,*b*,*c*), *VMTF-Node*) ∈ *Id* → *Id* → *Id* → *vmtf-node1-rel*›
and *vmtf-stamp-refine1*: ‹(λ(*a*,*b*,*c*). *a*, *stamp*) ∈ *vmtf-node1-rel* → *Id*›
and *vmtf-get-prev-refine1*: ‹(λ(*a*,*b*,*c*). *b*, *get-prev*) ∈ *vmtf-node1-rel* → ⟨*Id*⟩*option-rel*›
and *vmtf-get-next-refine1*: ‹(λ(*a*,*b*,*c*). *c*, *get-next*) ∈ *vmtf-node1-rel* → ⟨*Id*⟩*option-rel*›
  by (*auto simp*: *vmtf-node1-rel-def*)

sepref-def *VMTF-Node-impl* **is** []
  ‹*uncurry2* (*RETURN ooo* (λ*a b c*. (*a*,*b*,*c*)))›
  :: ‹*uint64-nat-assn*^{k} *$*_a$* (*atom.option-assn*)^k *$*_a$* (*atom.option-assn*)^k →$_a$ *vmtf-node2-assn*›
  unfolding *vmtf-node2-assn-def* **by** *sepref*

sepref-def *VMTF-stamp-impl*
  is [] ‹*RETURN o* (λ(*a*,*b*,*c*). *a*)›
  :: ‹*vmtf-node2-assn*^{k} →$_a$ *uint64-nat-assn*›
  unfolding *vmtf-node2-assn-def*
  by *sepref*

sepref-def *VMTF-get-prev-impl*
  is [] ‹*RETURN o* (λ(*a*,*b*,*c*). *b*)›
  :: ‹*vmtf-node2-assn*^{k} →$_a$ *atom.option-assn*›
  unfolding *vmtf-node2-assn-def*
  by *sepref*

sepref-def *VMTF-get-next-impl*
  is [] ‹*RETURN o* (λ(*a*,*b*,*c*). *c*)›
  :: ‹*vmtf-node2-assn*^{k} →$_a$ *atom.option-assn*›
  unfolding *vmtf-node2-assn-def*
  by *sepref*

lemma *workaround-hrcomp-id-norm*[*fcomp-norm-unfold*]: ‹*hr-comp R* (⟨*nat-rel*⟩*option-rel*) = *R*› **by** *simp*

lemmas [*sepref-fr-rules*] =
  VMTF-Node-impl.refine[*FCOMP vmtf-Node-refine1*]
  VMTF-stamp-impl.refine[*FCOMP vmtf-stamp-refine1*]
  VMTF-get-prev-impl.refine[*FCOMP vmtf-get-prev-refine1*]
  VMTF-get-next-impl.refine[*FCOMP vmtf-get-next-refine1*]

type-synonym *vmtf-assn* = ‹*vmtf-node-assn ptr* × *64 word* × *32 word* × *32 word* × *32 word*›

type-synonym *vmtf-remove-assn* = ‹*vmtf-assn* × (*32 word array-list64* × *1 word ptr*)›

definition *vmtf-assn* :: ‹- ⇒ *vmtf-assn* ⇒ *assn*› **where**
‹*vmtf-assn* ≡ (*array-assn vmtf-node-assn* ×$_a$ *uint64-nat-assn* ×$_a$ *atom-assn* ×$_a$ *atom-assn*
×$_a$ *atom.option-assn*)›

abbreviation *atoms-hash-assn* :: ‹*bool list* ⇒ *1 word ptr* ⇒ *assn*› **where**
‹*atoms-hash-assn* ≡ *array-assn bool1-assn*›

abbreviation *distinct-atoms-assn* **where**
‹*distinct-atoms-assn* ≡ *arl64-assn atom-assn* ×$_a$ *atoms-hash-assn*›

sepref-def *vmtf-heur-fst-code*
is ‹*RETURN o vmtf-heur-fst*›
:: ‹*vmtf-assn*^{k} →$_a$ *atom-assn*›
unfolding *vmtf-heur-fst-def vmtf-assn-def*
by *sepref*

definition *vmtf-heur-array-nth* :: ‹*vmtf* ⇒ -›**where**
‹*vmtf-heur-array-nth* = (λ(*ns*, -, -, -) *i*. *RETURN* (*ns* ! *i*))›

sepref-def *vmtf-heur-array-nth-code*
is ‹*uncurry* (*vmtf-heur-array-nth*)›
:: ‹[λ(*vm*,*i*). *i* < *length* (*fst vm*)]$_a$ *vmtf-assn*^{k} ∗$_a$ *atom-assn*^{k} → *vmtf-node-assn*›
supply [[*eta-contract* = *false*, *goals-limit*=*1*]]
supply [*sepref-fr-rules*] = *al-nth-hnr array-get-hnr*
unfolding *vmtf-heur-array-nth-def vmtf-assn-def*
apply (*rewrite at* ‹(!) - ⌑› *value-of-atm-def*[*symmetric*])
unfolding *index-of-atm-def*[*symmetric*]
by *sepref*

end
theory *IsaSAT-VDom-LLVM*
imports *IsaSAT-VDom IsaSAT-Stats-LLVM IsaSAT-Clauses-LLVM IsaSAT-Arena-Sorting-LLVM*
begin
hide-const (**open**) *NEMonad.ASSERT NEMonad.RETURN NEMonad.SPEC*
type-synonym *aivdom2* = ‹*vdom* × *vdom* × *vdom*›
abbreviation *aivdom-int-rel* :: ‹(*aivdom2* × *aivdom*) *set*› **where**
‹*aivdom-int-rel* ≡ {(*a*, (-, *a'*)). (*a*,*a'*) ∈ ⟨*nat-rel*⟩*list-rel* ×$_r$ ⟨*nat-rel*⟩*list-rel* ×$_r$ ⟨*nat-rel*⟩*list-rel*}›

abbreviation *aivdom-rel* :: ‹(*aivdom2* × *isasat-aivdom*) *set*› **where**
‹*aivdom-rel* ≡ ⟨*aivdom-int-rel*⟩*code-hider-rel*›

abbreviation *aivdom-int-assn* :: ‹*aivdom2* ⇒ - ⇒ *assn*› **where**
‹*aivdom-int-assn* ≡ *LBD-it.arr-assn* ×$_a$ *LBD-it.arr-assn* ×$_a$ *LBD-it.arr-assn*›
type-synonym *aivdom-assn* = ‹*vdom-fast-assn* × *vdom-fast-assn* × *vdom-fast-assn*›
definition *aivdom-assn* :: ‹*isasat-aivdom* ⇒ - ⇒ *assn*› **where**
‹*aivdom-assn* = *code-hider-assn aivdom-int-assn aivdom-int-rel*›

To keep my sanity, I use the same name, even if the function drops one component.

definition *add-learned-clause-aivdom-int* **where**
‹*add-learned-clause-aivdom-int* = (λ *C* (*avdom*, *ivdom*). (*avdom* @ [*C*], *ivdom*))›

definition *add-learned-clause-aivdom-strong-int* **where**
‹*add-learned-clause-aivdom-strong-int* = (λ *C* (*avdom*, *ivdom*, *tvdom*). (*avdom* @ [*C*], *ivdom*, *tvdom*

@ [C]))›

definition *add-init-clause-aivdom-strong-int* **where**
‹*add-init-clause-aivdom-strong-int* = (λ *C* (*avdom*, *ivdom*, *tvdom*). (*avdom*, *ivdom* @ [*C*], *tvdom* @ [*C*]))›

definition *remove-inactive-aivdom-int* :: ‹- ⇒ *aivdom2* ⇒ *aivdom2*› **where**
‹*remove-inactive-aivdom-int* = (λ*i* (*avdom*, *ivdom*). (*delete-index-and-swap avdom i*, *ivdom*))›

definition *remove-inactive-aivdom-tvdom-int* :: ‹- ⇒ *aivdom2* ⇒ *aivdom2*› **where**
‹*remove-inactive-aivdom-tvdom-int* = (λ*i* (*avdom*, *ivdom*, *tvdom*). (*avdom*, *ivdom*, *delete-index-and-swap tvdom i*))›

definition *avdom-aivdom-at-int* :: ‹*aivdom2* ⇒ *nat* ⇒ *nat*› **where**
‹*avdom-aivdom-at-int* = (λ(*b*,*c*) *C*. *b* ! *C*)›

definition *tvdom-aivdom-at-int* :: ‹*aivdom2* ⇒ *nat* ⇒ *nat*› **where**
‹*tvdom-aivdom-at-int* = (λ(*b*,*c*,*d*) *C*. *d* ! *C*)›

definition *length-ivdom-aivdom-int* :: ‹*aivdom2* ⇒ *nat*› **where**
‹*length-ivdom-aivdom-int* = (λ(*b*,*c*,*d*). *length c*)›

definition *ivdom-aivdom-at-int* :: ‹*aivdom2* ⇒ *nat* ⇒ *nat*› **where**
‹*ivdom-aivdom-at-int* = (λ(*b*,*c*,*d*) *C*. *c* ! *C*)›

definition *length-tvdom-aivdom-int* :: ‹*aivdom2* ⇒ *nat*› **where**
‹*length-tvdom-aivdom-int* = (λ(*b*,*c*,*d*). *length d*)›

definition *length-avdom-aivdom-int* :: ‹*aivdom2* ⇒ *nat*› **where**
‹*length-avdom-aivdom-int* = (λ(*b*,*c*,*d*). *length b*)›

definition *AIVDom-int* :: ‹- ⇒ - ⇒ - ⇒ - ⇒ *aivdom2*› **where**
‹*AIVDom-int* - *avdom ivdom tvdom* = (*avdom*, *ivdom*, *tvdom*)›

definition *swap-avdom-aivdom-int* :: ‹*aivdom2* ⇒ *nat* ⇒ *nat* ⇒ *aivdom2*› **where**
‹*swap-avdom-aivdom-int* = (λ(*avdom*, *ivdom*, *tvdom*) *i j*.
(*swap avdom i j*, *ivdom*, *tvdom*))›

lemma *swap-avdom-aivdom-alt-def*:
‹*swap-avdom-aivdom aivdom i j* =
(*AIvdom* (*get-vdom-aivdom aivdom*, *swap* (*get-avdom-aivdom aivdom*) *i j*, *get-ivdom-aivdom aivdom*, *get-tvdom-aivdom aivdom*))›
by (*cases aivdom*) *auto*

definition *take-avdom-aivdom-int* :: ‹*nat* ⇒ *aivdom2* ⇒ *aivdom2*› **where**
‹*take-avdom-aivdom-int* = (λ*i* (*avdom*, *ivdom*, *tvdom*).
(*take i avdom*, *ivdom*, *tvdom*))›

lemma *take-avdom-aivdom-alt-def*:
‹*take-avdom-aivdom i aivdom* =
(*AIvdom* (*get-vdom-aivdom aivdom*, *take i* (*get-avdom-aivdom aivdom*), *get-ivdom-aivdom aivdom*, *get-tvdom-aivdom aivdom*))›
by (*cases aivdom*) *auto*

definition *map-vdom-aivdom-int* :: ‹- ⇒ *aivdom2* ⇒ *aivdom2 nres*› **where**

‹*map-vdom-aivdom-int f* = (λ(*avdom*, *ivdom*, *tvdom*). *do* {
avdom ← *f avdom*;
RETURN ((*avdom*, *ivdom*, *tvdom*))
})›

definition *map-tvdom-aivdom-int* :: ‹- ⇒ *aivdom2* ⇒ *aivdom2 nres*› **where**
‹*map-tvdom-aivdom-int f* = (λ(*avdom*, *ivdom*, *tvdom*). *do* {
tvdom ← *f tvdom*;
RETURN ((*avdom*, *ivdom*, *tvdom*))
})›

definition *empty-aivdom-int* **where**
‹*empty-aivdom-int* = (λ(*avdom*, *ivdom*, *tvdom*). (*take 0 avdom*, *take 0 ivdom*, *take 0 tvdom*))›

definition *AIvdom-init-int* :: ‹*nat list* ⇒ *nat list* ⇒ *nat list* ⇒ *aivdom2*› **where**
‹*AIvdom-init-int vdom avdom ivdom* = (*avdom*, *ivdom*, *vdom*)›

definition *empty-tvdom-int* **where**
‹*empty-tvdom-int* = (λ(*avdom*, *ivdom*, *tvdom*). (*avdom*, *ivdom*, *take 0 tvdom*))›

definition *push-to-tvdom-int* :: ‹*nat* ⇒ *aivdom2* ⇒ *aivdom2*› **where**
‹*push-to-tvdom-int C* = (λ(*avdom*, *ivdom*, *tvdom*). (*avdom*, *ivdom*, *tvdom* @ [*C*]))›

lemma
add-learned-clause-aivdom-int:
‹(*uncurry* (*RETURN oo add-learned-clause-aivdom-int*), *uncurry* (*RETURN oo add-learned-clause-aivdom*))
∈ *nat-rel* ×*f* *aivdom-rel* →*f* ⟨*aivdom-rel*⟩*nres-rel*› **and**
add-learned-clause-aivdom-strong-int:
‹(*uncurry* (*RETURN oo add-learned-clause-aivdom-strong-int*), *uncurry* (*RETURN oo add-learned-clause-aivdom-strong
∈ *nat-rel* ×*f* *aivdom-rel* →*f* ⟨*aivdom-rel*⟩*nres-rel*› **and**
add-init-clause-aivdom-strong-int:
‹(*uncurry* (*RETURN oo add-init-clause-aivdom-strong-int*), *uncurry* (*RETURN oo add-init-clause-aivdom-strong*))
∈ *nat-rel* ×*f* *aivdom-rel* →*f* ⟨*aivdom-rel*⟩*nres-rel*› **and**
remove-inactive-aivdom-int:
‹(*uncurry* (*RETURN oo remove-inactive-aivdom-int*), *uncurry* (*RETURN oo remove-inactive-aivdom*))
∈ *nat-rel* ×*f* *aivdom-rel* →*f* ⟨*aivdom-rel*⟩*nres-rel*› **and**
remove-inactive-aivdom-tvdom-int:
‹(*uncurry* (*RETURN oo remove-inactive-aivdom-tvdom-int*), *uncurry* (*RETURN oo remove-inactive-aivdom-tvdom*))
∈ *nat-rel* ×*f* *aivdom-rel* →*f* ⟨*aivdom-rel*⟩*nres-rel*› **and**
avdom-aivdom-at-int:
‹(*uncurry* (*RETURN oo avdom-aivdom-at-int*), *uncurry* (*RETURN oo avdom-aivdom-at*)) ∈ *aivdom-rel*
×*f* *nat-rel* →*f* ⟨*nat-rel*⟩*nres-rel*›**and**
tvdom-aivdom-at-int:
‹(*uncurry* (*RETURN oo tvdom-aivdom-at-int*), *uncurry* (*RETURN oo tvdom-aivdom-at*)) ∈ *aivdom-rel*
×*f* *nat-rel* →*f* ⟨*nat-rel*⟩*nres-rel*›**and**
ivdom-aivdom-at-int:
‹(*uncurry* (*RETURN oo ivdom-aivdom-at-int*), *uncurry* (*RETURN oo ivdom-aivdom-at*)) ∈ *aivdom-rel*
×*f* *nat-rel* →*f* ⟨*nat-rel*⟩*nres-rel*›**and**
length-avdom-aivdom-int:
‹(*RETURN o length-avdom-aivdom-int*, *RETURN o length-avdom-aivdom*) ∈ *aivdom-rel* →*f* ⟨*nat-rel*⟩*nres-rel*›**and**
length-ivdom-aivdom-int:
‹(*RETURN o length-ivdom-aivdom-int*, *RETURN o length-ivdom-aivdom*) ∈ *aivdom-rel* →*f* ⟨*nat-rel*⟩*nres-rel*›
and
length-tvdom-aivdom-int:
‹(*RETURN o length-tvdom-aivdom-int*, *RETURN o length-tvdom-aivdom*) ∈ *aivdom-rel* →*f* ⟨*nat-rel*⟩*nres-rel*›
and

empty-aivdom-int:
‹(*RETURN o empty-aivdom-int*, *RETURN o empty-aivdom*) ∈ *aivdom-rel* →$_f$ ⟨*aivdom-rel*⟩*nres-rel*› **and**
empty-tvdom-int:
‹(*RETURN o empty-tvdom-int*, *RETURN o empty-tvdom*) ∈ *aivdom-rel* →$_f$ ⟨*aivdom-rel*⟩*nres-rel*› **and**
push-to-tvdom-int:
‹(*uncurry* (*RETURN oo push-to-tvdom-int*), *uncurry* (*RETURN oo push-to-tvdom*)) ∈ *nat-rel* ×$_f$ *aivdom-rel* →$_f$ ⟨*aivdom-rel*⟩*nres-rel*› **and**
AIvdom-init-int:
‹(*uncurry2* (*RETURN ooo AIvdom-init-int*), *uncurry2* (*RETURN ooo AIvdom-init*)) ∈ ⟨*nat-rel*⟩*list-rel* ×$_f$ ⟨*nat-rel*⟩*list-rel* ×$_f$ ⟨*nat-rel*⟩*list-rel* →$_f$ ⟨*aivdom-rel*⟩*nres-rel*› **and**
map-vdom-aivdom-int:
‹(*map-vdom-aivdom-int f*, *map-vdom-aivdom f*) ∈ *aivdom-rel* →$_f$ ⟨*aivdom-rel*⟩*nres-rel*› **and**
map-tvdom-aivdom-int:
‹(*map-tvdom-aivdom-int f*, *map-tvdom-aivdom f*) ∈ *aivdom-rel* →$_f$ ⟨*aivdom-rel*⟩*nres-rel*› **and**
swap-avdom-aivdom-int:
‹(*uncurry2* (*RETURN ooo swap-avdom-aivdom-int*), *uncurry2* (*RETURN ooo swap-avdom-aivdom*)) ∈ *aivdom-rel* ×$_f$ *nat-rel* ×$_f$ *nat-rel* →$_f$ ⟨*aivdom-rel*⟩*nres-rel*› **and**
take-avdom-aivdom-int:
‹(*uncurry* (*RETURN oo take-avdom-aivdom-int*), *uncurry* (*RETURN oo take-avdom-aivdom*)) ∈ *nat-rel* ×$_f$ *aivdom-rel* →$_f$ ⟨*aivdom-rel*⟩*nres-rel*›
apply (*auto intro*!: *frefI nres-relI simp*: *code-hider-rel-def add-learned-clause-aivdom-def remove-inactive-aivdom-def avdom-aivdom-at-alt-def*
ivdom-aivdom-at-alt-def vdom-aivdom-at-alt-def length-vdom-aivdom-alt-def length-avdom-aivdom-alt-def length-ivdom-aivdom-alt-def length-tvdom-aivdom-alt-def tvdom-aivdom-at-alt-def add-learned-clause-aivdom-int-def
IsaSAT-VDom.add-learned-clause-aivdom-strong-int-def add-learned-clause-aivdom-strong-def add-learned-clause-aivdo
IsaSAT-VDom.add-init-clause-aivdom-strong-int-def add-init-clause-aivdom-strong-def add-init-clause-aivdom-strong-i
add-init-clause-aivdom-def add-init-clause-aivdom-int-def
IsaSAT-VDom.add-learned-clause-aivdom-int-def
IsaSAT-VDom.remove-inactive-aivdom-int-def remove-inactive-aivdom-int-def
IsaSAT-VDom.remove-inactive-aivdom-tvdom-int-def remove-inactive-aivdom-tvdom-int-def
remove-inactive-aivdom-tvdom-def
IsaSAT-VDom.avdom-aivdom-at-int-def avdom-aivdom-at-int-def
IsaSAT-VDom.tvdom-aivdom-at-int-def tvdom-aivdom-at-int-def
IsaSAT-VDom.ivdom-aivdom-at-int-def ivdom-aivdom-at-int-def
IsaSAT-VDom.length-avdom-aivdom-int-def length-avdom-aivdom-int-def
IsaSAT-VDom.length-ivdom-aivdom-int-def length-ivdom-aivdom-int-def
IsaSAT-VDom.length-tvdom-aivdom-int-def length-tvdom-aivdom-int-def
IsaSAT-VDom-LLVM.empty-aivdom-int-def empty-aivdom-def IsaSAT-VDom.empty-aivdom-int-def
map-vdom-aivdom-def map-vdom-aivdom-int-def
AIvdom-init-int-def AIvdom-init-def map-tvdom-aivdom-int-def map-tvdom-aivdom-def
push-to-tvdom-int-def push-to-tvdom-def IsaSAT-VDom.push-to-tvdom-int-def
swap-avdom-aivdom-alt-def empty-tvdom-def empty-tvdom-int-def)
apply (*case-tac y*; *case-tac f ab*; *auto simp*: *swap-avdom-aivdom-int-def take-avdom-aivdom-int-def RES-RETURN-RES conc-fun-RES*)[]
apply (*case-tac y*; *case-tac f ba*; *auto simp*: *swap-avdom-aivdom-int-def take-avdom-aivdom-int-def RES-RETURN-RES conc-fun-RES*)[]
apply (*case-tac ab*; *auto simp*: *swap-avdom-aivdom-int-def take-avdom-aivdom-int-def*)
apply (*case-tac ba*; *auto simp*: *swap-avdom-aivdom-int-def take-avdom-aivdom-int-def*)
done

sepref-def *add-learned-clause-aivdom-impl*
is ‹*uncurry* (*RETURN oo add-learned-clause-aivdom-int*)›
:: ‹[λ(*C*,(*a*,*b*,*c*)). *Suc* (*length* (*a*)) < *max-snat 64*]$_a$ *sint64-nat-assn*^{k} ∗$_a$ *aivdom-int-assn*^{d} → *aivdom-int-assn*›
unfolding *add-learned-clause-aivdom-int-def*

by *sepref*

sepref-def *add-learned-clause-aivdom-strong-impl*
is ‹*uncurry* (*RETURN oo add-learned-clause-aivdom-strong-int*)›
:: ‹[λ(*C*,(*a*,*b*,*c*)). *Suc* (*length* (*a*)) < *max-snat 64* ∧ *Suc* (*length* (*c*)) < *max-snat 64*]$_a$ *sint64-nat-assn*^{k} *$_a$ *aivdom-int-assn*^{d} → *aivdom-int-assn*›
unfolding *add-learned-clause-aivdom-strong-int-def*
by *sepref*

sepref-def *add-init-clause-aivdom-strong-impl*
is ‹*uncurry* (*RETURN oo add-init-clause-aivdom-strong-int*)›
:: ‹[λ(*C*,(*a*,*b*,*c*)). *Suc* (*length* (*b*)) < *max-snat 64* ∧ *Suc* (*length* (*c*)) < *max-snat 64*]$_a$ *sint64-nat-assn*^{k} *$_a$ *aivdom-int-assn*^{d} → *aivdom-int-assn*›
unfolding *add-init-clause-aivdom-strong-int-def*
by *sepref*

sepref-def *remove-inactive-aivdom-impl*
is ‹*uncurry* (*RETURN oo remove-inactive-aivdom-int*)›
:: ‹[λ(*C*,(*a*,*b*,*c*)). *C* < (*length a*)]$_a$ *sint64-nat-assn*^{k} *$_a$ *aivdom-int-assn*^{d} → *aivdom-int-assn*›
unfolding *remove-inactive-aivdom-int-def*
by *sepref*

sepref-def *remove-inactive-aivdom-tvdom-impl*
is ‹*uncurry* (*RETURN oo remove-inactive-aivdom-tvdom-int*)›
:: ‹[λ(*C*,(*a*,*b*,*c*)). *C* < (*length c*)]$_a$ *sint64-nat-assn*^{k} *$_a$ *aivdom-int-assn*^{d} → *aivdom-int-assn*›
unfolding *remove-inactive-aivdom-tvdom-int-def*
by *sepref*

sepref-def *ivdom-aivdom-at-impl*
is ‹*uncurry* (*RETURN oo ivdom-aivdom-at-int*)›
:: ‹[λ((*b*,*c*,*d*), *C*). *C* < (*length c*)]$_a$ *aivdom-int-assn*^{k} *$_a$ *sint64-nat-assn*^{k} → *sint64-nat-assn*›
unfolding *ivdom-aivdom-at-int-def*
by *sepref*

sepref-def *avdom-aivdom-at-impl*
is ‹*uncurry* (*RETURN oo avdom-aivdom-at-int*)›
:: ‹[λ((*b*,*c*), *C*). *C* < (*length b*)]$_a$ *aivdom-int-assn*^{k} *$_a$ *sint64-nat-assn*^{k} → *sint64-nat-assn*›
unfolding *avdom-aivdom-at-int-def*
by *sepref*

sepref-def *tvdom-aivdom-at-impl*
is ‹*uncurry* (*RETURN oo tvdom-aivdom-at-int*)›
:: ‹[λ((*b*,*c*,*d*), *C*). *C* < (*length d*)]$_a$ *aivdom-int-assn*^{k} *$_a$ *sint64-nat-assn*^{k} → *sint64-nat-assn*›
unfolding *tvdom-aivdom-at-int-def*
by *sepref*

sepref-def *length-avdom-aivdom-impl*
is ‹*RETURN o length-avdom-aivdom-int*›
:: ‹*aivdom-int-assn*^{k} →$_a$ *sint64-nat-assn*›
unfolding *length-avdom-aivdom-int-def*
by *sepref*

definition *workaround-RF* **where**
‹*workaround-RF xs* = *length xs*›

sepref-def *workaround-RF-code* [*llvm-inline*]
  is ‹*RETURN o workaround-RF*›
  :: ‹*vdom-fast-assn*^{k} →$_a$ *sint64-nat-assn*›
  unfolding *workaround-RF-def*
  by *sepref*

sepref-def *length-ivdom-aivdom-impl*
  is ‹*RETURN o length-ivdom-aivdom-int*›
  :: ‹*aivdom-int-assn*^{k} →$_a$ *sint64-nat-assn*›
  unfolding *length-ivdom-aivdom-int-def comp-def workaround-RF-def*[*symmetric*]
  by *sepref*

sepref-def *length-tvdom-aivdom-impl*
  is ‹*RETURN o length-tvdom-aivdom-int*›
  :: ‹*aivdom-int-assn*^{k} →$_a$ *sint64-nat-assn*›
  unfolding *length-tvdom-aivdom-int-def comp-def workaround-RF-def*[*symmetric*]
  by *sepref*

sepref-def *swap-avdom-aivdom-impl*
  is ‹*uncurry2* (*RETURN ooo swap-avdom-aivdom-int*)›
 :: ‹[λ(((*b*,*c*,*d*),*i*), *j*). *i* < *length b* ∧ *j* < *length b*]$_a$ *aivdom-int-assn*^{d} *$_a$ *sint64-nat-assn*^{k} *$_a$ *sint64-nat-assn*^{k}
→ *aivdom-int-assn*›
  unfolding *swap-avdom-aivdom-int-def convert-swap gen-swap*
  by *sepref*

sepref-def *take-avdom-aivdom-impl*
  is ‹*uncurry* (*RETURN oo take-avdom-aivdom-int*)›
  :: ‹[λ(*i*, (*b*,*c*,*d*)). *i* ≤ *length b*]$_a$ *sint64-nat-assn*^{k} *$_a$ *aivdom-int-assn*^{d} → *aivdom-int-assn*›
  unfolding *take-avdom-aivdom-int-def*
  by *sepref*

sepref-def *empty-aivdom-impl*
  is ‹*RETURN o empty-aivdom-int*›
  :: ‹*aivdom-int-assn*^{d} →$_a$ *aivdom-int-assn*›
  unfolding *empty-aivdom-int-def*
  apply (*annot-snat-const* ‹*TYPE*(*64*)›)
  by *sepref*

sepref-def *AIvdom-init-impl*
  is ‹*uncurry2* (*RETURN ooo AIvdom-init-int*)›
  :: ‹*vdom-fast-assn*^{d} *$_a$ *vdom-fast-assn*^{d} *$_a$ *vdom-fast-assn*^{d} →$_a$ *aivdom-int-assn*›
  unfolding *AIvdom-init-int-def*
  by *sepref*

sepref-def *empty-tvdom-impl*
  is ‹*RETURN o empty-tvdom-int*›
  :: ‹*aivdom-int-assn*^{d} →$_a$ *aivdom-int-assn*›
  unfolding *empty-tvdom-int-def*
  apply (*annot-snat-const* ‹*TYPE*(*64*)›)
  by *sepref*

sepref-def *push-tvdom-impl*
  is ‹*uncurry* (*RETURN oo push-to-tvdom-int*)›
  :: ‹[λ(*C*,(-,-,*tv*)). *Suc* (*length tv*) < *max-snat 64*]$_a$
  sint64-nat-assn^k *$_a$ *aivdom-int-assn*^{d} → *aivdom-int-assn*›

unfolding *push-to-tvdom-int-def*
by *sepref*

lemma *aivdom-assn-alt-def*:
‹*aivdom-assn* = *hr-comp aivdom-int-assn* (⟨*aivdom-int-rel*⟩*code-hider-rel*)›
unfolding *aivdom-assn-def code-hider-assn-def* **by** *auto*

context
notes [*fcomp-norm-unfold*] = *aivdom-assn-alt-def*[*symmetric*] *aivdom-assn-def*[*symmetric*]
begin

theorem [*sepref-fr-rules*]:
‹(*uncurry add-learned-clause-aivdom-impl*, *uncurry* (*RETURN* ∘∘ *add-learned-clause-aivdom*))
∈ [λ(*C*,*ai*). *Suc* (*length* (*get-avdom-aivdom ai*)) < *max-snat 64*]$_a$ *snat-assn*^{k} *$*$*$_a$ *aivdom-assn*^{d} → *aivdom-assn*›
(**is** ‹*?c* ∈ [*?pre*]$_a$ *?im* → *?f*›)
proof −
have *H*: ‹*?c*
∈ [*comp-PRE* (*nat-rel* ×$_f$ *aivdom-rel*) (λ-. *True*)
(λ*x y*. *case y of* (*C*, *a*, *b*, *c*) ⇒ *Suc* (*length a*) < *max-snat 64*)
(λ*x*. *nofail* (*uncurry* (*RETURN* ∘∘ *add-learned-clause-aivdom*) *x*))]$_a$ *?im* → *?f*›
(**is** ‹- ∈ [*?pre'*]$_a$ - → -›)
using *hfref-compI-PRE*[*OF add-learned-clause-aivdom-impl.refine*
add-learned-clause-aivdom-int, *unfolded fcomp-norm-unfold aivdom-assn-alt-def*[*symmetric*]] **by** *blast*
have *pre*: ‹*?pre'* = *?pre*› **for** *x h*
by (*intro ext*, *rename-tac x*, *case-tac x*, *case-tac* ‹*snd x*›)
(*auto simp*: *comp-PRE-def code-hider-rel-def*)
show *?thesis*
using *H*
unfolding *pre*
by *blast*
qed

theorem [*sepref-fr-rules*]:
‹(*uncurry add-learned-clause-aivdom-strong-impl*, *uncurry* (*RETURN* ∘∘ *add-learned-clause-aivdom-strong*))
∈ [λ(*C*,*ai*). *Suc* (*length* (*get-avdom-aivdom ai*)) < *max-snat 64* ∧ *Suc* (*length* (*get-tvdom-aivdom ai*))
< *max-snat 64*]$_a$ *snat-assn*^{k} *$*$*$_a$ *aivdom-assn*^{d} → *aivdom-assn*›
(**is** ‹*?c* ∈ [*?pre*]$_a$ *?im* → *?f*›)
proof −
have *H*: ‹*?c*
∈ [*comp-PRE* (*nat-rel* ×$_f$ *aivdom-rel*) (λ-. *True*)
(λ*x y*. *case y of* (*C*, *a*, *b*, *c*) ⇒ *Suc* (*length a*) < *max-snat 64* ∧ *Suc* (*length c*) < *max-snat 64*)
(λ*x*. *nofail* (*uncurry* (*RETURN* ∘∘ *add-learned-clause-aivdom-strong*) *x*))]$_a$ *?im* → *?f*›
(**is** ‹- ∈ [*?pre'*]$_a$ - → -›)
using *hfref-compI-PRE*[*OF add-learned-clause-aivdom-strong-impl.refine*
add-learned-clause-aivdom-strong-int, *unfolded fcomp-norm-unfold aivdom-assn-alt-def*[*symmetric*]]
by *blast*
have *pre*: ‹*?pre'* = *?pre*› **for** *x h*
by (*intro ext*, *rename-tac x*, *case-tac x*, *case-tac* ‹*snd x*›)
(*auto simp*: *comp-PRE-def code-hider-rel-def*)
show *?thesis*
using *H*
unfolding *pre*
by *blast*
qed

theorem [*sepref-fr-rules*]:
‹(*uncurry add-init-clause-aivdom-strong-impl*, *uncurry* (*RETURN* ∘∘ *add-init-clause-aivdom-strong*))
∈ [λ(*C*,*ai*). *Suc* (*length* (*get-ivdom-aivdom ai*)) < *max-snat 64* ∧ *Suc* (*length* (*get-tvdom-aivdom ai*)) < *max-snat 64*]$_a$ *snat-assn*^{k} *$*_a$ *aivdom-assn*^{d} → *aivdom-assn*›
(**is** ‹*?c* ∈ [*?pre*]$_a$ *?im* → *?f*›)
proof −
have *H*: ‹*?c*
∈ [*comp-PRE* (*nat-rel* ×$_f$ *aivdom-rel*) (λ-. *True*)
(λ*x y*. *case y of* (*C*, *a*, *b*, *c*) ⇒ *Suc* (*length b*) < *max-snat 64* ∧ *Suc* (*length c*) < *max-snat 64*)
(λ*x*. *nofail* (*uncurry* (*RETURN* ∘∘ *add-init-clause-aivdom-strong*) *x*))]$_a$ *?im* → *?f*›
(**is** ‹- ∈ [*?pre'*]$_a$ - → -›)
using *hfref-compI-PRE*[*OF add-init-clause-aivdom-strong-impl.refine*
add-init-clause-aivdom-strong-int, *unfolded fcomp-norm-unfold aivdom-assn-alt-def*[*symmetric*]] **by** *blast*
have *pre*: ‹*?pre'* = *?pre*› **for** *x h*
by (*intro ext*, *rename-tac x*, *case-tac x*, *case-tac* ‹*snd x*›)
(*auto simp*: *comp-PRE-def code-hider-rel-def*)
show *?thesis*
using *H*
unfolding *pre*
by *blast*
qed

theorem [*sepref-fr-rules*]:
‹(*uncurry remove-inactive-aivdom-impl*, *uncurry* (*RETURN* ∘∘ *remove-inactive-aivdom*))
∈ [λ(*C*,*ai*). *C* < (*length* (*get-avdom-aivdom ai*))]$_a$ *snat-assn*^{k} *$*_a$ *aivdom-assn*^{d} → *aivdom-assn*›
(**is** ‹*?c* ∈ [*?pre*]$_a$ *?im* → *?f*›)
proof −
have *H*: ‹*?c*
∈ [*comp-PRE*
(*nat-rel* ×$_f$ *aivdom-rel*)
(λ-. *True*) (λ*x y*. *case y of* (*C*, *a*, *b*, *c*) ⇒ *C* < *length a*)
(λ*x*. *nofail* (*uncurry* (*RETURN* ∘∘ *remove-inactive-aivdom*) *x*))]$_a$ *?im* → *?f*›
(**is** ‹- ∈ [*?pre'*]$_a$ - → -›)
using *hfref-compI-PRE*[*OF remove-inactive-aivdom-impl.refine*
remove-inactive-aivdom-int, *unfolded fcomp-norm-unfold aivdom-assn-alt-def*[*symmetric*]] **by** *blast*
have *pre*: ‹*?pre'* = *?pre*› **for** *x h*
by (*intro ext*, *rename-tac x*, *case-tac x*, *case-tac* ‹*snd x*›)
(*auto simp*: *comp-PRE-def code-hider-rel-def*)
show *?thesis*
using *H*
unfolding *pre*
by *blast*
qed

theorem [*sepref-fr-rules*]:
‹(*uncurry remove-inactive-aivdom-tvdom-impl*, *uncurry* (*RETURN* ∘∘ *remove-inactive-aivdom-tvdom*))
∈ [λ(*C*,*ai*). *C* < (*length* (*get-tvdom-aivdom ai*))]$_a$ *snat-assn*^{k} *$*_a$ *aivdom-assn*^{d} → *aivdom-assn*›
(**is** ‹*?c* ∈ [*?pre*]$_a$ *?im* → *?f*›)
proof −
have *H*: ‹*?c*
∈ [*comp-PRE*
(*nat-rel* ×$_f$ *aivdom-rel*)
(λ-. *True*) (λ*x y*. *case y of* (*C*, *a*, *b*, *c*) ⇒ *C* < *length c*)
(λ*x*. *nofail* (*uncurry* (*RETURN* ∘∘ *remove-inactive-aivdom-tvdom*) *x*))]$_a$ *?im* → *?f*›

```
    (is ‹- ∈ [?pre']ₐ - → -›)
    using hfref-compI-PRE[OF remove-inactive-aivdom-tvdom-impl.refine
      remove-inactive-aivdom-tvdom-int, unfolded fcomp-norm-unfold aivdom-assn-alt-def[symmetric]]
    by blast
  have pre: ‹?pre' = ?pre› for x h
    by (intro ext, rename-tac x, case-tac x, case-tac ‹snd x›)
      (auto simp: comp-PRE-def code-hider-rel-def)
  show ?thesis
    using H
    unfolding pre
    by blast
qed


theorem avdom-aivdom-at-impl-refine[sepref-fr-rules]:
  ‹(uncurry avdom-aivdom-at-impl, uncurry (RETURN ∘∘ avdom-aivdom-at))
∈ [λ(ai, C). C < (length (get-avdom-aivdom ai)) ]ₐ aivdom-assn^k *ₐ sint64-nat-assn^k → sint64-nat-assn›
  (is ‹?c ∈ [?pre]ₐ ?im → ?f›)
proof -
  have H: ‹?c
∈ [comp-PRE
    (aivdom-rel ×_f nat-rel)
    (λ-. True) (λx y. case y of (x, xa) ⇒ (case x of (b, c) ⇒ λC. C < length b) xa)
    (λx. nofail
    (uncurry (RETURN ∘∘ avdom-aivdom-at) x))]ₐ ?im → ?f›
    (is ‹- ∈ [?pre']ₐ - → -›)
    using hfref-compI-PRE[OF avdom-aivdom-at-impl.refine
      avdom-aivdom-at-int, unfolded fcomp-norm-unfold aivdom-assn-alt-def[symmetric]] by simp
  have pre: ‹?pre' = ?pre› for x h
    by (intro ext, rename-tac x, case-tac x, case-tac ‹fst x›)
      (auto simp: comp-PRE-def code-hider-rel-def)
  show ?thesis
    using H
    unfolding pre
    by blast
qed


theorem ivdom-aivdom-at-impl-refine[sepref-fr-rules]:
  ‹(uncurry ivdom-aivdom-at-impl, uncurry (RETURN ∘∘ ivdom-aivdom-at))
∈ [λ(ai, C). C < (length (get-ivdom-aivdom ai)) ]ₐ aivdom-assn^k *ₐ sint64-nat-assn^k → sint64-nat-assn›
  (is ‹?c ∈ [?pre]ₐ ?im → ?f›)
proof -
  have H: ‹?c
∈ [comp-PRE
    (⟨aivdom-int-rel⟩code-hider-rel ×_f nat-rel)
    (λ-. True) (λx y. case y of (x, xa) ⇒ (case x of (b, c, d) ⇒ λC. C < length c) xa)
    (λx. nofail
    (uncurry (RETURN ∘∘ ivdom-aivdom-at) x))]ₐ ?im → ?f›
    (is ‹- ∈ [?pre']ₐ - → -›)
    using hfref-compI-PRE[OF ivdom-aivdom-at-impl.refine
      ivdom-aivdom-at-int, unfolded fcomp-norm-unfold aivdom-assn-alt-def[symmetric]] by simp
  have pre: ‹?pre' = ?pre› for x h
    by (intro ext, rename-tac x, case-tac x, case-tac ‹fst x›)
      (auto simp: comp-PRE-def code-hider-rel-def)
  show ?thesis
```

using *H*
unfolding *pre*
by *blast*
qed

theorem *tvdom-aivdom-at-impl-refine*[*sepref-fr-rules*]:
‹(*uncurry tvdom-aivdom-at-impl*, *uncurry* (*RETURN* ∘∘ *tvdom-aivdom-at*))
∈ [λ(*ai*, *C*). *C* < (*length* (*get-tvdom-aivdom ai*))]$_a$ *aivdom-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} → *sint64-nat-assn*›
(**is** ‹*?c* ∈ [*?pre*]$_a$ *?im* → *?f*›)
proof −
have *H*: ‹*?c*
∈ [*comp-PRE*
(*aivdom-rel* ×$_f$ *nat-rel*)
(λ-. *True*) (λ*x y*. *case y of* (*x*, *xa*) ⇒ (*case x of* (*b*, *c*, *d*) ⇒ λ*C*. *C* < *length d*) *xa*)
(λ*x*. *nofail*
(*uncurry* (*RETURN* ∘∘ *tvdom-aivdom-at*) *x*))]$_a$ *?im* → *?f*›
(**is** ‹- ∈ [*?pre*']$_a$ - → -›)
using *hfref-compI-PRE*[*OF tvdom-aivdom-at-impl.refine*
tvdom-aivdom-at-int, *unfolded fcomp-norm-unfold aivdom-assn-alt-def*[*symmetric*]] **by** *blast*
have *pre*: ‹*?pre'* = *?pre*› **for** *x h*
by (*intro ext*, *rename-tac x*, *case-tac x*, *case-tac* ‹*fst x*›)
(*auto simp*: *comp-PRE-def code-hider-rel-def*)
show *?thesis*
using *H*
unfolding *pre*
by *blast*
qed

theorem [*sepref-fr-rules*]:
‹(*uncurry take-avdom-aivdom-impl*, *uncurry* (*RETURN* ∘∘ *take-avdom-aivdom*))
∈ [λ(*C*, *ai*). *C* ≤ *length* (*get-avdom-aivdom ai*)]$_a$ *sint64-nat-assn*^{k} *$*_a$ *aivdom-assn*^{d} → *aivdom-assn*›
(**is** ‹*?c* ∈ [*?pre*]$_a$ *?im* → *?f*›)
proof −
have *H*: ‹*?c*
∈ [*comp-PRE* (*nat-rel* ×$_f$ *aivdom-rel*) (λ-. *True*)
(λ*x y*. *case y of* (*i*, *b*, *c*, *d*) ⇒ *i* ≤ *length b*)
(λ*x*. *nofail* (*uncurry* (*RETURN* ∘∘ *take-avdom-aivdom*) *x*))]$_a$ *?im* → *?f*›
(**is** ‹- ∈ [*?pre*']$_a$ - → -›)
using *hfref-compI-PRE*[*OF take-avdom-aivdom-impl.refine*
take-avdom-aivdom-int, *unfolded fcomp-norm-unfold aivdom-assn-alt-def*[*symmetric*]] **by** *blast*
have *pre*: ‹*?pre'* = *?pre*› **for** *x h*
by (*intro ext*, *rename-tac x*, *case-tac x*, *case-tac* ‹*snd x*›)
(*auto simp*: *comp-PRE-def code-hider-rel-def*)
show *?thesis*
using *H*
unfolding *pre*
by *blast*
qed

theorem *push-tvdom-impl-refine*[*sepref-fr-rules*]:
‹(*uncurry push-tvdom-impl*, *uncurry* (*RETURN* ∘∘ *push-to-tvdom*))
∈ [λ(*C*, *ai*). *Suc* (*length* (*get-tvdom-aivdom ai*)) < *max-snat 64*]$_a$ *sint64-nat-assn*^{k} *$*_a$ *aivdom-assn*^{d} →
aivdom-assn›

(**is** ‹?c ∈ [?pre]$_a$?im → ?f›)
proof −
have *H*: ‹?c
∈ [*comp-PRE* (*nat-rel* ×$_f$ *aivdom-rel*) (λ-. *True*)
(λ*x y*. *case y of* (*C*, *uu*-, *uua*-, *tv*) ⇒ *Suc* (*length tv*) < *max-snat 64*)
(λ*x*. *nofail* (*uncurry* (*RETURN* ∘∘ *push-to-tvdom*) *x*))]$_a$?im → ?f›
(**is** ‹- ∈ [?pre']$_a$ - → -›)
using *hfref-compI-PRE*[*OF push-tvdom-impl.refine*
push-to-tvdom-int, *unfolded fcomp-norm-unfold aivdom-assn-alt-def*[*symmetric*]] **by** *blast*
have *pre*: ‹?pre' = ?pre› **for** *x h*
by (*intro ext*, *rename-tac x*, *case-tac x*, *case-tac* ‹*snd x*›)
(*auto simp*: *comp-PRE-def code-hider-rel-def*)
show *?thesis*
using *H*
unfolding *pre*
by *blast*
qed

theorem *swap-avdom-aivdom-impl-refine*[*sepref-fr-rules*]:
‹(*uncurry2 swap-avdom-aivdom-impl*, *uncurry2* (*RETURN ooo swap-avdom-aivdom*))
∈ [λ((*ai*, *i*), *j*). *i* < *length* (*get-avdom-aivdom ai*) ∧ *j* < *length* (*get-avdom-aivdom ai*)]$_a$
aivdom-assn^d *$*_a$ *sint64-nat-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} → *aivdom-assn*›
(**is** ‹?c ∈ [?pre]$_a$?im → ?f›)
proof −
have *H*: ‹?c
∈ [*comp-PRE* (*aivdom-rel* ×$_f$ *nat-rel* ×$_f$ *nat-rel*) (λ-. *True*)
(λ*x y*. *case y of*
(*x*, *xa*) ⇒
(*case x of*
(*x*, *xa*) ⇒
(*case x of*
(*b*, *c*, *d*) ⇒ λ*i j*. *i* < *length b* ∧ *j* < *length b*)
xa)
xa)
(λ*x*. *nofail*
(*uncurry2* (*RETURN* ∘∘∘ *swap-avdom-aivdom*)
x))]$_a$?im → ?f›
(**is** ‹- ∈ [?pre']$_a$ - → -›)
using *hfref-compI-PRE*[*OF swap-avdom-aivdom-impl.refine*
swap-avdom-aivdom-int, *unfolded fcomp-norm-unfold aivdom-assn-alt-def*[*symmetric*]] **by** *blast*
have *pre*: ‹?pre' = ?pre› **for** *x h*
by (*intro ext*, *rename-tac x*, *case-tac x*, *case-tac* ‹*fst* (*fst x*)›)
(*auto simp*: *comp-PRE-def code-hider-rel-def*)
show *?thesis*
using *H*
unfolding *pre*
by *blast*
qed

lemma *aivdom-int-assn-alt-def*:
‹*aivdom-int-assn* = *hr-comp aivdom-int-assn*
(⟨*nat-rel*⟩*list-rel* ×$_f$ (⟨*nat-rel*⟩*list-rel* ×$_f$ ⟨*nat-rel*⟩*list-rel*))›
by *auto*

sepref-register *swap-avdom-aivdom take-avdom-aivdom add-init-clause-aivdom-strong add-learned-clause-aivdom-strong*
add-learned-clause-aivdom

lemma *vdom-fast-assn-alt-def*: ‹*vdom-fast-assn* = *hr-comp LBD-it.arr-assn* (⟨*nat-rel*⟩*list-rel*)›
by *auto*

lemmas *vdom-ref*[*sepref-fr-rules*] =
length-avdom-aivdom-impl.refine[*FCOMP length-avdom-aivdom-int*]
length-ivdom-aivdom-impl.refine[*FCOMP length-ivdom-aivdom-int*]
length-tvdom-aivdom-impl.refine[*FCOMP length-tvdom-aivdom-int*]
hn-id[*FCOMP Constructor-hnr*[*of aivdom-int-rel*], *of aivdom-int-assn*,
unfolded aivdom-assn-alt-def[*symmetric*] *aivdom-assn-def*[*symmetric*] *aivdom-int-assn-alt-def*[*symmetric*]]
hn-id[*FCOMP get-content-hnr*[*of aivdom-int-rel*], *of aivdom-int-assn*,
unfolded aivdom-assn-alt-def[*symmetric*] *aivdom-assn-def*[*symmetric*] *aivdom-int-assn-alt-def*[*symmetric*]]
empty-aivdom-impl.refine[*FCOMP empty-aivdom-int*]
AIvdom-init-impl.refine[*FCOMP AIvdom-init-int*, *unfolded vdom-fast-assn-alt-def*[*symmetric*]]
empty-tvdom-impl.refine[*FCOMP empty-tvdom-int*]
end

end
theory *Tuple4-LLVM*
imports *Tuple4 IsaSAT-Literals-LLVM*
begin

hide-const (**open**) *NEMonad.ASSERT NEMonad.RETURN*

This is the setup for accessing and modifying the state as an abstract tuple of 15 elements. The construction is kept generic (even if still targetting only our state). There is a lot of copy-paste that would be nice to automate at some point.

We define 3 sort of operations:

1. extracting an element, replacing it by an default element. Modifies the state. The name starts with *exctr*
2. reinserting an element, freeing the current one. Modifies the state. The name starts with *update*
3. in-place reading a value, possibly with pure parameters. Does not modify the state. The name starts with *read*

instantiation *tuple4* ::
(*llvm-rep*,*llvm-rep*,*llvm-rep*,*llvm-rep*) *llvm-rep*
begin
definition *to-val-tuple4* **where**
‹*to-val-tuple4* ≡ (λ*S*. *case S of*
Tuple4 M N D i ⇒ *LL-STRUCT* [*to-val M*, *to-val N*, *to-val D*, *to-val i*])›

definition *from-val-tuple4* :: ‹*llvm-val* ⇒ (′*a*, ′*b*, ′*c*, ′*d*) *tuple4*› **where**
‹*from-val-tuple4* ≡ (λ*p*. *case llvm-val.the-fields p of*
[*M*, *N*, *D*, *i*] ⇒
Tuple4 (*from-val M*) (*from-val N*) (*from-val D*) (*from-val i*))›

definition [*simp*]: *struct-of-tuple4* (- :: (′*a*, ′*b*, ′*c*, ′*d*) *tuple4 itself*) ≡
VS-STRUCT [*struct-of TYPE*(′*a*), *struct-of TYPE*(′*b*), *struct-of TYPE*(′*c*),
struct-of TYPE(′*d*)]

definition [*simp*]: *init-tuple4* :: (′*a*, ′*b*, ′*c*, ′*d*) *tuple4* ≡ *Tuple4 init init init init*

```
  instance
    apply standard
   unfolding from-val-tuple4-def to-val-tuple4-def struct-of-tuple4-def init-tuple4-def comp-def tuple4.case-distrib
    subgoal
      by (auto simp: init-zero fun-eq-iff from-val-tuple4-def split: tuple4.splits)
    subgoal for v
      by (cases v) (auto split: list.splits tuple4.splits)
    subgoal for v
      by (cases v)
       (simp add: LLVM-Shallow.null-def to-val-ptr-def split: tuple4.splits)
    subgoal
      by (simp add: LLVM-Shallow.null-def to-val-ptr-def to-val-word-def init-zero split: tuple4.splits)
    done
end
```

Setup for LLVM code export

Declare structure to code generator.

```
lemma to-val-tuple17[ll-struct-of]: struct-of TYPE(('a, 'b, 'c, 'd) tuple4) = VS-STRUCT [
  struct-of TYPE('a::llvm-rep),
  struct-of TYPE('b::llvm-rep),
  struct-of TYPE('c::llvm-rep),
  struct-of TYPE('d::llvm-rep)]
  by (auto)

lemma node-insert-value:
  ll-insert-value (Tuple4 M N D i) M' 0 = Mreturn (Tuple4 M' N D i)
  ll-insert-value (Tuple4 M N D i) N' (Suc 0) = Mreturn (Tuple4 M N' D i)
  ll-insert-value (Tuple4 M N D i) D' 2 = Mreturn (Tuple4 M N D' i)
  ll-insert-value (Tuple4 M N D i) i' 3 = Mreturn (Tuple4 M N D i')
  by (simp-all add: ll-insert-value-def llvm-insert-value-def Let-def checked-from-val-def
            to-val-tuple4-def from-val-tuple4-def)

lemma node-extract-value:
  ll-extract-value (Tuple4 M N D i) 0 = Mreturn M
  ll-extract-value (Tuple4 M N D i) (Suc 0) = Mreturn N
  ll-extract-value (Tuple4 M N D i) 2 = Mreturn D
  ll-extract-value (Tuple4 M N D i) 3 = Mreturn i
  apply (simp-all add: ll-extract-value-def llvm-extract-value-def Let-def checked-from-val-def
            to-val-tuple4-def from-val-tuple4-def)
  done
```

Lemmas to translate node construction and destruction

```
lemma inline-return-node[llvm-pre-simp]: Mreturn (Tuple4 M N D i) = doM {
    r ← ll-insert-value init M 0;
    r ← ll-insert-value r N 1;
    r ← ll-insert-value r D 2;
    r ← ll-insert-value r i 3;
    Mreturn r
  }
  apply (auto simp: node-insert-value)
  done

lemma inline-node-case[llvm-pre-simp]: (case r of (Tuple4 M N D i) ⇒ f M N D i) = doM {
```

```
    M ← ll-extract-value r 0;
    N ← ll-extract-value r 1;
    D ← ll-extract-value r 2;
    i ← ll-extract-value r 3;
  f M N D i
}
  apply (cases r)
  apply (auto simp: node-extract-value)
  done

lemma inline-return-node-case[llvm-pre-simp]: doM {Mreturn (case r of (Tuple4 M N D i) ⇒ f M N
D i)} = doM {
    M ← ll-extract-value r 0;
    N ← ll-extract-value r 1;
    D ← ll-extract-value r 2;
    i ← ll-extract-value r 3;
  Mreturn (f M N D i)
}
  apply (cases r)
  apply (auto simp: node-extract-value)
  done
lemma inline-direct-return-node-case[llvm-pre-simp]: doM {(case r of (Tuple4 M N D i) ⇒ f M N D
i)} = doM {
    M ← ll-extract-value r 0;
    N ← ll-extract-value r 1;
    D ← ll-extract-value r 2;
    i ← ll-extract-value r 3;
   (f M N D i)
}
  apply (cases r)
  apply (auto simp: node-extract-value)
  done

lemmas [llvm-inline] =
  tuple4.Tuple4-a-def
  tuple4.Tuple4-b-def
  tuple4.Tuple4-c-def
  tuple4.Tuple4-d-def

fun tuple4-assn :: ‹
  ('a ⇒ - ⇒ llvm-amemory ⇒ bool) ⇒
  ('b ⇒ - ⇒ llvm-amemory ⇒ bool) ⇒
  ('c ⇒ - ⇒ llvm-amemory ⇒ bool) ⇒
  ('d ⇒ - ⇒ llvm-amemory ⇒ bool) ⇒
  ('a, 'b, 'c, 'd) tuple4 ⇒ - ⇒ assn› where
  ‹tuple4-assn a-assn b-assn' c-assn d-assn S T =
   (case (S, T) of
  (Tuple4 M N D i,
   Tuple4 M' N' D' i')
     ⇒
 (a-assn M (M') ∧* b-assn' N (N') ∧* c-assn D (D') ∧* d-assn i (i')))
  ›

locale tuple4-state-ops =
  fixes
    a-assn :: ‹'a ⇒ 'xa ⇒ assn› and
```

b-assn :: ‹*'b* ⇒ *'xb* ⇒ *assn*› **and**
c-assn :: ‹*'c* ⇒ *'xc* ⇒ *assn*› **and**
d-assn :: ‹*'d* ⇒ *'xd* ⇒ *assn*› **and**
a-default :: *'a* **and**
a :: ‹*'xa llM*› **and**
b-default :: *'b* **and**
b :: ‹*'xb llM*› **and**
c-default :: *'c* **and**
c :: ‹*'xc llM*› **and**
d-default :: *'d* **and**
d :: ‹*'xd llM*›
begin

definition *tuple4-int-assn* :: ‹- ⇒ - ⇒ *assn*› **where**
‹*tuple4-int-assn* = *tuple4-assn*
a-assn b-assn c-assn d-assn›

definition *remove-a* :: ‹(*'a*, *'b*, *'c*, *'d*) *tuple4* ⇒ *'a* × (*'a*, *'b*, *'c*, *'d*) *tuple4*› **where**
‹*remove-a tuple4* = (*case tuple4 of Tuple4 x1 x2 x3 x4* ⇒
(*x1*, *Tuple4 a-default x2 x3 x4*))›

definition *remove-b* :: ‹(*'a*, *'b*, *'c*, *'d*) *tuple4* ⇒ *'b* × (*'a*, *'b*, *'c*, *'d*) *tuple4*› **where**
‹*remove-b tuple4* = (*case tuple4 of Tuple4 x1 x2 x3 x4* ⇒
(*x2*, *Tuple4 x1 b-default x3 x4*))›

definition *remove-c* :: ‹(*'a*, *'b*, *'c*, *'d*) *tuple4* ⇒ - × (*'a*, *'b*, *'c*, *'d*) *tuple4*› **where**
‹*remove-c tuple4* = (*case tuple4 of Tuple4 x1 x2 x3 x4* ⇒
(*x3*, *Tuple4 x1 x2 c-default x4*))›

definition *remove-d* :: ‹(*'a*, *'b*, *'c*, *'d*) *tuple4* ⇒ - × (*'a*, *'b*, *'c*, *'d*) *tuple4*› **where**
‹*remove-d tuple4* = (*case tuple4 of Tuple4 x1 x2 x3 x4* ⇒
(*x4*, *Tuple4 x1 x2 x3 d-default*))›

definition *update-a* :: ‹*'a* ⇒ (*'a*, *'b*, *'c*, *'d*) *tuple4* ⇒ (*'a*, *'b*, *'c*, *'d*) *tuple4*› **where**
‹*update-a x1 tuple4* = (*case tuple4 of Tuple4 M x2 x3 x4* ⇒
let - = M in
Tuple4 x1 x2 x3 x4)›

definition *update-b* :: ‹*'b* ⇒ (*'a*, *'b*, *'c*, *'d*) *tuple4* ⇒ (*'a*, *'b*, *'c*, *'d*) *tuple4*› **where**
‹*update-b x2 tuple4* = (*case tuple4 of Tuple4 x1 M x3 x4* ⇒
let - = M in
Tuple4 x1 x2 x3 x4)›

definition *update-c* :: ‹*'c* ⇒ (*'a*, *'b*, *'c*, *'d*) *tuple4* ⇒ (*'a*, *'b*, *'c*, *'d*) *tuple4*› **where**
‹*update-c x3 tuple4* = (*case tuple4 of Tuple4 x1 x2 M x4* ⇒
let - = M in
Tuple4 x1 x2 x3 x4)›

definition *update-d* :: ‹*'d* ⇒ (*'a*, *'b*, *'c*, *'d*) *tuple4* ⇒ (*'a*, *'b*, *'c*, *'d*) *tuple4*› **where**
‹*update-d x4 tuple4* = (*case tuple4 of Tuple4 x1 x2 x3 M* ⇒
let - = M in
Tuple4 x1 x2 x3 x4)›

end

lemma *tuple4-assn-conv*[*simp*]:

tuple4-assn P1 P2 P3 P4 (Tuple4 a1 a2 a3 a4)
(Tuple4 a1' a2' a3' a4') =
*(P1 a1 a1' ∧**
*P2 a2 a2' ∧**
*P3 a3 a3' ∧**
P4 a4 a4')
unfolding *tuple4-assn.simps* **by** *auto*

lemma *tuple4-assn-ctxt*:
‹*tuple4-assn P1 P2 P3 P4 (Tuple4 a1 a2 a3 a4)*
(Tuple4 a1' a2' a3' a4') = z ⟹
hn-ctxt (tuple4-assn P1 P2 P3 P4) (Tuple4 a1 a2 a3 a4)
(Tuple4 a1' a2' a3' a4')= z›
by (*simp add*: *hn-ctxt-def*)

lemma *hn-case-tuple4'*[*sepref-comb-rules*]:
assumes *FR*: ‹Γ ⊢ *hn-ctxt (tuple4-assn P1 P2 P3 P4) p' p ** Γ1*›
assumes *Pair*: ⋀*a1 a2 a3 a4 a1' a2' a3' a4'*.
⟦*p'=Tuple4 a1' a2' a3' a4'*⟧
⟹ *hn-refine (hn-ctxt P1 a1' a1 ∧* hn-ctxt P2 a2' a2 ∧* hn-ctxt P3 a3' a3 ∧* hn-ctxt P4 a4' a4 ∧* Γ1)*
(f a1 a2 a3 a4)
(Γ2 a1 a2 a3 a4 a1' a2' a3' a4') R
(CP a1 a2 a3 a4)
(f' a1' a2' a3' a4')
assumes *FR2*: ‹⋀*a1 a2 a3 a4 a1' a2' a3' a4'*.
Γ2 a1 a2 a3 a4 a1' a2' a3' a4' ⊢
*hn-ctxt P1' a1' a1 ** hn-ctxt P2' a2' a2 ** hn-ctxt P3' a3' a3 ** hn-ctxt P4' a4' a4 ** Γ1'*›
shows ‹*hn-refine Γ (case-tuple4 f p) (hn-ctxt (tuple4-assn P1' P2' P3' P4') p' p ** Γ1')*
R (case-tuple4 CP p) (case-tuple4$(λ$_2*a1 a2 a3 a4 . f' a1 a2 a3 a4)*$*p')*› (**is** ‹*?G Γ*›)
unfolding *autoref-tag-defs PROTECT2-def*
apply1 (*rule hn-refine-cons-pre*[*OF FR*])
apply1 (*cases p*; *cases p'*; *simp add*: *tuple4-assn-conv*[*THEN tuple4-assn-ctxt*])
unfolding *CP-SPLIT-def prod.simps*
apply (*rule hn-refine-cons*[*OF - Pair - entails-refl*])
applyS (*simp add*: *hn-ctxt-def*)
applyS *simp* **using** *FR2*
by (*simp add*: *hn-ctxt-def*)

lemma *case-tuple4-arity*[*sepref-monadify-arity*]:
‹*case-tuple4 ≡ λ*$_2*fp p. SP case-tuple4*$(λ$_2*a b. fp*$*a*$*b)*$*p*›
by (*simp-all only*: *SP-def APP-def PROTECT2-def RCALL-def*)

lemma *case-tuple4-comb*[*sepref-monadify-comb*]:
‹⋀*fp p. case-tuple4*$*fp*$*p ≡ Refine-Basic.bind*$(*EVAL*$*p*)$(λ$_2*p.* (*SP case-tuple4*$*fp*$*p*))›
by (*simp-all*)

lemma *case-tuple4-plain-comb*[*sepref-monadify-comb*]:
EVAL$(*case-tuple4*$(λ$_2*a1 a2 a3 a4 . fp a1 a2 a3 a4)*$*p*) ≡
Refine-Basic.bind$(*EVAL*$*p*)$(λ$_2*p. case-tuple4*$(λ$_2*a1 a2 a3 a4 . EVAL*$(*fp a1 a2 a3 a4)*)$*p*)
apply (*rule eq-reflection*, *simp split*: *list.split prod.split option.split tuple4.split*)+
done

lemma *ho-tuple4-move*[*sepref-preproc*]: ‹*case-tuple4 (λa1 a2 a3 a4 x. f x a1 a2 a3 a4) =*
(λp x. case-tuple4 (f x) p)›

by (*auto split*: *tuple4.splits*)

locale *tuple4-state* =
tuple4-state-ops a-assn b-assn c-assn d-assn
a-default a
b-default b
c-default c
d-default d
for
a-assn :: ‹'*a* ⇒ '*xa* ⇒ *assn*› **and**
b-assn :: ‹'*b* ⇒ '*xb* ⇒ *assn*› **and**
c-assn :: ‹'*c* ⇒ '*xc* ⇒ *assn*› **and**
d-assn :: ‹'*d* ⇒ '*xd* ⇒ *assn*› **and**
a-default :: '*a* **and**
a :: ‹'*xa llM*› **and**
b-default :: '*b* **and**
b :: ‹'*xb llM*› **and**
c-default :: '*c* **and**
c :: ‹'*xc llM*› **and**
d-default :: '*d* **and**
d :: ‹'*xd llM*› **and**
a-free :: ‹'*xa* ⇒ *unit llM*› **and**
b-free :: ‹'*xb* ⇒ *unit llM*› **and**
c-free :: ‹'*xc* ⇒ *unit llM*› **and**
d-free :: ‹'*xd* ⇒ *unit llM*› +
assumes
a: ‹(*uncurry0 a*, *uncurry0* (*RETURN a-default*)) ∈ *unit-assn*^{k} →$_a$ *a-assn*› **and**
b: ‹(*uncurry0 b*, *uncurry0* (*RETURN b-default*)) ∈ *unit-assn*^{k} →$_a$ *b-assn*› **and**
c: ‹(*uncurry0 c*, *uncurry0* (*RETURN c-default*)) ∈ *unit-assn*^{k} →$_a$ *c-assn*› **and**
d: ‹(*uncurry0 d*, *uncurry0* (*RETURN d-default*)) ∈ *unit-assn*^{k} →$_a$ *d-assn*› **and**
a-free: ‹*MK-FREE a-assn a-free*› **and**
b-free: ‹*MK-FREE b-assn b-free*›**and**
c-free: ‹*MK-FREE c-assn c-free*›**and**
d-free: ‹*MK-FREE d-assn d-free*›
notes [[*sepref-register-adhoc a-default b-default c-default d-default*]]
begin

lemmas [*sepref-comb-rules*] = *hn-case-tuple4* '[*of* - *a-assn b-assn c-assn d-assn*,
unfolded tuple4-int-assn-def[*symmetric*]]

lemma *ex-tuple4-iff*: (∃ *b* :: (-,-,-,-)*tuple4*. *P b*) ⟷
(∃ *a b c d*. *P* (*Tuple4 a b c d*))
apply *auto*
apply (*case-tac b*)
by *force*

lemmas [*sepref-frame-free-rules*] = *a-free b-free c-free d-free*
sepref-register
‹*Tuple4*›
lemma [*sepref-fr-rules*]: ‹(*uncurry3* (*Mreturn oooo Tuple4*), *uncurry3* (*RETURN oooo Tuple4*))
∈ *a-assn*^{d} ∗$_a$ *b-assn*^{d} ∗$_a$ *c-assn*^{d} ∗$_a$ *d-assn*^{d}
→$_a$ *tuple4-int-assn*›
unfolding *tuple4-int-assn-def*
apply *sepref-to-hoare*
apply *vcg*
unfolding *ex-tuple4-iff*

apply *vcg'*
done

lemma [*sepref-frame-match-rules*]:
‹ *hn-ctxt*
(*tuple4-assn* (*invalid-assn a-assn*) (*invalid-assn b-assn*) (*invalid-assn c-assn*) (*invalid-assn d-assn*))
ax bx ⊢ *hn-val UNIV ax bx*›
unfolding *hn-ctxt-def invalid-assn-def tuple4-int-assn-def entails-def*
apply (*auto split*: *prod.split tuple4.splits elim*: *is-pureE*
simp: *sep-algebra-simps pure-part-pure-conj-eq*)
apply (*auto simp*: *pure-part-def*)
apply (*auto simp*: *pure-def pure-true-conv*)
done

lemma *RETURN-case-tuple4-inverse*: ‹*RETURN*
(*let* - = *M*
in ff) =
(*do* {- ← *mop-free M*;
RETURN (*ff*)})›
by (*auto intro*!: *ext simp*: *mop-free-def split*: *tuple4.splits*)

sepref-definition *update-a-code*
is ‹*uncurry* (*RETURN oo update-a*)›
:: ‹*a-assn*^{d} *∗*$_a$ *tuple4-int-assn*^{d} →$_a$ *tuple4-int-assn*›
supply [[*goals-limit=5*]]
unfolding *update-a-def tuple4.case-distrib comp-def RETURN-case-tuple4-inverse*
by *sepref*

sepref-definition *update-b-code*
is ‹*uncurry* (*RETURN oo update-b*)›
:: ‹*b-assn*^{d} *∗*$_a$ *tuple4-int-assn*^{d} →$_a$ *tuple4-int-assn*›
supply [[*goals-limit=1*]]
unfolding *update-b-def tuple4.case-distrib comp-def RETURN-case-tuple4-inverse*
by *sepref*

sepref-definition *update-c-code*
is ‹*uncurry* (*RETURN oo update-c*)›
:: ‹*c-assn*^{d} *∗*$_a$ *tuple4-int-assn*^{d} →$_a$ *tuple4-int-assn*›
supply [[*goals-limit=1*]]
unfolding *update-c-def tuple4.case-distrib comp-def RETURN-case-tuple4-inverse*
by *sepref*

sepref-definition *update-d-code*
is ‹*uncurry* (*RETURN oo update-d*)›
:: ‹*d-assn*^{d} *∗*$_a$ *tuple4-int-assn*^{d} →$_a$ *tuple4-int-assn*›
supply [[*goals-limit=1*]]
unfolding *update-d-def tuple4.case-distrib comp-def RETURN-case-tuple4-inverse*
by *sepref*

lemma *RETURN-case-tuple4-invers*: ‹(*RETURN* ∘∘ *case-tuple4*)
(*λx1 x2 x3 x4*.
ff x1 x2 x3 x4) =
case-tuple4
(*λx1 x2 x3 x4*.
RETURN (*ff x1 x2 x3 x4*))›
by (*auto intro*!: *ext split*: *tuple4.splits*)

lemmas [*sepref-fr-rules*] = *a b c d*

sepref-definition *remove-a-code*
is ‹*RETURN o remove-a*›
:: ‹ *tuple4-int-assn*^{d} →$_a$ *a-assn* ×$_a$ *tuple4-int-assn*›
unfolding *remove-a-def RETURN-case-tuple4-invers*
by *sepref*

sepref-definition *remove-b-code*
is ‹*RETURN o remove-b*›
:: ‹ *tuple4-int-assn*^{d} →$_a$ *b-assn* ×$_a$ *tuple4-int-assn*›
unfolding *remove-b-def RETURN-case-tuple4-invers*
by *sepref*

sepref-definition *remove-c-code*
is ‹*RETURN o remove-c*›
:: ‹ *tuple4-int-assn*^{d} →$_a$ *c-assn* ×$_a$ *tuple4-int-assn*›
unfolding *remove-c-def RETURN-case-tuple4-invers*
by *sepref*

sepref-definition *remove-d-code*
is ‹*RETURN o remove-d*›
:: ‹ *tuple4-int-assn*^{d} →$_a$ *d-assn* ×$_a$ *tuple4-int-assn*›
unfolding *remove-d-def RETURN-case-tuple4-invers*
by *sepref*

lemmas *separation-rules* =
remove-a-code.refine
remove-b-code.refine
remove-c-code.refine
remove-d-code.refine

lemmas *code-rules* =
remove-a-code-def
remove-b-code-def
remove-c-code-def
remove-d-code-def

lemmas *setter-and-getters-def* =
update-a-def remove-a-def
update-b-def remove-b-def
update-c-def remove-c-def
update-d-def remove-d-def

end

context *tuple4-state*
begin
lemma *reconstruct-isasat*[*sepref-frame-match-rules*]:
‹*hn-ctxt*
(*tuple4-assn* (*a-assn*) (*b-assn*) (*c-assn*) (*d-assn*)) *ax bx* ⊢ *hn-ctxt tuple4-int-assn ax bx*›
unfolding *tuple4-int-assn-def*
apply (*auto split*: *prod.split tuple4.splits elim*: *is-pureE*

simp: *sep-algebra-simps pure-part-pure-conj-eq*)
done

context
fixes *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and**
read-all-code :: ‹'*xa* ⇒ '*xb* ⇒ '*xc* ⇒ '*xd* ⇒ '*q llM*› **and**
read-all :: ‹'*a* ⇒ '*b* ⇒ '*c* ⇒ '*d* ⇒ '*r nres*›
begin

definition *read-all-st-code* :: ‹-› **where**
‹*read-all-st-code xi* = (*case xi of*
Tuple4 a1 a2 a3 a4 ⇒
read-all-code a1 a2 a3 a4)›

definition *read-all-st* :: ‹('*a*, '*b*, '*c*, '*d*) *tuple4* ⇒ -› **where**
‹*read-all-st tuple4* = (*case tuple4 of Tuple4 a1 a2 a3 a4* ⇒
read-all a1 a2 a3 a4)›

context
fixes *P*
assumes *trail-read*[*sepref-fr-rules*]: ‹(*uncurry3 read-all-code*, *uncurry3 read-all*) ∈
[*uncurry3 P*]$_a$ *a-assn*^{k} *$*_a$ *b-assn*^{k} *$*_a$ *c-assn*^{k} *$*_a$ *d-assn*^{k} → *x-assn*›
notes [[*sepref-register-adhoc read-all*]]
begin
sepref-definition *read-all-st-code-tmp*
is *read-all-st*
:: ‹[*case-tuple4 P*]$_a$ *tuple4-int-assn*^{k} → *x-assn*›
unfolding *read-all-st-def*
by *sepref*

lemmas *read-all-st-code-refine* =
read-all-st-code-tmp.refine[*unfolded read-all-st-code-tmp-def*
read-all-st-code-def[*symmetric*]]
end

end

end

lemma *Mreturn-comp-Tuple4*:
‹(*Mreturn oooo Tuple4*) *a b c d* =
Mreturn (*Tuple4 a b c d*)›
by *auto*

lemma *tuple4-free*[*sepref-frame-free-rules*]:
assumes
‹*MK-FREE A freea*› ‹*MK-FREE B freeb*› ‹*MK-FREE C freec*› ‹*MK-FREE D freed*›
shows
‹
MK-FREE (*tuple4-assn A B C D*) (λ*S*. *case S of Tuple4 a b c d* ⇒ *do*$_M$ {
freea a; *freeb b*; *freec c*; *freed d*
})›
supply [*vcg-rules*] = *assms*[*THEN MK-FREED*]
apply (*rule*)+
apply (*clarsimp split*: *tuple4.splits*)
by *vcg'*

end
theory *IsaSAT-ACIDS-LLVM*
imports *IsaSAT-Literals-LLVM*
IsaSAT-ACIDS
Pairing-Heaps-Impl-LLVM
IsaSAT-Trail-LLVM
begin

definition *acids-assn2* **where**
‹*acids-assn2* = *acids-assn* ×$_a$ *uint64-nat-assn*›

sepref-register *ACIDS.mop-prio-insert-unchanged ACIDS.mop-prio-insert-raw-unchanged*
sepref-def *mop-prio-insert-raw-unchanged-impl*
is ‹*uncurry ACIDS.mop-prio-insert-raw-unchanged*›
:: ‹*atom-assn*^{k} *∗*$_a$ *acids-assn*^{d} →$_a$ *acids-assn*›
unfolding *ACIDS.mop-prio-insert-raw-unchanged-def*
by *sepref*

sepref-def *mop-prio-insert-unchanged-impl*
is ‹*uncurry ACIDS.mop-prio-insert-unchanged*›
:: ‹*atom-assn*^{k} *∗*$_a$ *acids-assn*^{d} →$_a$ *acids-assn*›
unfolding *ACIDS.mop-prio-insert-unchanged-def*
by *sepref*

sepref-def *acids-tl-impl*
is ‹*uncurry acids-tl*›
:: ‹*atom-assn*^{k} *∗*$_a$ *acids-assn2*^{d} →$_a$ *acids-assn2*›
unfolding *acids-assn2-def acids-tl-def max-def*
by *sepref*

sepref-def *acids-pop-min-impl*
is *acids-pop-min*
:: ‹*acids-assn2*^{d} →$_a$ *atom-assn* ×$_a$ *acids-assn2*›
unfolding *acids-pop-min-def acids-assn2-def*
by *sepref*

term *ACIDS.mop-prio-insert-maybe*

sepref-register *ACIDS.mop-prio-insert-maybe*
sepref-def *mop-prio-insert-maybe-impl*
is ‹*uncurry2* (*PR-CONST ACIDS.mop-prio-insert-maybe*)›
:: ‹*atom-assn*^{k} *∗*$_a$ *uint64-nat-assn*^{k} *∗*$_a$ *acids-assn*^{d} →$_a$ *acids-assn*›
unfolding *ACIDS.mop-prio-insert-maybe-def PR-CONST-def*
by *sepref*

sepref-def *acids-push-literal-impl*
is ‹*uncurry acids-push-literal*›
:: ‹*atom-assn*^{k} *∗*$_a$ *acids-assn2*^{d} →$_a$ *acids-assn2*›
unfolding *acids-push-literal-def acids-assn2-def*
min-def
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
by *sepref*

definition *bottom-acids0* :: ‹-› **where**
‹*bottom-acids0* = ((*replicate 0 None*, *replicate 0 None*, *replicate 0 None*, *replicate 0 None*, *replicate 0*

0, *None*))›

definition *bottom-acids* :: ‹-› **where**
‹*bottom-acids* = (*bottom-acids0*, *None*)›

sepref-def *bottom-acids0-impl*
is ‹*uncurry0* (*RETURN bottom-acids0*)›
:: ‹*unit-assn*^{k} →$_a$ *hp-assn*›
unfolding *bottom-acids0-def*
apply (*rewrite at* ‹(-, -, -, -, *replicate 0* ⌑ , -)›
unat-const-fold[**where** *'a=64*])
unfolding *hp-assn-def atom.fold-option array-fold-custom-replicate*
al-fold-custom-empty[**where** *'l=64*]
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

definition *empty-acids0* **where**
‹*empty-acids0* = ({#}, {#}, λ-::*nat*. *0*::*nat*)›

definition *empty-acids* **where**
‹*empty-acids* = (*empty-acids0*, *0*)›

lemma *bottom-acids0*:
‹(*uncurry0* (*RETURN bottom-acids0*), *uncurry0* (*RETURN empty-acids0*)) ∈
unit-rel →$_f$ ⟨((⟨*nat-rel*⟩*option-rel*, ⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel*)) *O*
acids-encoded-hmrel⟩*nres-rel*›
proof −
have [*intro*!]: ‹*Me* ∈# {#} ⟹ *False*›**for** *Me*
by *auto*
have *1*: ‹(({#}, (λ-. *None*, λ-. *None*, λ-. *None*, λ-. *None*, λ-. *None*), *None*), *empty-acids0*) ∈
acids-encoded-hmrel›
by (*auto simp*: *acids-encoded-hmrel-def bottom-acids0-def pairing-heaps-rel-def map-fun-rel-def*
ACIDS.hmrel-def encoded-hp-prop-list-conc-def encoded-hp-prop-def empty-outside-def empty-acids0-def
intro!: *relcompI*)

show *?thesis*
unfolding *uncurry0-def*
apply (*intro frefI nres-relI*)
apply (*auto intro*!: *relcompI*[*OF - 1*])
by(*auto simp*: *acids-encoded-hmrel-def bottom-acids0-def pairing-heaps-rel-def map-fun-rel-def*
ACIDS.hmrel-def encoded-hp-prop-list-conc-def encoded-hp-prop-def empty-outside-def)
qed

lemmas [*sepref-fr-rules*] =
bottom-acids0-impl.refine[*FCOMP bottom-acids0*, *unfolded hr-comp-assoc*[*symmetric*] *acids-assn-def*[*symmetric*]]

sepref-def *empty-acids-code*
is ‹*uncurry0* (*RETURN empty-acids*)›
:: ‹*unit-assn*^{k} →$_a$ *acids-assn2*›
unfolding *empty-acids-def acids-assn2-def*
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
by *sepref*

schematic-goal *free-acids-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE acids-assn ?a*›
unfolding *acids-assn-def hp-assn-def*

by *synthesize-free*

schematic-goal *free-acids-assn2*[*sepref-frame-free-rules*]: ‹*MK-FREE acids-assn2 ?a*›
unfolding *acids-assn2-def*
by *synthesize-free*

sepref-def *free-acids*
is ‹*mop-free*›
:: ‹*acids-assn2*^{d} $\rightarrow_a$ *unit-assn*›
by *sepref*

lemma *free-acids-assn2'*: ‹*MK-FREE acids-assn2 free-acids*›
unfolding *free-acids-def*
by (*rule back-subst*[*of* ‹*MK-FREE acids-assn2*›, *OF free-acids-assn2*])
(*auto intro*!: *ext*)

end
theory *IsaSAT-Bump-Heuristics-State-LLVM*
imports *IsaSAT-Bump-Heuristics-State*
IsaSAT-VMTF-Setup-LLVM
Tuple4-LLVM
IsaSAT-ACIDS-LLVM
begin
hide-const (**open**) *NEMonad.ASSERT NEMonad.RETURN*

type-synonym *bump-heuristics-assn* = ‹
((*32 word ptr* × *32 word ptr* × *32 word ptr* × *32 word ptr* × *64 word ptr* × *32 word*) × *64 word*,
(*64 word* × *32 word* × *32 word*) *ptr* × *64 word* × *32 word* × *32 word* × *32 word*,
1 word, (*64 word* × *64 word* × *32 word ptr*) × *1 word ptr*) *tuple4*›

definition *heuristic-bump-assn* :: ‹*bump-heuristics* ⇒ *bump-heuristics-assn* ⇒ -› **where**
‹*heuristic-bump-assn* = *tuple4-assn acids-assn2 vmtf-assn bool1-assn distinct-atoms-assn*›

definition *bottom-atom* **where**
‹*bottom-atom* = *0*›

definition *bottom-vmtf* :: ‹*vmtf*› **where**
‹*bottom-vmtf* = ((*replicate 0* (*VMTF-Node 0 None None*), *0*, *bottom-atom*, *bottom-atom*, *None*))›

definition *extract-bump-stable* **where**
‹*extract-bump-stable* = *tuple4-state-ops.remove-a empty-acids*›
definition *extract-bump-focused* **where**
‹*extract-bump-focused* = *tuple4-state-ops.remove-b bottom-vmtf*›

lemma [*sepref-fr-rules*]: ‹(*uncurry0* (*Mreturn 0*), *uncurry0* (*RETURN bottom-atom*)) ∈ *unit-assn*^{k} $\rightarrow_a$ *atom-assn*›
unfolding *bottom-atom-def*
apply *sepref-to-hoare*
apply *vcg*
apply (*auto simp*: *atom-rel-def unat-rel-def unat.rel-def br-def entails-def ENTAILS-def*)
by (*smt* (*verit*, *best*) *pure-true-conv rel-simps*(*51*) *sep.add-0*)

sepref-def *bottom-vmtf-code*
is ‹*uncurry0* (*RETURN bottom-vmtf*)›

:: ‹*unit-assn*^{k} $\rightarrow_a$ *vmtf-assn*›
unfolding *bottom-vmtf-def*
apply (*rewrite* **in** ‹((¤, -, -, -, -))› *array-fold-custom-replicate*)
unfolding
atom.fold-option array-fold-custom-replicate vmtf-assn-def
al-fold-custom-empty[**where** *'l=64*]
apply (*rewrite at 0* **in** ‹*VMTF-Node* ¤› *unat-const-fold*[**where** *'a=64*])
apply (*rewrite at* ‹(-, ¤, -, -)› *unat-const-fold*[**where** *'a=64*])
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

schematic-goal *free-vmtf-remove-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE vmtf-assn ?a*›
unfolding *vmtf-assn-def*
by *synthesize-free*

sepref-def *free-vmtf-remove*
is ‹*mop-free*›
:: ‹*vmtf-assn*^{d} $\rightarrow_a$ *unit-assn*›
by *sepref*

sepref-def *bottom-focused*
is ‹*uncurry0* (*RETURN False*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *bool1-assn*›
by *sepref*
sepref-def *free-focused*
is ‹*mop-free*›
:: ‹*bool1-assn*^{d} $\rightarrow_a$ *unit-assn*›
by *sepref*

definition *extract-bump-is-focused* **where**
‹*extract-bump-is-focused* = *tuple4-state-ops.remove-c False*›

definition *bottom-atms-hash* **where**
‹*bottom-atms-hash* = ([], *replicate 0 False*)›

definition *extract-bump-atms-to-bump* **where**
‹*extract-bump-atms-to-bump* = *tuple4-state-ops.remove-d bottom-atms-hash*›

sepref-def *bottom-atms-hash-code*
is ‹*uncurry0* (*RETURN bottom-atms-hash*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *distinct-atoms-assn*›
unfolding *bottom-atms-hash-def*
unfolding
atom.fold-option array-fold-custom-replicate
al-fold-custom-empty[**where** *'l=64*]
apply (*rewrite at* ‹(¤, -)› *annotate-assn*[**where** *A* =‹*al-assn atom-assn*›])
apply (*rewrite at* ‹(-, ¤)› *annotate-assn*[**where** *A* =‹*atoms-hash-assn*›])
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*
sepref-def *free-atms-hash-code*
is ‹*mop-free*›
:: ‹*distinct-atoms-assn*^{d} $\rightarrow_a$ *unit-assn*›
by *sepref*

lemma *free-vmtf-assn-assn2*: ‹*MK-FREE vmtf-assn free-vmtf-remove*›
unfolding *free-vmtf-remove-def*

by (*rule back-subst*[*of* ‹*MK-FREE vmtf-assn*›, *OF free-vmtf-remove-assn*])
(*auto intro*!: *ext*)

schematic-goal *free-focused-assn*: ‹*MK-FREE bool1-assn ?a*›
unfolding *vmtf-assn-def*
by *synthesize-free*
schematic-goal *free-distinct-atoms-assn*: ‹*MK-FREE distinct-atoms-assn ?a*›
by *synthesize-free*

lemma *free-focused-assn2*: ‹*MK-FREE bool1-assn free-focused*›
unfolding *free-focused-def*
by (*rule back-subst*[*of* ‹*MK-FREE bool1-assn*›, *OF free-focused-assn*])
(*auto intro*!: *ext*)

lemma *free-distinct-atoms-assn2*: ‹*MK-FREE* (*distinct-atoms-assn*) *free-atms-hash-code*›
unfolding *free-atms-hash-code-def*
by (*rule back-subst*[*of* ‹*MK-FREE distinct-atoms-assn*›, *OF free-distinct-atoms-assn*])
(*auto intro*!: *ext*)

global-interpretation *Bump-Heur*: *tuple4-state* **where**
a-assn = *acids-assn2* **and**
b-assn = *vmtf-assn* **and**
c-assn = *bool1-assn* **and**
d-assn = *distinct-atoms-assn* **and**
a-default = *empty-acids* **and**
a = ‹*empty-acids-code*› **and**
a-free = *free-acids* **and**
b-default = *bottom-vmtf* **and**
b = ‹*bottom-vmtf-code*› **and**
b-free = *free-vmtf-remove* **and**
c-default = *False* **and**
c = ‹*bottom-focused*› **and**
c-free = *free-focused* **and**
d-default = ‹*bottom-atms-hash*› **and**
d = ‹*bottom-atms-hash-code*› **and**
d-free = ‹*free-atms-hash-code*›
rewrites
‹*Bump-Heur.tuple4-int-assn* ≡ *heuristic-bump-assn*›**and**
‹*Bump-Heur.remove-a* ≡ *extract-bump-stable*› **and**
‹*Bump-Heur.remove-b* ≡ *extract-bump-focused*› **and**
‹*Bump-Heur.remove-c* ≡ *extract-bump-is-focused*› **and**
‹*Bump-Heur.remove-d* ≡ *extract-bump-atms-to-bump*›
apply *unfold-locales*
apply (*rule bottom-vmtf-code.refine bottom-focused.refine empty-acids-code.refine*
bottom-atms-hash-code.refine free-vmtf-assn-assn2 free-focused-assn2 free-acids-assn2′
free-distinct-atoms-assn2)+
subgoal unfolding *heuristic-bump-assn-def tuple4-state-ops.tuple4-int-assn-def* **by** *auto*
subgoal unfolding *extract-bump-stable-def* **by** *auto*
subgoal unfolding *extract-bump-focused-def* **by** *auto*
subgoal unfolding *extract-bump-is-focused-def* **by** *auto*
subgoal unfolding *extract-bump-atms-to-bump-def* **by** *auto*
done

lemmas [*unfolded Tuple4-LLVM.inline-direct-return-node-case*, *llvm-code*] =
Bump-Heur.code-rules[*unfolded Mreturn-comp-Tuple4*]

lemmas [*sepref-fr-rules*] =
Bump-Heur.separation-rules

lemma *switch-bump-heur-alt-def*:
‹*RETURN o switch-bump-heur* = (λ*x*. *case x of Bump-Heuristics hstable focused foc b* ⇒ *do* {
f ← *RETURN* (¬*foc*);
mop-free foc;
RETURN (*Bump-Heuristics hstable focused* (*f*) *b*)})›
by (*auto intro*!: *ext simp*: *mop-free-def switch-bump-heur-def split*: *bump-heuristics-splits*)

sepref-def *switch-bump-heur-code*
is ‹*RETURN o switch-bump-heur*›
:: ‹*heuristic-bump-assn*^{d} →$_a$ *heuristic-bump-assn*›
unfolding *switch-bump-heur-alt-def*
by *sepref*

end
theory *Tuple17-LLVM*
imports *Tuple17 IsaSAT-Literals-LLVM*
begin

hide-const (**open**) *NEMonad.ASSERT NEMonad.RETURN*

instantiation *tuple17* ::
(*llvm-rep*,*llvm-rep*,*llvm-rep*,*llvm-rep*,
llvm-rep,*llvm-rep*,*llvm-rep*,*llvm-rep*,
llvm-rep,*llvm-rep*,*llvm-rep*,*llvm-rep*,
llvm-rep,*llvm-rep*,*llvm-rep*,*llvm-rep*,*llvm-rep*) *llvm-rep*
begin
definition *to-val-tuple17* **where**
‹*to-val-tuple17* ≡ (λ*S*. *case S of*
Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs ⇒ *LL-STRUCT* [*to-val M*, *to-val N*, *to-val D*, *to-val i*, *to-val W*, *to-val ivmtf*,
to-val icount, *to-val ccach*, *to-val lbd*,
to-val outl, *to-val stats*, *to-val heur*, *to-val aivdom*, *to-val clss*, *to-val opts*, *to-val arena*, *to-val occs*])›

definition *from-val-tuple17* :: ‹*llvm-val* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17*› **where**
‹*from-val-tuple17* ≡ (λ*p*. *case llvm-val.the-fields p of*
[*M*, *N*, *D*, *i*, *W*, *ivmtf*, *icount*, *ccach*, *lbd*, *outl*, *stats*, *heur*, *aivdom*, *clss*, *opts*, *arena*, *occs*] ⇒
Tuple17 (*from-val M*) (*from-val N*) (*from-val D*) (*from-val i*) (*from-val W*) (*from-val ivmtf*) (*from-val icount*) (*from-val ccach*) (*from-val lbd*)
(*from-val outl*) (*from-val stats*) (*from-val heur*) (*from-val aivdom*) (*from-val clss*) (*from-val opts*) (*from-val arena*) (*from-val occs*))›

definition [*simp*]: *struct-of-tuple17* (- :: (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17 itself*) ≡
VS-STRUCT [*struct-of TYPE*(*'a*), *struct-of TYPE*(*'b*), *struct-of TYPE*(*'c*),
struct-of TYPE(*'d*), *struct-of TYPE*(*'e*), *struct-of TYPE*(*'f*), *struct-of TYPE*(*'g*), *struct-of TYPE*(*'h*),
struct-of TYPE(*'i*), *struct-of TYPE*(*'j*), *struct-of TYPE*(*'k*), *struct-of TYPE*(*'l*),
struct-of TYPE(*'m*), *struct-of TYPE*(*'n*), *struct-of TYPE*(*'o*), *struct-of TYPE*(*'p*),
struct-of TYPE(*'q*)]

definition [*simp*]: *init-tuple17* :: ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*, '*p*, '*q*) *tuple17* ≡ *Tuple17 init init init init init init init init init init init init init init init init init*

instance
apply *standard*
unfolding *from-val-tuple17-def to-val-tuple17-def struct-of-tuple17-def init-tuple17-def comp-def tuple17.case-distrib*
subgoal
by (*auto simp*: *init-zero fun-eq-iff from-val-tuple17-def split*: *tuple17.splits*)
subgoal for *v*
by (*cases v*) (*auto split*: *list.splits tuple17.splits*)
subgoal for *v*
by (*cases v*)
(*simp add*: *LLVM-Shallow.null-def to-val-ptr-def split*: *tuple17.splits*)
subgoal
by (*simp add*: *LLVM-Shallow.null-def to-val-ptr-def to-val-word-def init-zero split*: *tuple17.splits*)
done
end

Setup for LLVM code export

Declare structure to code generator.

lemma *to-val-tuple17*[*ll-struct-of*]: *struct-of TYPE*(('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*, '*p*, '*q*) *tuple17*) = *VS-STRUCT* [
struct-of TYPE('*a*::*llvm-rep*),
struct-of TYPE('*b*::*llvm-rep*),
struct-of TYPE('*c*::*llvm-rep*),
struct-of TYPE('*d*::*llvm-rep*),
struct-of TYPE('*e*::*llvm-rep*),
struct-of TYPE('*f*::*llvm-rep*),
struct-of TYPE('*g*::*llvm-rep*),
struct-of TYPE('*h*::*llvm-rep*),
struct-of TYPE('*i*::*llvm-rep*),
struct-of TYPE('*j*::*llvm-rep*),
struct-of TYPE('*k*::*llvm-rep*),
struct-of TYPE('*l*::*llvm-rep*),
struct-of TYPE('*m*::*llvm-rep*),
struct-of TYPE('*n*::*llvm-rep*),
struct-of TYPE('*o*::*llvm-rep*),
struct-of TYPE('*p*::*llvm-rep*),
struct-of TYPE('*q*::*llvm-rep*)]
by (*auto*)

This is the old version of the declaration:

lemma *to-val x* = *LL-STRUCT* [
to-val (*Tuple17-get-a x*),
to-val (*Tuple17-get-b x*),
to-val (*Tuple17-get-c x*),
to-val (*Tuple17-get-d x*),
to-val (*Tuple17-get-e x*),
to-val (*Tuple17-get-f x*),
to-val (*Tuple17-get-g x*),
to-val (*Tuple17-get-h x*),
to-val (*Tuple17-get-i x*),
to-val (*Tuple17-get-j x*),

to-val (*Tuple17-get-k x*),
to-val (*Tuple17-get-l x*),
to-val (*Tuple17-get-m x*),
to-val (*Tuple17-get-n x*),
to-val (*Tuple17-get-o x*),
to-val (*Tuple17-get-p x*),
to-val (*Tuple17-get-q x*)]
apply (*cases x*)
apply (*auto simp*: *to-val-tuple17-def*)
done

lemma *node-insert-value*:
ll-insert-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *M' 0* = *Mreturn* (*Tuple17 M' N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*)
ll-insert-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *N' (Suc 0)* = *Mreturn* (*Tuple17 M N' D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*)
ll-insert-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *D' 2* = *Mreturn* (*Tuple17 M N D' i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*)
ll-insert-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *i' 3* = *Mreturn* (*Tuple17 M N D i' W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*)
ll-insert-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *W' 4* = *Mreturn* (*Tuple17 M N D i W' ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*)
ll-insert-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *ivmtf' 5* = *Mreturn* (*Tuple17 M N D i W ivmtf' icount ccach lbd outl stats heur aivdom clss opts arena occs*)
ll-insert-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *icount' 6* = *Mreturn* (*Tuple17 M N D i W ivmtf icount' ccach lbd outl stats heur aivdom clss opts arena occs*)
ll-insert-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *ccach' 7* = *Mreturn* (*Tuple17 M N D i W ivmtf icount ccach' lbd outl stats heur aivdom clss opts arena occs*)
ll-insert-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *lbd' 8* = *Mreturn* (*Tuple17 M N D i W ivmtf icount ccach lbd' outl stats heur aivdom clss opts arena occs*)
ll-insert-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *outl' 9* = *Mreturn* (*Tuple17 M N D i W ivmtf icount ccach lbd outl' stats heur aivdom clss opts arena occs*)
ll-insert-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *stats' 10* = *Mreturn* (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats' heur aivdom clss opts arena occs*)
ll-insert-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *heur' 11* = *Mreturn* (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur' aivdom clss opts arena occs*)
ll-insert-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *aivdom' 12* = *Mreturn* (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom' clss opts arena occs*)
ll-insert-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *clss' 13* = *Mreturn* (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss' opts arena occs*)
ll-insert-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *opts' 14* = *Mreturn* (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts' arena occs*)

ll-insert-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *arena′ 15* = *Mreturn* (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena′ occs*)
ll-insert-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *occs′ 16* = *Mreturn* (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs′*)
by (*simp-all add*: *ll-insert-value-def llvm-insert-value-def Let-def checked-from-val-def to-val-tuple17-def from-val-tuple17-def*)

lemma *node-extract-value*:
ll-extract-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *0* = *Mreturn M*
ll-extract-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) (*Suc 0*) = *Mreturn N*
ll-extract-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *2* = *Mreturn D*
ll-extract-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *3* = *Mreturn i*
ll-extract-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *4* = *Mreturn W*
ll-extract-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *5* = *Mreturn ivmtf*
ll-extract-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *6* = *Mreturn icount*
ll-extract-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *7* = *Mreturn ccach*
ll-extract-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *8* = *Mreturn lbd*
ll-extract-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *9* = *Mreturn outl*
ll-extract-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *10* = *Mreturn stats*
ll-extract-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *11* = *Mreturn heur*
ll-extract-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *12* = *Mreturn aivdom*
ll-extract-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *13* = *Mreturn clss*
ll-extract-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *14* = *Mreturn opts*
ll-extract-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *15* = *Mreturn arena*
ll-extract-value (*Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*) *16* = *Mreturn occs*
apply (*simp-all add*: *ll-extract-value-def llvm-extract-value-def Let-def checked-from-val-def to-val-tuple17-def from-val-tuple17-def*)
done

Lemmas to translate node construction and destruction

lemma *inline-return-node*[*llvm-pre-simp*]: *Mreturn* (*Tuple17 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs*) = *doM* {
r ← *ll-insert-value init M 0*;
r ← *ll-insert-value r N 1*;
r ← *ll-insert-value r D 2*;
r ← *ll-insert-value r i 3*;
r ← *ll-insert-value r W 4*;

```
    r ← ll-insert-value r ivmtf 5;
    r ← ll-insert-value r icount 6;
    r ← ll-insert-value r ccach 7;
    r ← ll-insert-value r lbd 8;
    r ← ll-insert-value r outl 9;
    r ← ll-insert-value r heur 10;
    r ← ll-insert-value r stats 11;
    r ← ll-insert-value r aivdom 12;
    r ← ll-insert-value r clss 13;
    r ← ll-insert-value r opts 14;
    r ← ll-insert-value r arena 15;
    r ← ll-insert-value r occs 16;
    Mreturn r
  }
  apply (auto simp: node-insert-value)
  done

lemma inline-node-case[llvm-pre-simp]: (case r of (Tuple17 M N D i W ivmtf icount ccach lbd outl heur
stats aivdom clss opts arena occs) ⇒ f M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss
opts arena occs) = doM {
    M ← ll-extract-value r 0;
    N ← ll-extract-value r 1;
    D ← ll-extract-value r 2;
    i ← ll-extract-value r 3;
    W ← ll-extract-value r 4;
    ivmtf ← ll-extract-value r 5;
    icount ← ll-extract-value r 6;
    ccach ← ll-extract-value r 7;
    lbd ← ll-extract-value r 8;
    outl ← ll-extract-value r 9;
    heur ← ll-extract-value r 10;
    stats ← ll-extract-value r 11;
    aivdom ← ll-extract-value r 12;
    clss ← ll-extract-value r 13;
    opts ← ll-extract-value r 14;
    arena ← ll-extract-value r 15;
    occs ← ll-extract-value r 16;
  f M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs
}
  apply (cases r)
  apply (auto simp: node-extract-value)
  done

lemma inline-return-node-case[llvm-pre-simp]: doM {Mreturn (case r of (Tuple17 M N D i W ivmtf
icount ccach lbd outl heur stats aivdom clss opts arena occs) ⇒ f M N D i W ivmtf icount ccach lbd outl
heur stats aivdom clss opts arena occs)} = doM {
    M ← ll-extract-value r 0;
    N ← ll-extract-value r 1;
    D ← ll-extract-value r 2;
    i ← ll-extract-value r 3;
    W ← ll-extract-value r 4;
    ivmtf ← ll-extract-value r 5;
    icount ← ll-extract-value r 6;
    ccach ← ll-extract-value r 7;
    lbd ← ll-extract-value r 8;
    outl ← ll-extract-value r 9;
```

```
    heur ← ll-extract-value r 10;
    stats ← ll-extract-value r 11;
    aivdom ← ll-extract-value r 12;
    clss ← ll-extract-value r 13;
    opts ← ll-extract-value r 14;
    arena ← ll-extract-value r 15;
    occs ← ll-extract-value r 16;
  Mreturn (f M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs)
}
  apply (cases r)
  apply (auto simp: node-extract-value)
  done
lemma inline-direct-return-node-case[llvm-pre-simp]: doM {(case r of (Tuple17 M N D i W ivmtf icount
ccach lbd outl heur stats aivdom clss opts arena occs) ⇒ f M N D i W ivmtf icount ccach lbd outl heur
stats aivdom clss opts arena occs)} = doM {
    M ← ll-extract-value r 0;
    N ← ll-extract-value r 1;
    D ← ll-extract-value r 2;
    i ← ll-extract-value r 3;
    W ← ll-extract-value r 4;
    ivmtf ← ll-extract-value r 5;
    icount ← ll-extract-value r 6;
    ccach ← ll-extract-value r 7;
    lbd ← ll-extract-value r 8;
    outl ← ll-extract-value r 9;
    heur ← ll-extract-value r 10;
    stats ← ll-extract-value r 11;
    aivdom ← ll-extract-value r 12;
    clss ← ll-extract-value r 13;
    opts ← ll-extract-value r 14;
    arena ← ll-extract-value r 15;
    occs ← ll-extract-value r 16;
   (f M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs)
}
  apply (cases r)
  apply (auto simp: node-extract-value)
  done

lemmas [llvm-inline] =
  tuple17.Tuple17-get-a-def
  tuple17.Tuple17-get-b-def
  tuple17.Tuple17-get-c-def
  tuple17.Tuple17-get-d-def
  tuple17.Tuple17-get-e-def
  tuple17.Tuple17-get-f-def
  tuple17.Tuple17-get-g-def
  tuple17.Tuple17-get-h-def
  tuple17.Tuple17-get-i-def
  tuple17.Tuple17-get-j-def
  tuple17.Tuple17-get-l-def
  tuple17.Tuple17-get-k-def
  tuple17.Tuple17-get-m-def
  tuple17.Tuple17-get-n-def
  tuple17.Tuple17-get-o-def
  tuple17.Tuple17-get-p-def
  tuple17.Tuple17-get-q-def
```

fun *tuple17-assn* :: ‹
('a ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('b ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('c ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('d ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('e ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('f ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('g ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('h ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('i ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('j ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('k ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('l ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('m ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('n ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('o ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('p ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('q ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ - ⇒ *assn*› **where**
‹*tuple17-assn a-assn b-assn' c-assn d-assn e-assn f-assn g-assn h-assn i-assn j-assn k-assn l-assn m-assn n-assn o-assn p-assn q-assn S T* =
(*case* (*S*, *T*) *of*
(*Tuple17 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts arena occs*,
Tuple17 M' N' D' i' W' ivmtf' icount' ccach' lbd' outl' heur' stats' aivdom' clss' opts' arena' occs')
⇒
(*a-assn M* (*M'*) ∧* *b-assn' N* (*N'*) ∧* *c-assn D* (*D'*) ∧* *d-assn i* (*i'*) ∧*
e-assn W (*W'*) ∧* *f-assn ivmtf* (*ivmtf'*) ∧* *g-assn icount* (*icount'*) ∧* *h-assn ccach* (*ccach'*) ∧*
i-assn lbd (*lbd'*) ∧* *j-assn outl* (*outl'*) ∧* *k-assn heur* (*heur'*) ∧* *l-assn stats* (*stats'*) ∧*
m-assn aivdom (*aivdom'*) ∧* *n-assn clss* (*clss'*) ∧* *o-assn opts* (*opts'*) ∧* *p-assn arena* (*arena'*) ∧*
q-assn occs occs'))
›

locale *isasat-state-ops* =
fixes
a-assn :: ‹'a ⇒ 'xa :: *llvm-rep* ⇒ *assn*› **and**
b-assn :: ‹'b ⇒ 'xb:: *llvm-rep* ⇒ *assn*› **and**
c-assn :: ‹'c ⇒ 'xc:: *llvm-rep* ⇒ *assn*› **and**
d-assn :: ‹'d ⇒ 'xd:: *llvm-rep* ⇒ *assn*› **and**
e-assn :: ‹'e ⇒ 'xe:: *llvm-rep* ⇒ *assn*› **and**
f-assn :: ‹'f ⇒ 'xf:: *llvm-rep* ⇒ *assn*› **and**
g-assn :: ‹'g ⇒ 'xg:: *llvm-rep* ⇒ *assn*› **and**
h-assn :: ‹'h ⇒ 'xh:: *llvm-rep* ⇒ *assn*› **and**
i-assn :: ‹'i ⇒ 'xi:: *llvm-rep* ⇒ *assn*› **and**
j-assn :: ‹'j ⇒ 'xj:: *llvm-rep* ⇒ *assn*› **and**
k-assn :: ‹'k ⇒ 'xk:: *llvm-rep* ⇒ *assn*› **and**
l-assn :: ‹'l ⇒ 'xl:: *llvm-rep* ⇒ *assn*› **and**
m-assn :: ‹'m ⇒ 'xm:: *llvm-rep* ⇒ *assn*› **and**
n-assn :: ‹'n ⇒ 'xn:: *llvm-rep* ⇒ *assn*› **and**
o-assn :: ‹'o ⇒ 'xo:: *llvm-rep* ⇒ *assn*› **and**
p-assn :: ‹'p ⇒ 'xp:: *llvm-rep* ⇒ *assn*› **and**
q-assn :: ‹'q ⇒ 'xq:: *llvm-rep* ⇒ *assn*› **and**
a-default :: 'a **and**
a :: ‹'xa *llM*› **and**

b-default :: *'b* **and**
b :: ‹*'xb llM*› **and**
c-default :: *'c* **and**
c :: ‹*'xc llM*› **and**
d-default :: *'d* **and**
d :: ‹*'xd llM*› **and**
e-default :: *'e* **and**
e :: ‹*'xe llM*› **and**
f-default :: *'f* **and**
f :: ‹*'xf llM*› **and**
g-default :: *'g* **and**
g :: ‹*'xg llM*› **and**
h-default :: *'h* **and**
h :: ‹*'xh llM*› **and**
i-default :: *'i* **and**
i :: ‹*'xi llM*› **and**
j-default :: *'j* **and**
j :: ‹*'xj llM*› **and**
k-default :: *'k* **and**
k :: ‹*'xk llM*› **and**
l-default :: *'l* **and**
l :: ‹*'xl llM*› **and**
m-default :: *'m* **and**
m :: ‹*'xm llM*› **and**
n-default :: *'n* **and**
n :: ‹*'xn llM*› **and**
ko-default :: *'o* **and**
ko :: ‹*'xo llM*› **and**
p-default :: *'p* **and**
p :: ‹*'xp llM*›**and**
q-default :: *'q* **and**
q :: ‹*'xq llM*›
begin

definition *isasat-assn* :: ‹- ⇒ - ⇒ *assn*› **where**
‹*isasat-assn* = *tuple17-assn*
a-assn b-assn c-assn d-assn
e-assn f-assn g-assn h-assn
i-assn j-assn k-assn l-assn
m-assn n-assn o-assn p-assn
q-assn›

definition *remove-a* :: ‹(*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17* ⇒ - × (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17*› **where**
‹*remove-a tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
(*x1*, *Tuple17 a-default x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17*))›

definition *remove-b* :: ‹(*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17* ⇒ *'b* × (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17*› **where**
‹*remove-b tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
(*x2*, *Tuple17 x1 b-default x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17*))›

definition *remove-c*:: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ - × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17*› **where**
‹*remove-c tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
(*x3*, *Tuple17 x1 x2 c-default x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17*))›

definition *remove-d* :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ - × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17*› **where**
‹*remove-d tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
(*x4*, *Tuple17 x1 x2 x3 d-default x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17*))›

definition *remove-e* :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ 'e × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17*› **where**
‹*remove-e tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
(*x5*, *Tuple17 x1 x2 x3 x4 e-default x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17*))›

definition *remove-f* :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ 'f × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17*› **where**
‹*remove-f tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
(*x6*, *Tuple17 x1 x2 x3 x4 x5 f-default x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17*))›

definition *remove-g* :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ 'g × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17*› **where**
‹*remove-g tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
(*x7*, *Tuple17 x1 x2 x3 x4 x5 x6 g-default x8 x9 x10 x11 x12 x13 x14 x15 x16 x17*))›

definition *remove-h* :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ 'h × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17*› **where**
‹*remove-h tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
(*x8*, *Tuple17 x1 x2 x3 x4 x5 x6 x7 h-default x9 x10 x11 x12 x13 x14 x15 x16 x17*))›

definition *remove-i* :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ 'i × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17*› **where**
‹*remove-i tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
(*x9*, *Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 i-default x10 x11 x12 x13 x14 x15 x16 x17*))›

definition *remove-j* :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ 'j × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17*› **where**
‹*remove-j tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
(*x10*, *Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 j-default x11 x12 x13 x14 x15 x16 x17*))›

definition *remove-k* :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ 'k × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17*› **where**
‹*remove-k tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
(*x11*, *Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 k-default x12 x13 x14 x15 x16 x17*))›

definition *remove-l* :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ 'l × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17*› **where**
‹*remove-l tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
(*x12*, *Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 l-default x13 x14 x15 x16 x17*))›

definition *remove-m* :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ 'm × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17*› **where**
‹*remove-m tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
(*x13*, *Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 m-default x14 x15 x16 x17*))›

definition *remove-n* :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ 'n × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17*› **where**
‹*remove-n tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
(*x14*, *Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 n-default x15 x16 x17*))›

definition *remove-o* :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ 'o × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17*› **where**
‹*remove-o tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
(*x15*, *Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 ko-default x16 x17*))›

definition *remove-p* :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ 'p × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17*› **where**
‹*remove-p tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
(*x16*, *Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 p-default x17*))›

definition *remove-q* :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ 'q × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17*› **where**
‹*remove-q tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
(*x17*, *Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 q-default*))›

definition *update-a* :: ‹'a ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17* ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) *tuple17*› **where**
‹*update-a x1 tuple17* = (*case tuple17 of Tuple17 M x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
let - = M in
Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17)›

definition *update-b* :: ‹'b ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) tuple17 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) tuple17› **where**
‹*update-b x2 tuple17* = (*case tuple17 of Tuple17 x1 M x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
let - = M in
Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17)›

definition *update-c*:: ‹'c ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) tuple17 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) tuple17› **where**
‹*update-c x3 tuple17* = (*case tuple17 of Tuple17 x1 x2 M x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
let - = M in
Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17)›

definition *update-d* :: ‹'d ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) tuple17 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) tuple17› **where**
‹*update-d x4 tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 M x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
let - = M in
Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17)›

definition *update-e* :: ‹'e ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) tuple17 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) tuple17› **where**
‹*update-e x5 tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 M x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
let - = M in
Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17)›

definition *update-f* :: ‹'f ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) tuple17 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) tuple17› **where**
‹*update-f x6 tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 M x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
let - = M in
Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17)›

definition *update-g* :: ‹'g ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) tuple17 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) tuple17› **where**
‹*update-g x7 tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 M x8 x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
let - = M in
Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17)›

definition *update-h* :: ‹'h ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) tuple17 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o, 'p, 'q) tuple17› **where**
‹*update-h x8 tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 M x9 x10 x11 x12 x13 x14 x15 x16 x17* ⇒
let - = M in
Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17)›

definition *update-i* :: ‹*'i* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17*› **where**
‹*update-i x9 tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 M x10 x11 x12 x13 x14 x15 x16 x17* ⇒
let - = M in
Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17)›

definition *update-j* :: ‹*'j* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17*› **where**
‹*update-j x10 tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 M x11 x12 x13 x14 x15 x16 x17* ⇒
let - = M in
Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17)›

definition *update-k* :: ‹*'k* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17*› **where**
‹*update-k x11 tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 M x12 x13 x14 x15 x16 x17* ⇒
let - = M in
Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17)›

definition *update-l* :: ‹*'l* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17*› **where**
‹*update-l x12 tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 M x13 x14 x15 x16 x17* ⇒
let - = M in
Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17)›

definition *update-m* :: ‹*'m* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17*› **where**
‹*update-m x13 tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 M x14 x15 x16 x17* ⇒
let - = M in
Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17)›

definition *update-n* :: ‹*'n* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17*› **where**
‹*update-n x14 tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 M x15 x16 x17* ⇒
let - = M in
Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17)›

definition *update-o* :: ‹*'o* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17*› **where**
‹*update-o x15 tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 M x16 x17* ⇒
let - = M in
Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17)›

definition *update-p* :: ‹*'p* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17*› **where**
‹*update-p x16 tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 M x17* ⇒
let - = *M in*
Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17)›

definition *update-q* :: ‹*'q* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*, *'p*, *'q*) *tuple17*› **where**
‹*update-q x17 tuple17* = (*case tuple17 of Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 M* ⇒
let - = *M in*
Tuple17 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17)›

end

lemma *tuple17-assn-conv*[*simp*]:
tuple17-assn P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13 P14 P15 P16 P17 (*Tuple17 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17*)
(*Tuple17 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15' a16' a17'*) =
(*P1 a1 a1'* ∧*
P2 a2 a2' ∧*
P3 a3 a3' ∧*
P4 a4 a4' ∧*
P5 a5 a5' ∧*
P6 a6 a6' ∧*
P7 a7 a7' ∧*
P8 a8 a8' ∧* *P9 a9 a9'* ∧* *P10 a10 a10'* ∧* *P11 a11 a11'* ∧* *P12 a12 a12'* ∧* *P13 a13 a13'* ∧* *P14 a14 a14'* ∧* *P15 a15 a15'* ∧* *P16 a16 a16'*
∧* *P17 a17 a17'*)
unfolding *tuple17-assn.simps* **by** *auto*
lemma *tuple17-assn-ctxt*:
‹*tuple17-assn P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13 P14 P15 P16 P17* (*Tuple17 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17*)
(*Tuple17 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15' a16' a17'*) = *z* ⟹
hn-ctxt (*tuple17-assn P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13 P14 P15 P16 P17*) (*Tuple17 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17*)
(*Tuple17 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15' a16' a17'*)= *z*›
by (*simp add*: *hn-ctxt-def*)

lemma *hn-case-tuple17'*[*sepref-comb-rules*]:
assumes *FR*: ‹Γ ⊢ *hn-ctxt* (*tuple17-assn P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13 P14 P15 P16 P17*) *p' p* ** Γ*1*›
assumes *Pair*: ⋀*a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15' a16' a17'*.
⟦*p'*=*Tuple17 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15' a16' a17'*⟧
⟹ *hn-refine* (*hn-ctxt P1 a1' a1* ∧* *hn-ctxt P2 a2' a2* ∧* *hn-ctxt P3 a3' a3* ∧* *hn-ctxt P4 a4' a4* ∧*
hn-ctxt P5 a5' a5 ∧* *hn-ctxt P6 a6' a6* ∧* *hn-ctxt P7 a7' a7* ∧* *hn-ctxt P8 a8' a8* ∧*
hn-ctxt P9 a9' a9 ∧* *hn-ctxt P10 a10' a10* ∧* *hn-ctxt P11 a11' a11* ∧* *hn-ctxt P12 a12' a12* ∧*

hn-ctxt P13 a13' a13 ∧* *hn-ctxt P14 a14' a14* ∧* *hn-ctxt P15 a15' a15* ∧* *hn-ctxt P16 a16' a16* ∧* *hn-ctxt P17 a17' a17* ∧* Γ*1*)
(*f a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17*)
(Γ*2 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15' a16' a17'*) *R*
(*CP a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17*)
(*f' a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15' a16' a17'*)
assumes *FR2*: ‹⋀*a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15' a16' a17'*.
Γ*2 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15' a16' a17'* ⊢
hn-ctxt P1' a1' a1 ** *hn-ctxt P2' a2' a2* ** *hn-ctxt P3' a3' a3* ** *hn-ctxt P4' a4' a4* **
hn-ctxt P5' a5' a5 ** *hn-ctxt P6' a6' a6* ** *hn-ctxt P7' a7' a7* ** *hn-ctxt P8' a8' a8* **
hn-ctxt P9' a9' a9 ** *hn-ctxt P10' a10' a10* ** *hn-ctxt P11' a11' a11* ** *hn-ctxt P12' a12' a12* **
hn-ctxt P13' a13' a13 ** *hn-ctxt P14' a14' a14* ** *hn-ctxt P15' a15' a15* ** *hn-ctxt P16' a16' a16* **
hn-ctxt P17' a17' a17 ** Γ*1'*›
shows ‹*hn-refine* Γ (*case-tuple17 f p*) (*hn-ctxt* (*tuple17-assn P1' P2' P3' P4' P5' P6' P7' P8' P9' P10' P11' P12' P13' P14' P15' P16' P17'*) *p' p* ** Γ*1'*)
R (*case-tuple17 CP p*) (*case-tuple17*$(λ$_2$*a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17. f' a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17*)$*p'*)› (**is** ‹*?G* Γ›)
unfolding *autoref-tag-defs PROTECT2-def*
apply1 (*rule hn-refine-cons-pre*[*OF FR*])
apply1 (*cases p*; *cases p'*; *simp add*: *tuple17-assn-conv*[*THEN tuple17-assn-ctxt*])
unfolding *CP-SPLIT-def prod.simps*
apply (*rule hn-refine-cons*[*OF - Pair - entails-refl*])
applyS (*simp add*: *hn-ctxt-def*)
applyS *simp* **using** *FR2*
by (*simp add*: *hn-ctxt-def*)

lemma *case-tuple17-arity*[*sepref-monadify-arity*]:
‹*case-tuple17* ≡ λ$_2$*fp p. SP case-tuple17*$(λ$_2$*a b. fp*$*a*$*b*)$*p*›
by (*simp-all only*: *SP-def APP-def PROTECT2-def RCALL-def*)

lemma *case-tuple17-comb*[*sepref-monadify-comb*]:
‹⋀*fp p. case-tuple17*$*fp*$*p* ≡ *Refine-Basic.bind*$(*EVAL*$*p*)$(λ$_2$*p.* (*SP case-tuple17*$*fp*$*p*))›
by (*simp-all*)

lemma *case-tuple17-plain-comb*[*sepref-monadify-comb*]:
EVAL$(*case-tuple17*$(λ$_2$*a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17. fp a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17*)$*p*) ≡
Refine-Basic.bind$(*EVAL*$*p*)$(λ$_2$*p. case-tuple17*$(λ$_2$*a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17. EVAL*$(*fp a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17*))$*p*)
apply (*rule eq-reflection*, *simp split*: *list.split prod.split option.split tuple17.split*)+
done

lemma *ho-tuple17-move*[*sepref-preproc*]: ‹*case-tuple17* (λ*a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17 x. f x a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17*) =
(λ*p x. case-tuple17* (*f x*) *p*)›
by (*auto split*: *tuple17.splits*)

locale *isasat-state* =
isasat-state-ops a-assn b-assn c-assn d-assn e-assn
f-assn g-assn h-assn i-assn j-assn

k-assn l-assn m-assn n-assn o-assn p-assn q-assn
a-default a
b-default b
c-default c
d-default d
e-default e
f-default f
g-default g
h-default h
i-default i
j-default j
k-default k
l-default l
m-default m
n-default n
ko-default ko
p-default p
q-default q
for
a-assn :: ‹*'a* ⇒ *'xa*:: *llvm-rep* ⇒ *assn*› **and**
b-assn :: ‹*'b* ⇒ *'xb*:: *llvm-rep* ⇒ *assn*› **and**
c-assn :: ‹*'c* ⇒ *'xc*:: *llvm-rep* ⇒ *assn*› **and**
d-assn :: ‹*'d* ⇒ *'xd*:: *llvm-rep* ⇒ *assn*› **and**
e-assn :: ‹*'e* ⇒ *'xe*:: *llvm-rep* ⇒ *assn*› **and**
f-assn :: ‹*'f* ⇒ *'xf*:: *llvm-rep* ⇒ *assn*› **and**
g-assn :: ‹*'g* ⇒ *'xg*:: *llvm-rep* ⇒ *assn*› **and**
h-assn :: ‹*'h* ⇒ *'xh*:: *llvm-rep* ⇒ *assn*› **and**
i-assn :: ‹*'i* ⇒ *'xi*:: *llvm-rep* ⇒ *assn*› **and**
j-assn :: ‹*'j* ⇒ *'xj*:: *llvm-rep* ⇒ *assn*› **and**
k-assn :: ‹*'k* ⇒ *'xk*:: *llvm-rep* ⇒ *assn*› **and**
l-assn :: ‹*'l* ⇒ *'xl*:: *llvm-rep* ⇒ *assn*› **and**
m-assn :: ‹*'m* ⇒ *'xm*:: *llvm-rep* ⇒ *assn*› **and**
n-assn :: ‹*'n* ⇒ *'xn*:: *llvm-rep* ⇒ *assn*› **and**
o-assn :: ‹*'o* ⇒ *'xo*:: *llvm-rep* ⇒ *assn*› **and**
p-assn :: ‹*'p* ⇒ *'xp*:: *llvm-rep* ⇒ *assn*› **and**
q-assn :: ‹*'q* ⇒ *'xq*:: *llvm-rep* ⇒ *assn*› **and**
a-default :: *'a* **and**
a :: ‹*'xa llM*› **and**
b-default :: *'b* **and**
b :: ‹*'xb llM*› **and**
c-default :: *'c* **and**
c :: ‹*'xc llM*› **and**
d-default :: *'d* **and**
d :: ‹*'xd llM*› **and**
e-default :: *'e* **and**
e :: ‹*'xe llM*› **and**
f-default :: *'f* **and**
f :: ‹*'xf llM*› **and**
g-default :: *'g* **and**
g :: ‹*'xg llM*› **and**
h-default :: *'h* **and**
h :: ‹*'xh llM*› **and**
i-default :: *'i* **and**
i :: ‹*'xi llM*› **and**
j-default :: *'j* **and**
j :: ‹*'xj llM*› **and**

k-default :: *'k* **and**
k :: ‹*'xk llM*› **and**
l-default :: *'l* **and**
l :: ‹*'xl llM*› **and**
m-default :: *'m* **and**
m :: ‹*'xm llM*› **and**
n-default :: *'n* **and**
n :: ‹*'xn llM*› **and**
ko-default :: *'o* **and**
ko :: ‹*'xo llM*› **and**
p-default :: *'p* **and**
p :: ‹*'xp llM*› **and**
q-default :: *'q* **and**
q :: ‹*'xq llM*› **and**
a-free :: ‹*'xa* ⇒ *unit llM*› **and**
b-free :: ‹*'xb* ⇒ *unit llM*› **and**
c-free :: ‹*'xc* ⇒ *unit llM*› **and**
d-free :: ‹*'xd* ⇒ *unit llM*› **and**
e-free :: ‹*'xe* ⇒ *unit llM*› **and**
f-free :: ‹*'xf* ⇒ *unit llM*› **and**
g-free :: ‹*'xg* ⇒ *unit llM*› **and**
h-free :: ‹*'xh* ⇒ *unit llM*› **and**
i-free :: ‹*'xi* ⇒ *unit llM*› **and**
j-free :: ‹*'xj* ⇒ *unit llM*› **and**
k-free :: ‹*'xk* ⇒ *unit llM*› **and**
l-free :: ‹*'xl* ⇒ *unit llM*› **and**
m-free :: ‹*'xm* ⇒ *unit llM*› **and**
n-free :: ‹*'xn* ⇒ *unit llM*› **and**
o-free :: ‹*'xo* ⇒ *unit llM*› **and**
p-free :: ‹*'xp* ⇒ *unit llM*› **and**
q-free :: ‹*'xq* ⇒ *unit llM*› +
assumes
a: ‹(*uncurry0 a*, *uncurry0* (*RETURN a-default*)) ∈ *unit-assn*^{k} →$_a$ *a-assn*› **and**
b: ‹(*uncurry0 b*, *uncurry0* (*RETURN b-default*)) ∈ *unit-assn*^{k} →$_a$ *b-assn*› **and**
c: ‹(*uncurry0 c*, *uncurry0* (*RETURN c-default*)) ∈ *unit-assn*^{k} →$_a$ *c-assn*› **and**
d: ‹(*uncurry0 d*, *uncurry0* (*RETURN d-default*)) ∈ *unit-assn*^{k} →$_a$ *d-assn*› **and**
e: ‹(*uncurry0 e*, *uncurry0* (*RETURN e-default*)) ∈ *unit-assn*^{k} →$_a$ *e-assn*› **and**
f: ‹(*uncurry0 f*, *uncurry0* (*RETURN f-default*)) ∈ *unit-assn*^{k} →$_a$ *f-assn*› **and**
g: ‹(*uncurry0 g*, *uncurry0* (*RETURN g-default*)) ∈ *unit-assn*^{k} →$_a$ *g-assn*› **and**
h: ‹(*uncurry0 h*, *uncurry0* (*RETURN h-default*)) ∈ *unit-assn*^{k} →$_a$ *h-assn*› **and**
i: ‹(*uncurry0 i*, *uncurry0* (*RETURN i-default*)) ∈ *unit-assn*^{k} →$_a$ *i-assn*› **and**
j: ‹(*uncurry0 j*, *uncurry0* (*RETURN j-default*)) ∈ *unit-assn*^{k} →$_a$ *j-assn*› **and**
k: ‹(*uncurry0 k*, *uncurry0* (*RETURN k-default*)) ∈ *unit-assn*^{k} →$_a$ *k-assn*› **and**
l: ‹(*uncurry0 l*, *uncurry0* (*RETURN l-default*)) ∈ *unit-assn*^{k} →$_a$ *l-assn*› **and**
m: ‹(*uncurry0 m*, *uncurry0* (*RETURN m-default*)) ∈ *unit-assn*^{k} →$_a$ *m-assn*› **and**
n: ‹(*uncurry0 n*, *uncurry0* (*RETURN n-default*)) ∈ *unit-assn*^{k} →$_a$ *n-assn*› **and**
o: ‹(*uncurry0 ko*, *uncurry0* (*RETURN ko-default*)) ∈ *unit-assn*^{k} →$_a$ *o-assn*›**and**
p: ‹(*uncurry0 p*, *uncurry0* (*RETURN p-default*)) ∈ *unit-assn*^{k} →$_a$ *p-assn*› **and**
q: ‹(*uncurry0 q*, *uncurry0* (*RETURN q-default*)) ∈ *unit-assn*^{k} →$_a$ *q-assn*› **and**
a-free: ‹*MK-FREE a-assn a-free*› **and**
b-free: ‹*MK-FREE b-assn b-free*›**and**
c-free: ‹*MK-FREE c-assn c-free*›**and**
d-free: ‹*MK-FREE d-assn d-free*›**and**
e-free: ‹*MK-FREE e-assn e-free*›**and**
f-free: ‹*MK-FREE f-assn f-free*›**and**
g-free: ‹*MK-FREE g-assn g-free*›**and**

h-free: ‹*MK-FREE h-assn h-free*›**and**
i-free: ‹*MK-FREE i-assn i-free*›**and**
j-free: ‹*MK-FREE j-assn j-free*›**and**
k-free: ‹*MK-FREE k-assn k-free*›**and**
l-free: ‹*MK-FREE l-assn l-free*›**and**
m-free: ‹*MK-FREE m-assn m-free*›**and**
n-free: ‹*MK-FREE n-assn n-free*›**and**
o-free: ‹*MK-FREE o-assn o-free*›**and**
p-free: ‹*MK-FREE p-assn p-free*›**and**
q-free: ‹*MK-FREE q-assn q-free*›
notes [[*sepref-register-adhoc a-default b-default c-default d-default e-default f-default g-default h-default i-default j-default k-default l-default m-default n-default ko-default p-default q-default*]]
begin

lemmas [*sepref-comb-rules*] = *hn-case-tuple17'*[*of* - *a-assn b-assn c-assn d-assn e-assn f-assn g-assn h-assn i-assn j-assn k-assn l-assn m-assn n-assn o-assn p-assn q-assn*, *unfolded isasat-assn-def*[*symmetric*]]

lemma *ex-tuple17-iff*: (∃ *b* :: (-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-,-)*tuple17*. *P b*) ⟷
(∃ *a b c d e f g h i j k l m n ko p q*. *P* (*Tuple17 a b c d e f g h i j k l m n ko p q*))
apply *auto*
apply (*case-tac b*)
by *force*

lemmas [*sepref-frame-free-rules*] = *a-free b-free c-free d-free e-free f-free g-free h-free i-free j-free k-free l-free m-free n-free o-free p-free q-free*
sepref-register
‹*Tuple17*›
lemma [*sepref-fr-rules*]: ‹(*uncurry16* (*Mreturn* o_{17} *Tuple17*), *uncurry16* (*RETURN* o_{17} *Tuple17*))
∈ $a\text{-}assn^d$ $*_a$ $b\text{-}assn^d$ $*_a$ $c\text{-}assn^d$ $*_a$ $d\text{-}assn^d$ $*_a$
$e\text{-}assn^d$ $*_a$ $f\text{-}assn^d$ $*_a$ $g\text{-}assn^d$ $*_a$ $h\text{-}assn^d$ $*_a$
$i\text{-}assn^d$ $*_a$ $j\text{-}assn^d$ $*_a$ $k\text{-}assn^d$ $*_a$ $l\text{-}assn^d$ $*_a$
$m\text{-}assn^d$ $*_a$ $n\text{-}assn^d$ $*_a$ $o\text{-}assn^d$ $*_a$ $p\text{-}assn^d$ $*_a$
$q\text{-}assn^d$
$\rightarrow_a$ *isasat-assn*›
unfolding *isasat-assn-def*
apply *sepref-to-hoare*
apply *vcg*
unfolding *ex-tuple17-iff*
apply *vcg'*
done

lemma [*sepref-frame-match-rules*]:
‹ *hn-ctxt*
(*tuple17-assn* (*invalid-assn a-assn*) (*invalid-assn b-assn*) (*invalid-assn c-assn*) (*invalid-assn d-assn*) (*invalid-assn e-assn*)
(*invalid-assn f-assn*) (*invalid-assn g-assn*) (*invalid-assn h-assn*) (*invalid-assn i-assn*) (*invalid-assn j-assn*) (*invalid-assn k-assn*)
(*invalid-assn l-assn*) (*invalid-assn m-assn*) (*invalid-assn n-assn*) (*invalid-assn o-assn*) (*invalid-assn p-assn*) (*invalid-assn q-assn*)) *ax bx* ⊢ *hn-val UNIV ax bx*›
unfolding *hn-ctxt-def invalid-assn-def isasat-assn-def entails-def*
apply (*auto split*: *prod.split tuple17.splits elim*: *is-pureE*
simp: *sep-algebra-simps pure-part-pure-conj-eq*)
apply (*auto simp*: *pure-part-def*)
apply (*auto simp*: *pure-def pure-true-conv*)
done

lemma *RETURN-case-tuple17-inverse*: ‹*RETURN*
(*let* - = *M*
in ff) =
(*do* {- ← *mop-free M*;
RETURN (*ff*)})›
by (*auto intro*!: *ext simp*: *mop-free-def split*: *tuple17.splits*)

sepref-def *update-a-code*
is ‹*uncurry* (*RETURN oo update-a*)›
:: ‹*a-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=5*]]
unfolding *update-a-def tuple17.case-distrib comp-def RETURN-case-tuple17-inverse*
by *sepref*

sepref-def *update-b-code*
is ‹*uncurry* (*RETURN oo update-b*)›
:: ‹*b-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-b-def tuple17.case-distrib comp-def RETURN-case-tuple17-inverse*
by *sepref*

sepref-def *update-c-code*
is ‹*uncurry* (*RETURN oo update-c*)›
:: ‹*c-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-c-def tuple17.case-distrib comp-def RETURN-case-tuple17-inverse*
by *sepref*

sepref-def *update-d-code*
is ‹*uncurry* (*RETURN oo update-d*)›
:: ‹*d-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-d-def tuple17.case-distrib comp-def RETURN-case-tuple17-inverse*
by *sepref*

sepref-def *update-e-code*
is ‹*uncurry* (*RETURN oo update-e*)›
:: ‹*e-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-e-def tuple17.case-distrib comp-def RETURN-case-tuple17-inverse*
by *sepref*

sepref-def *update-f-code*
is ‹*uncurry* (*RETURN oo update-f*)›
:: ‹*f-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-f-def tuple17.case-distrib comp-def RETURN-case-tuple17-inverse*
by *sepref*

sepref-def *update-g-code*
is ‹*uncurry* (*RETURN oo update-g*)›
:: ‹*g-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-g-def tuple17.case-distrib comp-def RETURN-case-tuple17-inverse*
by *sepref*

sepref-def *update-h-code*
is ‹*uncurry* (*RETURN oo update-h*)›
:: ‹*h-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-h-def tuple17.case-distrib comp-def RETURN-case-tuple17-inverse*
by *sepref*

sepref-def *update-i-code*
is ‹*uncurry* (*RETURN oo update-i*)›
:: ‹*i-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-i-def tuple17.case-distrib comp-def RETURN-case-tuple17-inverse*
by *sepref*

sepref-def *update-j-code*
is ‹*uncurry* (*RETURN oo update-j*)›
:: ‹*j-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-j-def tuple17.case-distrib comp-def RETURN-case-tuple17-inverse*
by *sepref*

sepref-def *update-k-code*
is ‹*uncurry* (*RETURN oo update-k*)›
:: ‹*k-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-k-def tuple17.case-distrib comp-def RETURN-case-tuple17-inverse*
by *sepref*

sepref-def *update-l-code*
is ‹*uncurry* (*RETURN oo update-l*)›
:: ‹*l-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-l-def tuple17.case-distrib comp-def RETURN-case-tuple17-inverse*
by *sepref*

sepref-def *update-m-code*
is ‹*uncurry* (*RETURN oo update-m*)›
:: ‹*m-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-m-def tuple17.case-distrib comp-def RETURN-case-tuple17-inverse*
by *sepref*

sepref-def *update-n-code*
is ‹*uncurry* (*RETURN oo update-n*)›
:: ‹*n-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-n-def tuple17.case-distrib comp-def RETURN-case-tuple17-inverse*
by *sepref*

sepref-def *update-o-code*
is ‹*uncurry* (*RETURN oo update-o*)›
:: ‹*o-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-o-def tuple17.case-distrib comp-def RETURN-case-tuple17-inverse*
by *sepref*

sepref-def *update-p-code*
is ‹*uncurry* (*RETURN oo update-p*)›
:: ‹$p\text{-}assn^d *_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-p-def tuple17.case-distrib comp-def RETURN-case-tuple17-inverse*
by *sepref*

sepref-def *update-q-code*
is ‹*uncurry* (*RETURN oo update-q*)›
:: ‹$q\text{-}assn^d *_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
supply [[*goals-limit=1*]]
unfolding *update-q-def tuple17.case-distrib comp-def RETURN-case-tuple17-inverse*
by *sepref*

method *stuff-pre* =
sepref-to-hoare;
case-tac x;
vcg;
unfold wpa-return;
subst (*asm*)(*2*) *sep-algebra-class.sep-conj-empty'*[*symmetric*];
rule apply-htriple-rule

method *stuff-post1* =
rule POSTCONDI;
rule STATE-monoI

method *stuff-post2* =
unfold ex-tuple17-iff entails-def;
auto simp: *Exists-eq-simp ex-tuple17-iff entails-def entails-eq-iff pure-true-conv sep-conj-left-commute*;
smt (*z3*) *entails-def entails-eq-iff pure-true-conv sep-conj-aci*(*4*) *sep-conj-aci*(*5*) *sep-conj-left-commute*

lemma *RETURN-case-tuple17-invers*: ‹(*RETURN* ∘∘ *case-tuple17*)
(λ*x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17*.
ff x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17) =
case-tuple17
(λ*x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17*.
RETURN (*ff x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17*))›
by (*auto intro*!: *ext split*: *tuple17.splits*)

lemmas [*sepref-fr-rules*] = *a b c d e f g h i j k l m n o p q*

sepref-definition *remove-a-code*
is ‹*RETURN o remove-a*›
:: ‹ $isasat\text{-}assn^d \rightarrow_a a\text{-}assn \times_a isasat\text{-}assn$›
unfolding *remove-a-def RETURN-case-tuple17-invers*
by *sepref*

sepref-definition *remove-b-code*
is ‹*RETURN o remove-b*›
:: ‹ $isasat\text{-}assn^d \rightarrow_a b\text{-}assn \times_a isasat\text{-}assn$›
unfolding *remove-b-def RETURN-case-tuple17-invers*
by *sepref*

sepref-definition *remove-c-code*
is ‹*RETURN o remove-c*›

:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *c-assn* $\times_a$ *isasat-assn*›
unfolding *remove-c-def RETURN-case-tuple17-invers*
by *sepref*

sepref-definition *remove-d-code*
is ‹*RETURN o remove-d*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *d-assn* $\times_a$ *isasat-assn*›
unfolding *remove-d-def RETURN-case-tuple17-invers*
by *sepref*

sepref-definition *remove-e-code*
is ‹*RETURN o remove-e*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *e-assn* $\times_a$ *isasat-assn*›
unfolding *remove-e-def RETURN-case-tuple17-invers*
by *sepref*

sepref-definition *remove-f-code*
is ‹*RETURN o remove-f*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *f-assn* $\times_a$ *isasat-assn*›
unfolding *remove-f-def RETURN-case-tuple17-invers*
by *sepref*

sepref-definition *remove-g-code*
is ‹*RETURN o remove-g*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *g-assn* $\times_a$ *isasat-assn*›
unfolding *remove-g-def RETURN-case-tuple17-invers*
by *sepref*

sepref-definition *remove-h-code*
is ‹*RETURN o remove-h*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *h-assn* $\times_a$ *isasat-assn*›
unfolding *remove-h-def RETURN-case-tuple17-invers*
by *sepref*

sepref-definition *remove-i-code*
is ‹*RETURN o remove-i*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *i-assn* $\times_a$ *isasat-assn*›
unfolding *remove-i-def RETURN-case-tuple17-invers*
by *sepref*

sepref-definition *remove-j-code*
is ‹*RETURN o remove-j*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *j-assn* $\times_a$ *isasat-assn*›
unfolding *remove-j-def RETURN-case-tuple17-invers*
by *sepref*

sepref-definition *remove-k-code*
is ‹*RETURN o remove-k*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *k-assn* $\times_a$ *isasat-assn*›
unfolding *remove-k-def RETURN-case-tuple17-invers*
by *sepref*

sepref-definition *remove-l-code*
is ‹*RETURN o remove-l*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *l-assn* $\times_a$ *isasat-assn*›
unfolding *remove-l-def RETURN-case-tuple17-invers*

by *sepref*

sepref-definition *remove-m-code*
is ‹*RETURN o remove-m*›
:: ‹ *isasat-assn*^{d} →$_a$ *m-assn* ×$_a$ *isasat-assn*›
unfolding *remove-m-def RETURN-case-tuple17-invers*
by *sepref*

sepref-definition *remove-n-code*
is ‹*RETURN o remove-n*›
:: ‹ *isasat-assn*^{d} →$_a$ *n-assn* ×$_a$ *isasat-assn*›
unfolding *remove-n-def RETURN-case-tuple17-invers*
by *sepref*

sepref-definition *remove-o-code*
is ‹*RETURN o remove-o*›
:: ‹ *isasat-assn*^{d} →$_a$ *o-assn* ×$_a$ *isasat-assn*›
unfolding *remove-o-def RETURN-case-tuple17-invers*
by *sepref*

sepref-definition *remove-p-code*
is ‹*RETURN o remove-p*›
:: ‹ *isasat-assn*^{d} →$_a$ *p-assn* ×$_a$ *isasat-assn*›
unfolding *remove-p-def RETURN-case-tuple17-invers*
by *sepref*

sepref-definition *remove-q-code*
is ‹*RETURN o remove-q*›
:: ‹ *isasat-assn*^{d} →$_a$ *q-assn* ×$_a$ *isasat-assn*›
unfolding *remove-q-def RETURN-case-tuple17-invers*
by *sepref*

lemma *remove-a-code-alt-def*: ‹*remove-a-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 0*;
x ← *a*;
x ← *ll-insert-value xi x 0*;
return$_M$ (*M*, *x*)
}› **and**
remove-b-code-alt-def: ‹*remove-b-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 1*;
x ← *b*;
x ← *ll-insert-value xi x 1*;
return$_M$ (*M*, *x*)
}›**and**
remove-c-code-alt-def: ‹*remove-c-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 2*;
x ← *c*;
x ← *ll-insert-value xi x 2*;
return$_M$ (*M*, *x*)
}›**and**
remove-d-code-alt-def: ‹*remove-d-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 3*;
x ← *d*;
x ← *ll-insert-value xi x 3*;
return$_M$ (*M*, *x*)

}›**and**
remove-e-code-alt-def: ‹*remove-e-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 4*;
x ← *e*;
x ← *ll-insert-value xi x 4*;
return$_M$ (*M*, *x*)
}›**and**
remove-f-code-alt-def: ‹*remove-f-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 5*;
x ← *f*;
x ← *ll-insert-value xi x 5*;
return$_M$ (*M*, *x*)
}›**and**
remove-g-code-alt-def: ‹*remove-g-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 6*;
x ← *g*;
x ← *ll-insert-value xi x 6*;
return$_M$ (*M*, *x*)
}›**and**
remove-h-code-alt-def: ‹*remove-h-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 7*;
x ← *h*;
x ← *ll-insert-value xi x 7*;
return$_M$ (*M*, *x*)
}›**and**
remove-i-code-alt-def: ‹*remove-i-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 8*;
x ← *i*;
x ← *ll-insert-value xi x 8*;
return$_M$ (*M*, *x*)
}›**and**
remove-j-code-alt-def: ‹*remove-j-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 9*;
x ← *j*;
x ← *ll-insert-value xi x 9*;
return$_M$ (*M*, *x*)
}›**and**
remove-k-code-alt-def: ‹*remove-k-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 10*;
x ← *k*;
x ← *ll-insert-value xi x 10*;
return$_M$ (*M*, *x*)
}›**and**
remove-l-code-alt-def: ‹*remove-l-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 11*;
x ← *l*;
x ← *ll-insert-value xi x 11*;
return$_M$ (*M*, *x*)
}›**and**
remove-m-code-alt-def: ‹*remove-m-code xi* = *do*$_M$ {
M ← *ll-extract-value xi 12*;
x ← *m*;
x ← *ll-insert-value xi x 12*;
return$_M$ (*M*, *x*)
}›**and**
remove-n-code-alt-def: ‹*remove-n-code xi* = *do*$_M$ {

```
        M ← ll-extract-value xi 13;
        x ← n;
        x ← ll-insert-value xi x 13;
        return_M (M, x)
  }›and
  remove-o-code-alt-def: ‹remove-o-code xi = do_M {
        M ← ll-extract-value xi 14;
        x ← ko;
        x ← ll-insert-value xi x 14;
        return_M (M, x)
  }›and
  remove-p-code-alt-def: ‹remove-p-code xi = do_M {
        M ← ll-extract-value xi 15;
        x ← p;
        x ← ll-insert-value xi x 15;
        return_M (M, x)
  }›and
  remove-q-code-alt-def: ‹remove-q-code xi = do_M {
        M ← ll-extract-value xi 16;
        x ← q;
        x ← ll-insert-value xi x 16;
        return_M (M, x)
  }›
  supply [simp] = llvm-extract-value-def
    ll-extract-value-def to-val-tuple17-def
    checked-from-val-def ll-insert-value-def
    llvm-insert-value-def
  unfolding remove-a-code-def remove-b-code-def inline-direct-return-node-case
    remove-c-code-def remove-d-code-def remove-e-code-def remove-f-code-def remove-g-code-def
    remove-h-code-def remove-i-code-def remove-j-code-def remove-k-code-def remove-l-code-def
    remove-m-code-def remove-n-code-def remove-o-code-def remove-p-code-def remove-q-code-def
  by (solves ‹cases xi, rewrite in ‹Mreturn (-, ⌑)› llvm-rep-class.from-to-id'[symmetric],
    simp flip: from-val-tuple17-def›)+

lemma update-a-code-alt-def: ‹⋀x. update-a-code x xi = do_M {
        M ← ll-extract-value xi 0; a-free M;
        x ← ll-insert-value xi x 0;
        return_M (x)
  }› and
  update-b-code-alt-def: ‹⋀x. update-b-code x xi = do_M {
        M ← ll-extract-value xi 1; b-free M;
        x ← ll-insert-value xi x 1;
        return_M (x)
  }›and
  update-c-code-alt-def: ‹⋀x. update-c-code x xi = do_M {
        M ← ll-extract-value xi 2; c-free M;
        x ← ll-insert-value xi x 2;
        return_M (x)
  }›and
  update-d-code-alt-def: ‹⋀x. update-d-code x xi = do_M {
        M ← ll-extract-value xi 3; d-free M;
        x ← ll-insert-value xi x 3;
        return_M (x)
  }›and
  update-e-code-alt-def: ‹⋀x. update-e-code x xi = do_M {
        M ← ll-extract-value xi 4; e-free M;
```

```
        x ← ll-insert-value xi x 4;
        return_M (x)
}›and
update-f-code-alt-def: ‹⋀x. update-f-code x xi = do_M {
        M ← ll-extract-value xi 5; f-free M;
        x ← ll-insert-value xi x 5;
        return_M (x)
}›and
update-g-code-alt-def: ‹⋀x. update-g-code x xi = do_M {
        M ← ll-extract-value xi 6; g-free M;
        x ← ll-insert-value xi x 6;
        return_M (x)
}›and
update-h-code-alt-def: ‹⋀x. update-h-code x xi = do_M {
        M ← ll-extract-value xi 7; h-free M;
        x ← ll-insert-value xi x 7;
        return_M (x)
}›and
update-i-code-alt-def: ‹⋀x. update-i-code x xi = do_M {
        M ← ll-extract-value xi 8; i-free M;
        x ← ll-insert-value xi x 8;
        return_M (x)
}›and
update-j-code-alt-def: ‹⋀x. update-j-code x xi = do_M {
        M ← ll-extract-value xi 9; j-free M;
        x ← ll-insert-value xi x 9;
        return_M (x)
}›and
update-k-code-alt-def: ‹⋀x. update-k-code x xi = do_M {
        M ← ll-extract-value xi 10; k-free M;
        x ← ll-insert-value xi x 10;
        return_M (x)
}›and
update-l-code-alt-def: ‹⋀x. update-l-code x xi = do_M {
        M ← ll-extract-value xi 11; l-free M;
        x ← ll-insert-value xi x 11;
        return_M (x)
}›and
update-m-code-alt-def: ‹⋀x. update-m-code x xi = do_M {
        M ← ll-extract-value xi 12; m-free M;
        x ← ll-insert-value xi x 12;
        return_M (x)
}›and
update-n-code-alt-def: ‹⋀x. update-n-code x xi = do_M {
        M ← ll-extract-value xi 13; n-free M;
        x ← ll-insert-value xi x 13;
        return_M (x)
}›and
update-o-code-alt-def: ‹⋀x. update-o-code x xi = do_M {
        M ← ll-extract-value xi 14; o-free M;
        x ← ll-insert-value xi x 14;
        return_M (x)
}›and
update-p-code-alt-def: ‹⋀x. update-p-code x xi = do_M {
        M ← ll-extract-value xi 15; p-free M;
        x ← ll-insert-value xi x 15;
```

```
        return_M (x)
  }›and
  update-q-code-alt-def: ‹⋀x. update-q-code x xi = do_M {
        M ← ll-extract-value xi 16; q-free M;
        x ← ll-insert-value xi x 16;
        return_M (x)
  }›
  supply [simp] = llvm-extract-value-def
    ll-extract-value-def to-val-tuple17-def
    checked-from-val-def ll-insert-value-def
    llvm-insert-value-def
  unfolding update-a-code-def update-b-code-def inline-direct-return-node-case
    update-c-code-def update-d-code-def update-e-code-def update-f-code-def update-g-code-def
    update-h-code-def update-i-code-def update-j-code-def update-k-code-def update-l-code-def
    update-m-code-def update-n-code-def update-o-code-def update-p-code-def update-q-code-def
    comp-def
  by (solves ‹cases xi, rewrite in ‹Mreturn (¤)› llvm-rep-class.from-to-id'[symmetric],
    simp flip: from-val-tuple17-def›)+

end


context isasat-state
begin
lemma reconstruct-isasat[sepref-frame-match-rules]:
  ‹hn-ctxt
     (tuple17-assn (a-assn) (b-assn) (c-assn) (d-assn) (e-assn)
    (f-assn) (g-assn) (h-assn) (i-assn) (j-assn) (k-assn)
   (l-assn) (m-assn) (n-assn) (o-assn) (p-assn) q-assn) ax bx ⊢ hn-ctxt isasat-assn ax bx›
    unfolding isasat-assn-def
    apply (auto split: prod.split tuple17.splits elim: is-pureE
      simp: sep-algebra-simps pure-part-pure-conj-eq)
      done

context
  fixes x-assn :: ‹'r ⇒ 'qst ⇒ assn› and
    read-all-code :: ‹'xa ⇒ 'xb ⇒ 'xc ⇒ 'xd ⇒ 'xe ⇒ 'xf ⇒ 'xg ⇒ 'xh ⇒ 'xi ⇒ 'xj ⇒ 'xk ⇒ 'xl ⇒ 'xm
⇒ 'xn ⇒ 'xo ⇒ 'xp ⇒ 'xq ⇒ 'qst llM› and
    read-all :: ‹'a ⇒ 'b ⇒ 'c ⇒ 'd ⇒ 'e ⇒ 'f ⇒ 'g ⇒ 'h ⇒ 'i ⇒ 'j ⇒ 'k ⇒ 'l ⇒ 'm ⇒ 'n ⇒ 'o ⇒ 'p
⇒ 'q ⇒ 'r nres›
begin

definition read-all-st-code :: ‹-› where
  ‹read-all-st-code xi = (case xi of
  Tuple17 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17 ⇒
    read-all-code a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17)›

definition read-all-st :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
     'k, 'l, 'm, 'n, 'o, 'p, 'q) tuple17 ⇒ -› where
  ‹read-all-st tuple17 = (case tuple17 of Tuple17 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15
a16 a17 ⇒
  read-all a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17)›

context
  fixes P
  assumes trail-read[sepref-fr-rules]: ‹(uncurry16 read-all-code, uncurry16 read-all) ∈
```

$[uncurry16\ P]_a\ a\text{-}assn^k *_a b\text{-}assn^k *_a c\text{-}assn^k *_a d\text{-}assn^k *_a e\text{-}assn^k *_a f\text{-}assn^k *_a$
$g\text{-}assn^k *_a h\text{-}assn^k *_a i\text{-}assn^k *_a j\text{-}assn^k *_a k\text{-}assn^k *_a l\text{-}assn^k *_a$
$m\text{-}assn^k *_a n\text{-}assn^k *_a o\text{-}assn^k *_a p\text{-}assn^k *_a q\text{-}assn^k \to x\text{-}assn\rangle$

notes [[*sepref-register-adhoc read-all*]]

begin

sepref-definition *read-all-code-tmp*
is *read-all-st*
:: $\langle[case\text{-}tuple17\ P]_a\ isasat\text{-}assn^k \to x\text{-}assn\rangle$
unfolding *read-all-st-def*
by *sepref*

lemmas *read-all-code-refine* =
read-all-code-tmp.refine[*unfolded read-all-code-tmp-def*
read-all-st-code-def[*symmetric*]]

end

end

lemmas [*unfolded Let-def*, *tuple17-getters-setters*] =
update-a-def
update-b-def
update-c-def
update-d-def
update-e-def
update-f-def
update-g-def
update-h-def
update-i-def
update-j-def
update-k-def
update-l-def
update-m-def
update-n-def
update-o-def
update-p-def
update-q-def

remove-a-def
remove-b-def
remove-c-def
remove-d-def
remove-e-def
remove-f-def
remove-g-def
remove-h-def
remove-i-def
remove-j-def
remove-k-def
remove-l-def
remove-m-def
remove-n-def
remove-o-def
remove-p-def
remove-q-def

end

lemmas [*tuple17-getters-setters*] =
isasat-state-ops.remove-a-def
isasat-state-ops.remove-b-def
isasat-state-ops.remove-c-def
isasat-state-ops.remove-d-def
isasat-state-ops.remove-e-def
isasat-state-ops.remove-f-def
isasat-state-ops.remove-g-def
isasat-state-ops.remove-h-def
isasat-state-ops.remove-i-def
isasat-state-ops.remove-j-def
isasat-state-ops.remove-k-def
isasat-state-ops.remove-l-def
isasat-state-ops.remove-m-def
isasat-state-ops.remove-n-def
isasat-state-ops.remove-o-def
isasat-state-ops.remove-p-def
isasat-state-ops.remove-q-def

end
theory *IsaSAT-Setup0-LLVM*
imports *IsaSAT-Watch-List-LLVM IsaSAT-Lookup-Conflict-LLVM*
More-Sepref.WB-More-Refinement IsaSAT-Clauses-LLVM LBD-LLVM
IsaSAT-Options-LLVM IsaSAT-VMTF-Setup-LLVM
IsaSAT-Arena-Sorting-LLVM
IsaSAT-Rephase-LLVM
IsaSAT-EMA-LLVM
IsaSAT-Stats-LLVM
IsaSAT-VDom-LLVM
IsaSAT-Bump-Heuristics-State-LLVM
Isabelle-LLVM.LLVM-DS-Block-Alloc
Tuple17-LLVM
begin

hide-const (**open**) *NEMonad.ASSERT NEMonad.RETURN*

This is the setup for accessing and modifying the state. The construction is kept generic (even if still targetting only our state). There is a lot of copy-paste that would be nice to automate at some point.

We define 3 sort of operations:

1. extracting an element, replacing it by an default element. Modifies the state. The name starts with *exctr*
2. reinserting an element, freeing the current one. Modifies the state. The name starts with *update*
3. in-place reading a value, possibly with pure parameters. Does not modify the state. The name starts with *read*

type-synonym *occs-assn* = ‹(*64*,(*64 word*),*64*)*array-array-list*›

abbreviation ‹*occs-assn* ≡ *aal-assn′ TYPE*(*64*) *TYPE*(*64*) *sint64-nat-assn*›

type-synonym *twl-st-wll-trail-fast2* =
‹(*trail-pol-fast-assn*, *arena-assn*, *option-lookup-clause-assn*,
64 word, *watched-wl-uint32*, *bump-heuristics-assn*,
32 word, *cach-refinement-l-assn*, *lbd-assn*, *out-learned-assn*,
isasat-stats-assn, *heur-assn*,
aivdom-assn, (*64 word* × *64 word* × *64 word* × *64 word* × *64 word*),
opts-assn, *arena-assn*, *occs-assn*) *tuple17*›

definition *isasat-bounded-assn* :: ‹*isasat* ⇒ *twl-st-wll-trail-fast2* ⇒ *assn*› **where**
‹*isasat-bounded-assn* = *tuple17-assn*
trail-pol-fast-assn arena-fast-assn
conflict-option-rel-assn
sint64-nat-assn
watchlist-fast-assn
heuristic-bump-assn
uint32-nat-assn
cach-refinement-l-assn
lbd-assn
out-learned-assn
isasat-stats-assn
heuristic-assn
aivdom-assn
lcount-assn
opts-assn
arena-fast-assn
occs-assn›

sepref-register *mop-arena-length*

type-synonym *twl-st-wll-trail-fast* =
‹*trail-pol-fast-assn* × *arena-assn* × *option-lookup-clause-assn* ×
64 word × *watched-wl-uint32* × *bump-heuristics-assn* ×
32 word × *cach-refinement-l-assn* × *lbd-assn* × *out-learned-assn* × *isasat-stats* ×
heur-assn ×
aivdom-assn × (*64 word* × *64 word* × *64 word* × *64 word* × *64 word*) ×
opts-assn × *arena-assn* × *occs-assn*›

The following constants are not useful for the initialisation for the solver, but only as temporary replacement for values in state.

definition *bottom-trail* :: *trail-pol* **where**
‹*bottom-trail* = *do* {
let M0 = [] *in*
let cs = [] *in*
let M = *replicate 0 UNSET in*
let M' = *replicate 0 0 in*
let M'' = *replicate 0 1 in*
((*M0*, *M*, *M'*, *M''*, *0*, *cs*, *0*))
}›

definition *extract-trail-wl-heur* **where**
‹*extract-trail-wl-heur* = *isasat-state-ops.remove-a bottom-trail*›

sepref-def *bottom-trail-code*
is ‹*uncurry0* (*RETURN bottom-trail*)›
:: ‹*unit-assn*^{k} →$_a$ *trail-pol-fast-assn*›
unfolding *bottom-trail-def trail-pol-fast-assn-def*

apply (*rewrite* **in** ‹*let - = ⌑ in ->* *annotate-assn*[**where** *A=‹arl64-assn unat-lit-assn›*])
apply (*rewrite* **in** ‹*let - = ⌑ in ->* *al-fold-custom-empty*[**where** *'l=64*])
apply (*rewrite* **in** ‹*let - = -; - = ⌑ in ->* *al-fold-custom-empty*[**where** *'l=64*])
apply (*rewrite* **in** ‹*let - = -; - = ⌑ in ->* *annotate-assn*[**where** *A=‹arl64-assn uint32-nat-assn›*])

apply (*rewrite* **in** ‹*let - = -;- = ⌑ in ->* *annotate-assn*[**where** *A=‹larray64-assn (tri-bool-assn)›*])
apply (*rewrite* **in** ‹*let - = -;- = -;- = ⌑ in ->* *annotate-assn*[**where** *A=‹larray64-assn uint32-nat-assn›*])
apply (*rewrite* **in** ‹*let - = - in ->* *larray-fold-custom-replicate*)
apply (*rewrite* **in** ‹*let - = - in ->* *larray-fold-custom-replicate*)
apply (*rewrite* **in** ‹*let - = - in ->* *larray-fold-custom-replicate*)
apply (*rewrite at* ‹(-, ⌑, -)› *unat-const-fold*[**where** *'a=32*])
apply (*rewrite at* ‹(-, -, -, - , ⌑)› *snat-const-fold*[**where** *'a=64*])
apply (*rewrite at* ‹(*op-larray-custom-replicate* - ⌑)› *unat-const-fold*[**where** *'a=32*])
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
supply [[*goals-limit = 1*]]
by *sepref*

definition *bottom-arena* :: ‹*arena*› **where**
‹*bottom-arena* = []›

definition *extract-arena-wl-heur* **where**
‹*extract-arena-wl-heur* = *isasat-state-ops.remove-b bottom-arena*›

sepref-def *bottom-arena-code*
is ‹*uncurry0* (*RETURN bottom-arena*)›
:: ‹*unit-assn*^{k} →$_a$ *arena-fast-assn*›
unfolding *bottom-arena-def al-fold-custom-empty*[**where** *'l=64*]
by *sepref*

definition *bottom-conflict* :: ‹*conflict-option-rel*› **where**
‹*bottom-conflict* = (*True*, *0*, *replicate 0 NOTIN*)›

definition *extract-conflict-wl-heur* **where**
‹*extract-conflict-wl-heur* = *isasat-state-ops.remove-c bottom-conflict*›

sepref-def *bottom-conflict-code*
is ‹*uncurry0* (*RETURN bottom-conflict*)›
:: ‹*unit-assn*^{k} →$_a$ *conflict-option-rel-assn*›
unfolding *bottom-conflict-def*
conflict-option-rel-assn-def lookup-clause-rel-assn-def array-fold-custom-replicate
apply (*rewrite at* ‹(-, ⌑, -)› *unat-const-fold*[**where** *'a=32*])
apply (*annot-snat-const* ‹*TYPE*(*32*)›)
by *sepref*

definition *bottom-decision-level* :: *nat* **where**
‹*bottom-decision-level* = *0*›

definition *extract-literals-to-update-wl-heur* :: ‹- ⇒ -› **where**
‹*extract-literals-to-update-wl-heur* = *isasat-state-ops.remove-d bottom-decision-level*›

sepref-def *bottom-decision-level-code*
is ‹*uncurry0* (*RETURN bottom-decision-level*)›
:: ‹*unit-assn*^{k} →$_a$ *sint64-nat-assn*›
unfolding *bottom-decision-level-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

definition *bottom-watchlist* :: ‹(*nat*) *watcher list list*› **where**
‹*bottom-watchlist* = *replicate 0* []›

definition *extract-watchlist-wl-heur* **where**
‹*extract-watchlist-wl-heur* = *isasat-state-ops.remove-e bottom-watchlist*›

sepref-def *bottom-watchlist-code*
is ‹*uncurry0* (*RETURN bottom-watchlist*)›
:: ‹*unit-assn*^{k} →$_a$ *watchlist-fast-assn*›
unfolding *bottom-watchlist-def aal-fold-custom-empty*[**where** *'l=64* **and** *'ll=64*]
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

definition *bottom-atom* **where**
‹*bottom-atom* = *0*›
lemma [*sepref-fr-rules*]: ‹(*uncurry0* (*Mreturn 0*), *uncurry0* (*RETURN bottom-atom*)) ∈ *unit-assn*^{k} →$_a$ *atom-assn*›
unfolding *bottom-atom-def*
apply *sepref-to-hoare*
apply *vcg*
apply (*auto simp*: *atom-rel-def unat-rel-def unat.rel-def br-def entails-def ENTAILS-def*)
by (*smt* (*verit*, *best*) *pure-true-conv rel-simps*(*51*) *sep.add-0*)

definition *bottom-bump* :: ‹*bump-heuristics*› **where**
‹*bottom-bump* = *Tuple4 empty-acids bottom-vmtf False bottom-atms-hash*›

definition *extract-vmtf-wl-heur* **where**
‹*extract-vmtf-wl-heur* = *isasat-state-ops.remove-f bottom-bump*›

sepref-def *bottom-bump-code*
is ‹*uncurry0* (*RETURN bottom-bump*)›
:: ‹*unit-assn*^{k} →$_a$ *heuristic-bump-assn*›
unfolding *bottom-bump-def*
by *sepref*

definition *bottom-clvls* :: ‹*nat*› **where**
‹*bottom-clvls* = *0*›

definition *extract-clvls-wl-heur* **where**
‹*extract-clvls-wl-heur* = *isasat-state-ops.remove-g bottom-clvls*›

sepref-def *bottom-clvls-code*
is ‹*uncurry0* (*RETURN bottom-clvls*)›
:: ‹*unit-assn*^{k} →$_a$ *uint32-nat-assn*›
unfolding *bottom-clvls-def*
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

definition *bottom-ccach* :: ‹*minimize-status list* × *nat list*› **where**
‹*bottom-ccach* = (*replicate 0 SEEN-UNKNOWN*, [])›

definition *extract-ccach-wl-heur* **where**
‹*extract-ccach-wl-heur* = *isasat-state-ops.remove-h bottom-ccach*›

sepref-def *bottom-ccach-code*

is ‹*uncurry0* (*RETURN bottom-ccach*)›
:: ‹*unit-assn*^{k} →$_a$ *cach-refinement-l-assn*›
unfolding *bottom-ccach-def cach-refinement-l-assn-def array-fold-custom-replicate*
apply (*rewrite at* ‹(-, ⌑)› *al-fold-custom-empty*[**where** *'l=64*])
apply (*rewrite at* ‹(⌑, -)› *annotate-assn*[**where** *A* = ‹*IICF-Array.array-assn minimize-status-assn*›])
apply (*annot-snat-const* ‹*TYPE*(*32*)›)
by *sepref*

definition *extract-lbd-wl-heur* **where**
‹*extract-lbd-wl-heur* = *isasat-state-ops.remove-i empty-lbd*›

definition *bottom-outl* :: ‹*out-learned*› **where**
‹*bottom-outl* = []›

definition *extract-outl-wl-heur* **where**
‹*extract-outl-wl-heur* = *isasat-state-ops.remove-j bottom-outl*›

sepref-def *bottom-outl-code*
is ‹*uncurry0* (*RETURN bottom-outl*)›
:: ‹*unit-assn*^{k} →$_a$ *out-learned-assn*›
unfolding *bottom-outl-def cach-refinement-l-assn-def array-fold-custom-replicate*
apply (*rewrite at* ‹(⌑)› *al-fold-custom-empty*[**where** *'l=64*])
by *sepref*

definition *bottom-stats* :: ‹*isasat-stats*› **where**
‹*bottom-stats* = *empty-stats*›

definition *extract-stats-wl-heur* **where**
‹*extract-stats-wl-heur* = *isasat-state-ops.remove-k bottom-stats*›

sepref-def *bottom-stats-code*
is ‹*uncurry0* (*RETURN bottom-stats*)›
:: ‹*unit-assn*^{k} →$_a$ *isasat-stats-assn*›
unfolding *bottom-stats-def*
by *sepref*

definition *bottom-heur-int* :: ‹*restart-heuristics*› **where**
‹*bottom-heur-int* = (
let φ = *replicate 0 False in*
let fema = *ema-init* (*0*) *in*
let sema = *ema-init* (*0*) *in*
let other-fema = *ema-init* (*0*) *in*
let other-sema = *ema-init* (*0*) *in*
let ccount = *restart-info-init in*
let n = *0 in*
(*fema*, *sema*, *ccount*, *0*, (φ, *0*, *replicate n False*, *0*, *replicate n False*, *10000*, *1000*, *1*), *reluctant-init*, *False*, *replicate 0 False*, (*0*, *0*, *0*), *other-fema*, *other-sema*))
›
sepref-def *bottom-heur-int-code*
is ‹*uncurry0* (*RETURN bottom-heur-int*)›
:: ‹*unit-assn*^{k} →$_a$ *heuristic-int-assn*›
supply [[*goals-limit=1*]]
unfolding *bottom-heur-int-def heuristic-int-assn-def phase-heur-assn-def*
apply (*rewrite* **in** ‹(*replicate - False*, -)› *annotate-assn*[**where** *A=phase-saver'-assn*])
apply (*rewrite* **in** ‹(*replicate - False*, -)› *array-fold-custom-replicate*)
apply (*rewrite* **in** ‹(-, -, ⌑, -)› *array-fold-custom-replicate*)

apply (*rewrite* **in** ‹*let - =⌑ in ->* *annotate-assn*[**where** *A=phase-saver-assn*])
unfolding *larray-fold-custom-replicate*
apply (*annot-snat-const* ‹*TYPE(64)*›)
by *sepref*

definition *bottom-heur* :: ‹-› **where**
‹*bottom-heur = Restart-Heuristics* (*bottom-heur-int*)›

definition *extract-heur-wl-heur* **where**
‹*extract-heur-wl-heur = isasat-state-ops.remove-l bottom-heur*›

sepref-def *bottom-heur-code*
is ‹*uncurry0* (*RETURN bottom-heur*)›
:: ‹*unit-assn*^{k} →$_a$ *heuristic-assn*›
supply [[*goals-limit=1*]]
unfolding *bottom-heur-def*
by *sepref*

definition *bottom-vdom* :: ‹-› **where**
‹*bottom-vdom = AIvdom-init* [] [] []›

definition *extract-vdom-wl-heur* **where**
‹*extract-vdom-wl-heur = isasat-state-ops.remove-m bottom-vdom*›

sepref-def *bottom-vdom-code*
is ‹*uncurry0* (*RETURN bottom-vdom*)›
:: ‹*unit-assn*^{k} →$_a$ *aivdom-assn*›
supply [[*goals-limit=1*]]
unfolding *bottom-vdom-def*
unfolding *al-fold-custom-empty*[**where** *'l=64*]
by *sepref*

definition *bottom-lcount* :: ‹*clss-size*› **where**
‹*bottom-lcount* = (*0*, *0*, *0*, *0*, *0*)›

definition *extract-lcount-wl-heur* **where**
‹*extract-lcount-wl-heur = isasat-state-ops.remove-n bottom-lcount*›

sepref-def *bottom-lcount-code*
is ‹*uncurry0* (*RETURN bottom-lcount*)›
:: ‹*unit-assn*^{k} →$_a$ *lcount-assn*›
supply [[*goals-limit=1*]]
unfolding *bottom-lcount-def lcount-assn-def*
unfolding *al-fold-custom-empty*[**where** *'l=64*]
apply (*annot-unat-const* ‹*TYPE(64)*›)
by *sepref*

definition *bottom-opts* :: ‹*opts*› **where**
‹*bottom-opts = IsaOptions False False False 0 0 0 0 0 0 0 True*›

definition *extract-opts-wl-heur* **where**
‹*extract-opts-wl-heur = isasat-state-ops.remove-o bottom-opts*›

sepref-def *bottom-opts-code*
is ‹*uncurry0* (*RETURN bottom-opts*)›
:: ‹*unit-assn*^{k} →$_a$ *opts-assn*›

supply [[*goals-limit=1*]]
unfolding *bottom-opts-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

definition *bottom-old-arena* :: ‹*arena*› **where**
‹*bottom-old-arena* = []›

definition *extract-old-arena-wl-heur* **where**
‹*extract-old-arena-wl-heur* = *isasat-state-ops.remove-p bottom-old-arena*›

sepref-def *bottom-old-arena-code*
is ‹*uncurry0* (*RETURN bottom-old-arena*)›
:: ‹*unit-assn*^{k} →$_a$ *arena-fast-assn*›
supply [[*goals-limit=1*]]
unfolding *bottom-old-arena-def al-fold-custom-empty*[**where** *'l=64*]
by *sepref*

schematic-goal *free-trail-pol-fast-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE trail-pol-fast-assn ?a*›
unfolding *trail-pol-fast-assn-def*
by *synthesize-free*

sepref-def *free-trail-pol-fast*
is ‹*mop-free*›
:: ‹*trail-pol-fast-assn*^{d} →$_a$ *unit-assn*›
by *sepref*

lemma *free-trail-pol-fast-assn2*: ‹*MK-FREE trail-pol-fast-assn free-trail-pol-fast*›
unfolding *free-trail-pol-fast-def*
by (*rule back-subst*[*of* ‹*MK-FREE trail-pol-fast-assn*›, *OF free-trail-pol-fast-assn*])
(*auto intro*!: *ext*)

schematic-goal *free-arena-fast-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE arena-fast-assn ?a*›
by *synthesize-free*

sepref-def *free-arena-fast*
is ‹*mop-free*›
:: ‹*arena-fast-assn*^{d} →$_a$ *unit-assn*›
by *sepref*

lemma *free-arena-fast-assn2*: ‹*MK-FREE arena-fast-assn free-arena-fast*›
unfolding *free-arena-fast-def*
by (*rule back-subst*[*of* ‹*MK-FREE arena-fast-assn*›, *OF free-arena-fast-assn*])
(*auto intro*!: *ext*)

schematic-goal *free-conflict-option-rel-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE conflict-option-rel-assn ?a*›
by *synthesize-free*

sepref-def *free-conflict-option-rel*
is ‹*mop-free*›
:: ‹*conflict-option-rel-assn*^{d} →$_a$ *unit-assn*›
by *sepref*

lemma *free-conflict-option-rel-assn2*: ‹*MK-FREE conflict-option-rel-assn free-conflict-option-rel*›
unfolding *free-conflict-option-rel-def*

by (*rule back-subst*[*of* ‹*MK-FREE conflict-option-rel-assn*›, *OF free-conflict-option-rel-assn*])
(*auto intro*!: *ext*)

schematic-goal *free-sint64-nat-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE sint64-nat-assn ?a*›
by *synthesize-free*

sepref-def *free-sint64-nat*
is ‹*mop-free*›
:: ‹*sint64-nat-assn*^{d} →$_a$ *unit-assn*›
by *sepref*

lemma *free-sint64-nat-assn-assn2*: ‹*MK-FREE sint64-nat-assn free-sint64-nat*›
unfolding *free-sint64-nat-def*
by (*rule back-subst*[*of* ‹*MK-FREE sint64-nat-assn*›, *OF free-sint64-nat-assn*])
(*auto intro*!: *ext*)

schematic-goal *free-watchlist-fast-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE watchlist-fast-assn ?a*›
by *synthesize-free*

sepref-def *free-watchlist-fast*
is ‹*mop-free*›
:: ‹*watchlist-fast-assn*^{d} →$_a$ *unit-assn*›
by *sepref*

lemma *free-watchlist-fast-assn2*: ‹*MK-FREE watchlist-fast-assn free-watchlist-fast*›
unfolding *free-watchlist-fast-def*
by (*rule back-subst*[*of* ‹*MK-FREE watchlist-fast-assn*›, *OF free-watchlist-fast-assn*])
(*auto intro*!: *ext*)

schematic-goal *free-heuristic-bump-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE heuristic-bump-assn ?a*›
unfolding *heuristic-bump-assn-def*
by *synthesize-free*

sepref-def *free-vmtf-remove*
is ‹*mop-free*›
:: ‹*heuristic-bump-assn*^{d} →$_a$ *unit-assn*›
by *sepref*

lemma *free-heuristic-bump-assn2*: ‹*MK-FREE heuristic-bump-assn free-vmtf-remove*›
unfolding *free-vmtf-remove-def*
apply (*rule back-subst*[*of* ‹*MK-FREE heuristic-bump-assn*›, *OF free-heuristic-bump-assn*])
apply (*auto intro*!: *ext simp*: *M-monad-laws*)
by (*metis M-monad-laws*(*1*))

schematic-goal *free-uint32-nat-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE uint32-nat-assn ?a*›
by *synthesize-free*

sepref-def *free-uint32-nat*
is ‹*mop-free*›
:: ‹*uint32-nat-assn*^{d} →$_a$ *unit-assn*›
by *sepref*

lemma *free-uint32-nat-assn2*: ‹*MK-FREE uint32-nat-assn free-uint32-nat*›
unfolding *free-uint32-nat-def*
by (*rule back-subst*[*of* ‹*MK-FREE uint32-nat-assn*›, *OF free-uint32-nat-assn*])

(*auto intro*!: *ext*)

schematic-goal *free-cach-refinement-l-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE cach-refinement-l-assn ?a*›
by *synthesize-free*

sepref-def *free-cach-refinement-l*
is ‹*mop-free*›
:: ‹*cach-refinement-l-assn*^{d} →$_a$ *unit-assn*›
by *sepref*

lemma *free-cach-refinement-l-assn2*: ‹*MK-FREE cach-refinement-l-assn free-cach-refinement-l*›
unfolding *free-cach-refinement-l-def*
by (*rule back-subst*[*of* ‹*MK-FREE cach-refinement-l-assn*›, *OF free-cach-refinement-l-assn*])
(*auto intro*!: *ext*)

schematic-goal *free-lbd-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE lbd-assn ?a*›
by *synthesize-free*

sepref-def *free-lbd*
is ‹*mop-free*›
:: ‹*lbd-assn*^{d} →$_a$ *unit-assn*›
by *sepref*

lemma *free-lbd-assn2*: ‹*MK-FREE lbd-assn free-lbd*›
unfolding *free-lbd-def*
by (*rule back-subst*[*of* ‹*MK-FREE lbd-assn*›, *OF free-lbd-assn*])
(*auto intro*!: *ext*)

schematic-goal *free-outl-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE out-learned-assn ?a*›
by *synthesize-free*

sepref-def *free-outl*
is ‹*mop-free*›
:: ‹*out-learned-assn*^{d} →$_a$ *unit-assn*›
by *sepref*

lemma *free-outl-assn2*: ‹*MK-FREE out-learned-assn free-outl*›
unfolding *free-outl-def*
by (*rule back-subst*[*of* ‹*MK-FREE out-learned-assn*›, *OF free-outl-assn*])
(*auto intro*!: *ext*)

schematic-goal *free-heur-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE heuristic-assn ?a*›
unfolding *heuristic-assn-def code-hider-assn-def*
by *synthesize-free*

sepref-def *free-heur*
is ‹*mop-free*›
:: ‹*heuristic-assn*^{d} →$_a$ *unit-assn*›
by *sepref*

lemma *free-heur-assn2*: ‹*MK-FREE heuristic-assn free-heur*›
unfolding *free-heur-def*
by (*rule back-subst*[*of* ‹*MK-FREE heuristic-assn*›, *OF free-heur-assn*])
(*auto intro*!: *ext*)

schematic-goal *free-isasat-stats-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE isasat-stats-assn ?a*›
  unfolding *isasat-stats-assn-def code-hider-assn-def*
  by *synthesize-free*

sepref-def *free-stats*
  is ‹*mop-free*›
  :: ‹*isasat-stats-assn*^{d} →$_a$ *unit-assn*›
  by *sepref*

lemma *free-isasat-stats-assn2*: ‹*MK-FREE isasat-stats-assn free-stats*›
  unfolding *free-stats-def*
  by (*rule back-subst*[*of* ‹*MK-FREE isasat-stats-assn*›, *OF free-isasat-stats-assn*])
    (*auto intro*!: *ext*)

schematic-goal *free-vdom-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE aivdom-assn ?a*›
    unfolding *aivdom-assn-def code-hider-assn-def*
    by *synthesize-free*

sepref-def *free-vdom*
  is ‹*mop-free*›
  :: ‹*aivdom-assn*^{d} →$_a$ *unit-assn*›
  by *sepref*

lemma *free-vdom-assn2*: ‹*MK-FREE aivdom-assn free-vdom*›
  unfolding *free-vdom-def*
  by (*rule back-subst*[*of* ‹*MK-FREE aivdom-assn*›, *OF free-vdom-assn*])
    (*auto intro*!: *ext*)

schematic-goal *free-lcount-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE lcount-assn ?a*›
    unfolding *lcount-assn-def code-hider-assn-def*
    by *synthesize-free*

sepref-def *free-lcount*
  is ‹*mop-free*›
  :: ‹*lcount-assn*^{d} →$_a$ *unit-assn*›
  by *sepref*

lemma *free-lcount-assn2*: ‹*MK-FREE lcount-assn free-lcount*›
  unfolding *free-lcount-def*
  by (*rule back-subst*[*of* ‹*MK-FREE lcount-assn*›, *OF free-lcount-assn*])
    (*auto intro*!: *ext*)

schematic-goal *free-opts-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE opts-assn ?a*›
    unfolding *opts-assn-def code-hider-assn-def opts-rel-assn-def*
    by *synthesize-free*

sepref-def *free-opts*
  is ‹*mop-free*›
  :: ‹*opts-assn*^{d} →$_a$ *unit-assn*›
  by *sepref*

lemma *free-opts-assn2*: ‹*MK-FREE opts-assn free-opts*›
  unfolding *free-opts-def*
  by (*rule back-subst*[*of* ‹*MK-FREE opts-assn*›, *OF free-opts-assn*])
    (*auto intro*!: *ext*)

schematic-goal *free-old-arena-fast-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE arena-fast-assn ?a*›
  by *synthesize-free*

sepref-def *free-old-arena-fast*
  is ‹*mop-free*›
  :: ‹*arena-fast-assn*^{d} →$_a$ *unit-assn*›
  by *sepref*

lemma *free-old-arena-fast-assn2*: ‹*MK-FREE arena-fast-assn free-old-arena-fast*›
  unfolding *free-old-arena-fast-def free-arena-fast-def*
  by (*rule back-subst*[*of* ‹*MK-FREE arena-fast-assn*›, *OF free-old-arena-fast-assn*])
    (*auto intro*!: *ext*)

sepref-def *free-occs-fast*
  is ‹*mop-free*›
  :: ‹*occs-assn*^{d} →$_a$ *unit-assn*›
  by *sepref*

definition *bottom-occs* :: ‹*nat list list*› **where**
  ‹*bottom-occs* = *op-aal-lempty TYPE*(*64*) *TYPE*(*64*) *0*›

definition *extract-occs-wl-heur* **where**
  ‹*extract-occs-wl-heur* = *isasat-state-ops.remove-q bottom-occs*›

sepref-def *bottom-occs-code*
  is ‹*uncurry0* (*RETURN bottom-occs*)›
  :: ‹*unit-assn*^{k} →$_a$ *occs-assn*›
  supply [[*goals-limit*=*1*]]
  unfolding *bottom-occs-def aal-fold-custom-empty*[**where** *'l*=*64* **and** *'ll*=*64*]
  apply (*annot-snat-const* ‹*TYPE*(*64*)›)
  by *sepref*

schematic-goal *free-occs-fast-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE occs-assn ?a*›
  by *synthesize-free*

lemma *free-occs-fast-assn2*: ‹*MK-FREE occs-assn free-occs-fast*›
  unfolding *free-occs-fast-def*
  by (*rule back-subst*[*of* ‹*MK-FREE occs-assn*›, *OF free-occs-fast-assn*])
    (*auto intro*!: *ext*)

global-interpretation *isasat-state* **where**
  a-assn = *trail-pol-fast-assn* **and**
  b-assn = *arena-fast-assn* **and**
  c-assn = *conflict-option-rel-assn* **and**
  d-assn = *sint64-nat-assn* **and**
  e-assn = *watchlist-fast-assn* **and**
  f-assn = *heuristic-bump-assn* **and**
  g-assn = *uint32-nat-assn* **and**
  h-assn = *cach-refinement-l-assn* **and**
  i-assn = *lbd-assn* **and**
  j-assn = *out-learned-assn* **and**
  k-assn = *isasat-stats-assn* **and**
  l-assn = *heuristic-assn* **and**
  m-assn = *aivdom-assn* **and**
  n-assn = *lcount-assn* **and**

o-assn = *opts-assn* **and**
p-assn = *arena-fast-assn* **and**
q-assn = *occs-assn* **and**
a-default = *bottom-trail* **and**
a = ‹*bottom-trail-code*› **and**
a-free = *free-trail-pol-fast* **and**
b-default = *bottom-arena* **and**
b = ‹*bottom-arena-code*› **and**
b-free = *free-arena-fast* **and**
c-default = *bottom-conflict* **and**
c = ‹*bottom-conflict-code*› **and**
c-free = *free-conflict-option-rel* **and**
d-default = ‹*bottom-decision-level*› **and**
d = ‹(*bottom-decision-level-code*)› **and**
d-free = ‹*free-sint64-nat*› **and**
e-default = *bottom-watchlist* **and**
e = ‹*bottom-watchlist-code*› **and**
e-free = *free-watchlist-fast* **and**
f-default = *bottom-bump* **and**
f = ‹*bottom-bump-code*› **and**
f-free = *free-vmtf-remove* **and**
g-default = *bottom-clvls* **and**
g = ‹*bottom-clvls-code*›**and**
g-free = *free-uint32-nat* **and**
h-default = *bottom-ccach* **and**
h = ‹*bottom-ccach-code*› **and**
h-free = *free-cach-refinement-l* **and**
i-default = *empty-lbd* **and**
i = ‹*empty-lbd-code*› **and**
i-free = *free-lbd* **and**
j-default = *bottom-outl* **and**
j = ‹*bottom-outl-code*› **and**
j-free = *free-outl* **and**
k-default = *bottom-stats* **and**
k = ‹*bottom-stats-code*› **and**
k-free = *free-stats* **and**
l-default = *bottom-heur* **and**
l = ‹*bottom-heur-code*› **and**
l-free = *free-heur* **and**
m-default = *bottom-vdom* **and**
m = ‹*bottom-vdom-code*› **and**
m-free = *free-vdom* **and**
n-default = *bottom-lcount* **and**
n = ‹*bottom-lcount-code*› **and**
n-free = *free-lcount* **and**
ko-default = *bottom-opts* **and**
ko = ‹*bottom-opts-code*› **and**
o-free = *free-opts* **and**
p-default = *bottom-old-arena* **and**
p = ‹*bottom-old-arena-code*› **and**
p-free = *free-old-arena-fast* **and**
q-default = *bottom-occs* **and**
q = *bottom-occs-code* **and**
q-free = *free-occs-fast*
rewrites
‹*isasat-assn* ≡ *isasat-bounded-assn*› **and**

‹*remove-a* ≡ *extract-trail-wl-heur*› **and**
‹*remove-b* ≡ *extract-arena-wl-heur*› **and**
‹*remove-c* ≡ *extract-conflict-wl-heur*› **and**
‹*remove-d* ≡ *extract-literals-to-update-wl-heur*› **and**
‹*remove-e* ≡ *extract-watchlist-wl-heur*› **and**
‹*remove-f* ≡ *extract-vmtf-wl-heur*› **and**
‹*remove-g* ≡ *extract-clvls-wl-heur*› **and**
‹*remove-h* ≡ *extract-ccach-wl-heur*› **and**
‹*remove-i* ≡ *extract-lbd-wl-heur*› **and**
‹*remove-j* ≡ *extract-outl-wl-heur*› **and**
‹*remove-k* ≡ *extract-stats-wl-heur*› **and**
‹*remove-l* ≡ *extract-heur-wl-heur*› **and**
‹*remove-m* ≡ *extract-vdom-wl-heur*› **and**
‹*remove-n* ≡ *extract-lcount-wl-heur*› **and**
‹*remove-o* ≡ *extract-opts-wl-heur*› **and**
‹*remove-p* ≡ *extract-old-arena-wl-heur*›**and**
‹*remove-q* ≡ *extract-occs-wl-heur*›
apply *unfold-locales*
subgoal by (*rule bottom-trail-code.refine*)
subgoal by (*rule bottom-arena-code.refine*)
subgoal by (*rule bottom-conflict-code.refine*)
subgoal by (*rule bottom-decision-level-code.refine*)
subgoal by (*rule bottom-watchlist-code.refine*)
subgoal by (*rule bottom-bump-code.refine*)
subgoal by (*rule bottom-clvls-code.refine*)
subgoal by (*rule bottom-ccach-code.refine*)
subgoal by (*rule empty-lbd-hnr*)
subgoal by (*rule bottom-outl-code.refine*)
subgoal by (*rule bottom-stats-code.refine*)
subgoal by (*rule bottom-heur-code.refine*)
subgoal by (*rule bottom-vdom-code.refine*)
subgoal by (*rule bottom-lcount-code.refine*)
subgoal by (*rule bottom-opts-code.refine*)
subgoal by (*rule bottom-old-arena-code.refine*)
subgoal by (*rule bottom-occs-code.refine*)
subgoal by (*rule free-trail-pol-fast-assn2*)
subgoal by (*rule free-arena-fast-assn2*)
subgoal by (*rule free-conflict-option-rel-assn2*)
subgoal by (*rule free-sint64-nat-assn-assn2*)
subgoal by (*rule free-watchlist-fast-assn2*)
subgoal by (*rule free-heuristic-bump-assn2*)
subgoal by (*rule free-uint32-nat-assn2*)
subgoal by (*rule free-cach-refinement-l-assn2*)
subgoal by (*rule free-lbd-assn2*)
subgoal by (*rule free-outl-assn2*)
subgoal by (*rule free-isasat-stats-assn2*)
subgoal by (*rule free-heur-assn2*)
subgoal by (*rule free-vdom-assn2*)
subgoal by (*rule free-lcount-assn2*)
subgoal by (*rule free-opts-assn2*)
subgoal by (*rule free-old-arena-fast-assn2*)
subgoal by (*rule free-occs-fast-assn2*)
subgoal unfolding *isasat-bounded-assn-def isasat-state-ops.isasat-assn-def* **.**
subgoal unfolding *extract-trail-wl-heur-def* **.**
subgoal unfolding *extract-arena-wl-heur-def* **.**
subgoal unfolding *extract-conflict-wl-heur-def* **.**

subgoal unfolding *extract-literals-to-update-wl-heur-def* .
subgoal unfolding *extract-watchlist-wl-heur-def* .
subgoal unfolding *extract-vmtf-wl-heur-def* .
subgoal unfolding *extract-clvls-wl-heur-def* .
subgoal unfolding *extract-ccach-wl-heur-def* .
subgoal unfolding *extract-lbd-wl-heur-def* .
subgoal unfolding *extract-outl-wl-heur-def* .
subgoal unfolding *extract-stats-wl-heur-def* .
subgoal unfolding *extract-heur-wl-heur-def* .
subgoal unfolding *extract-vdom-wl-heur-def* .
subgoal unfolding *extract-lcount-wl-heur-def* .
subgoal unfolding *extract-opts-wl-heur-def* .
subgoal unfolding *extract-old-arena-wl-heur-def* .
subgoal unfolding *extract-occs-wl-heur-def* .
done

lemma [*llvm-pre-simp*]:
‹(*Mreturn* ∘$_{17}$ *IsaSAT-int*) *a1 a2 a3 x a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17* =
Mreturn (*IsaSAT-int a1 a2 a3 x a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17*)›
by *auto*

lemmas [*llvm-code del*] =
update-a-code-def
update-b-code-def
update-c-code-def
update-d-code-def
update-e-code-def
update-f-code-def
update-h-code-def
update-i-code-def
update-j-code-def
update-k-code-def
update-l-code-def
update-m-code-def
update-n-code-def
update-o-code-def
update-p-code-def
update-q-code-def

lemmas [*unfolded comp-def inline-node-case*, *llvm-code*] =
remove-d-code-alt-def
remove-b-code-alt-def
remove-a-code-alt-def
bottom-decision-level-code-def
bottom-arena-code-def
bottom-trail-code-def
update-a-code-alt-def
update-b-code-alt-def
update-c-code-alt-def
update-d-code-alt-def
update-e-code-alt-def
update-f-code-alt-def
update-h-code-alt-def
update-i-code-alt-def

update-j-code-alt-def
update-k-code-alt-def
update-l-code-alt-def
update-m-code-alt-def
update-n-code-alt-def
update-o-code-alt-def
update-p-code-alt-def
update-q-code-alt-def

lemma *add-pure-parameter*:
assumes ‹$\bigwedge C\ C'.\ (C,\ C') \in R \Longrightarrow (f\ C,\ f'\ C') \in [P\ C']_a\ A \to b$›
shows ‹$(uncurry\ f,\ uncurry\ f') \in [uncurry\ P]_a\ (pure\ R)^k *_a A \to b$›
apply *sepref-to-hoare*
apply *vcg*
apply *auto*
apply (*subst POSTCOND-def hn-ctxt-def sep-conj-empty′ pure-true-conv EXTRACT-def*)+
apply (*rule assms*[*to-hnr*, *simplified*, *unfolded hn-ctxt-def hn-refine-def htriple-def sep-conj-empty′ pure-true-conv*, *rule-format*])
apply *auto*
done

lemma *remove-pure-parameter*:
assumes ‹$(uncurry\ f,\ uncurry\ f') \in [uncurry\ P]_a\ (pure\ R)^k *_a A \to b$› ‹$(C,\ C') \in R$›
shows ‹$(f\ C,\ f'\ C') \in [P\ C']_a\ A \to b$›
using *assms*(*2*) *assms*(*1*)[*to-hnr*, *simplified*, *unfolded hn-ctxt-def hn-refine-def htriple-def sep-conj-empty′ pure-true-conv*, *rule-format*]
apply *sepref-to-hoare*
apply *vcg*
apply (*subst POSTCOND-def hn-ctxt-def sep-conj-empty′ pure-true-conv EXTRACT-def*)+
by (*auto simp*: *pure-true-conv*)

lemma *add-pure-parameter2*:
assumes ‹$\bigwedge C\ C'.\ (C,\ C') \in R \Longrightarrow (\lambda S.\ f\ S\ C,\ \lambda S.\ f'\ S\ C') \in [\lambda S.\ P\ S\ C']_a\ A \to b$›
shows ‹$(uncurry\ f,\ uncurry\ f') \in [uncurry\ P]_a\ A *_a (pure\ R)^k \to b$›
apply *sepref-to-hoare*
apply *vcg*
apply *auto*
apply (*subst POSTCOND-def hn-ctxt-def sep-conj-empty′ pure-true-conv EXTRACT-def*)+
apply (*rule assms*[*to-hnr*, *simplified*, *unfolded hn-ctxt-def hn-refine-def htriple-def sep-conj-empty′ pure-true-conv*, *rule-format*])
apply *auto*
done

lemma *remove-pure-parameter2*:
assumes ‹$(uncurry\ f,\ uncurry\ f') \in [uncurry\ P]_a\ A *_a (pure\ R)^k \to b$› ‹$(C,\ C') \in R$›
shows ‹$(\lambda S.\ f\ S\ C,\ \lambda S.\ f'\ S\ C') \in [\lambda S.\ P\ \ S\ C']_a\ A \to b$›
using *assms*(*2*) *assms*(*1*)[*to-hnr*, *simplified*, *unfolded hn-ctxt-def hn-refine-def htriple-def sep-conj-empty′ pure-true-conv*, *rule-format*]
apply *sepref-to-hoare*
apply *vcg*
apply (*subst POSTCOND-def hn-ctxt-def sep-conj-empty′ pure-true-conv EXTRACT-def*)+
apply (*auto simp*: *pure-true-conv*)
done

lemma *remove-pure-parameter2′*:
assumes ‹$(uncurry\ (\lambda S\ C.\ f\ C\ S),\ uncurry\ (\lambda S\ C.\ f'\ C\ S)) \in [uncurry\ P]_a\ A *_a (pure\ R)^k \to b$›

‹(C, C') ∈ R›
shows ‹(f C, f' C') ∈ [λS. P S C']$_a$ A → b›
by (*rule remove-pure-parameter2*)
(*rule assms*)+

lemma *remove-pure-parameter2-twoargs*:
assumes ‹(*uncurry2 f*, *uncurry2 f'*) ∈ [*uncurry2 P*]$_a$ *A* *$_a$ (*pure R*)^k *$_a$ (*pure R'*)^k → b› ‹(C, C') ∈ R› ‹(D,D')∈R'›
shows ‹(λS. f S C D, λS. f' S C' D') ∈ [λS. P S C' D']$_a$ A → b›
using *assms*(2−) *assms*(1)[*to-hnr*, *simplified*, *unfolded hn-ctxt-def hn-refine-def htriple-def sep-conj-empty' pure-true-conv*, *rule-format*]
apply *sepref-to-hoare*
apply *vcg*
apply (*subst POSTCOND-def hn-ctxt-def sep-conj-empty' pure-true-conv EXTRACT-def*)+
apply (*auto simp*: *pure-true-conv*)
done

locale *read-trail-param-adder0-ops* =
fixes P :: ‹*trail-pol* ⇒ *bool*› **and** f' :: ‹*trail-pol* ⇒ 'r *nres*›
begin

definition *mop* **where**
‹*mop N* = *do* {
ASSERT (P (*get-trail-wl-heur N*));
read-all-st (λM - - - - - - - - - - - - - - - - -. f' M) N
}›

end

lemma *remove-component-right*:
assumes ‹(f, f') ∈ [P]$_a$ A → B›
shows ‹(*uncurry* (λM -. f M), *uncurry* (λM -. f' M)) ∈ [*uncurry* (λM -. P M)]$_a$ A *$_a$ X^k → B›
using *assms*
unfolding *hfref-def*
apply *clarsimp*
apply (*rule hn-refine-frame'*)
apply *auto*
done
lemma *hn-refine-frame''*: *hn-refine* Γ c Γ' R CP m ⟹ *hn-refine* (F**Γ) c (F**Γ') R CP m
by (*metis hn-refine-frame' sep-conj-c*(1))

lemma *hn-refine-frame-mid''*: *hn-refine* (F**G) c (F'**G') R CP m ⟹ *hn-refine* (F**Γ**G) c (F'**Γ**G') R CP m
by (*smt* (*verit*) *hn-refine-frame' sep-conj-aci*(2) *sep-conj-c*(1))

lemma *remove-component-left*:
assumes ‹(f, f') ∈ [P]$_a$ A → B›
shows ‹(*uncurry* (λ- M. f M), *uncurry* (λ- M. f' M)) ∈ [*uncurry* (λ- M. P M)]$_a$ X^k *$_a$ A → B›
using *assms*
unfolding *hfref-def*
apply *clarsimp*
apply (*rule hn-refine-frame''*)
apply *auto*
done

lemma *remove-component-middle*:
assumes ‹$(f, f') \in [P]_a\ A *_a B \to C$›
shows ‹$(uncurry2\ (\lambda M\ -\ N.\ f\ (M,\ N)),\ uncurry2\ (\lambda M\ -\ N.\ f'\ (M,\ N))) \in [uncurry2\ (\lambda M\ -\ N.\ P\ (M,\ N))]_a\ \ A *_a X^k *_a B \to C$›
using *assms*
unfolding *hfref-def*
apply *clarsimp*
apply (*rule hn-refine-frame-mid''*)
apply *auto*
done

lemma (**in** −)*hfref-cong*: ‹$(a,\ b) \in [P]_a\ A \to B \Longrightarrow a = a' \Longrightarrow b = b' \Longrightarrow P = P' \Longrightarrow (a',b') \in [P']_a\ A \to B$›
by *auto*

lemma *split-snd-pure-arg*:
assumes ‹$(uncurry\ (\lambda N\ C.\ f\ C\ N),\ uncurry\ (\lambda N\ C'.\ \ f'\ C'\ N))$
$\in [uncurry\ (\lambda S\ C.\ P\ S\ C)]_a\ K^k *_a (pure\ (R \times_f R'))^k \to x\text{-}assn$›
shows ‹$(uncurry2\ (\lambda N\ C\ D.\ f\ (C,D)\ N),\ uncurry2\ (\lambda N\ C'\ D'.\ \ f'\ (C',D')\ N))$
$\in [uncurry2\ (\lambda S\ C\ D.\ P\ S\ (C,\ D))]_a\ K^k *_a (pure(R))^k *_a (pure\ R')^k \to x\text{-}assn$›
using *assms* **unfolding** *hfref-def*
by (*auto simp flip*: *prod-assn-pure-conv*)

lemma *add-pure-parameter2-twoargs*:
assumes ‹$\bigwedge C\ C'\ D\ D'.\ (C,\ C') \in R \Longrightarrow\ (D,\ D') \in R' \Longrightarrow (\lambda S.\ f\ S\ C\ D,\ \lambda S.\ f'\ S\ C'\ D') \in [\lambda S.\ P\ S\ C'\ D']_a\ A \to b$›
shows ‹$(uncurry2\ f,\ uncurry2\ f') \in [uncurry2\ P]_a\ A *_a (pure\ R)^k *_a (pure\ R')^k \to b$›
apply *sepref-to-hoare*
apply *vcg*
apply *auto*
apply (*subst POSTCOND-def hn-ctxt-def sep-conj-empty' pure-true-conv EXTRACT-def*)+
apply (*rule assms*[*to-hnr*, *simplified*, *unfolded hn-ctxt-def hn-refine-def htriple-def EXTRACT-def sep-conj-empty' pure-true-conv*, *rule-format*])
apply *auto*
done

lemma *remove-unused-unit-parameter*:
assumes ‹$(uncurry\ (\lambda S\ -.\ f\ S),\ uncurry\ (\lambda S\ -.\ f'\ S)) \in [uncurry\ (\lambda S\ -.\ P\ S)]_a\ A *_a (pure\ unit\text{-}rel)^k \to b$›
shows ‹$(f,\ f') \in [P]_a\ A \to b$›
apply *sepref-to-hoare*
apply *vcg*
apply (*subst POSTCOND-def hn-ctxt-def sep-conj-empty' pure-true-conv EXTRACT-def*)+
apply (*rule assms*[*to-hnr*, *simplified*, *unfolded hn-ctxt-def hn-refine-def htriple-def sep-conj-empty' pure-true-conv*, *rule-format*])
apply *auto*
done

lemma *add-pure-parameter-unit*:
assumes ‹$(\lambda S.\ f\ S\ (),\ \lambda S.\ f'\ S\ ()) \in [\lambda S.\ P\ S]_a\ A \to b$›
shows ‹$(f\ (),\ f'\ ()) \in [P]_a\ A \to b$›
apply *sepref-to-hoare*
apply *vcg*
apply (*subst POSTCOND-def hn-ctxt-def sep-conj-empty' pure-true-conv EXTRACT-def*)+
apply (*rule assms*[*to-hnr*, *simplified*, *unfolded hn-ctxt-def hn-refine-def htriple-def*

sep-conj-empty′ pure-true-conv, *rule-format*])
apply *auto*
done

abbreviation *read-trail-wl-heur-code* :: ‹- ⇒ - ⇒ -› **where**
‹*read-trail-wl-heur-code f* ≡ *read-all-st-code* (λ*M* - - - - - - - - - - - - - - - - -. *f M*)›
abbreviation *read-trail-wl-heur* :: ‹- ⇒ *isasat* ⇒ -› **where**
‹*read-trail-wl-heur f* ≡ *read-all-st* (λ*M* - - - - - - - - - - - - - - - - -. *f M*)›

locale *read-trail-param-adder0* = *read-trail-param-adder0-ops P f′*
for *P* :: ‹ *trail-pol* ⇒ *bool*› **and** *f′* :: ‹*trail-pol* ⇒ *′r nres*› +
fixes *f* **and** *x-assn* :: ‹*′r* ⇒ *′q* ⇒ *assn*›
assumes *not-deleted-code-refine*: ‹(*f*, *f′*) ∈ [*P*]$_a$ *trail-pol-fast-assn*^{k} → *x-assn*›
begin

I tried to automate the generation of the theorem but I failed (although generating the sequence is actually very easy...)

lemma *not-deleted-code-refine′*:
‹(*uncurry16* (λ*M* - - - - - - - - - - - - - - - - -. *f M*), *uncurry16* (λ*M* - - - - - - - - - - - - - - - - -. *f′ M*)) ∈
[*uncurry16* (λ*M* - - - - - - - - - - - - - - - - -. *P M*)]$_a$
trail-pol-fast-assn^k *∗$_a$ arena-fast-assn*^{k} *∗$_a$ conflict-option-rel-assn*^{k} *∗$_a$ sint64-nat-assn*^{k} *∗$_a$*
watchlist-fast-assn^k *∗$_a$ heuristic-bump-assn*^{k} *∗$_a$ uint32-nat-assn*^{k} *∗$_a$ cach-refinement-l-assn*^{k} *∗$_a$*
lbd-assn^k *∗$_a$ out-learned-assn*^{k} *∗$_a$ isasat-stats-assn*^{k} *∗$_a$ heuristic-assn*^{k} *∗$_a$*
aivdom-assn^k *∗$_a$ lcount-assn*^{k} *∗$_a$ opts-assn*^{k} *∗$_a$ arena-fast-assn*^{k} *∗$_a$ occs-assn*^{k} → *x-assn*›
apply (*insert not-deleted-code-refine*)
apply (*drule remove-component-right*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-right*[**where** *X* = *conflict-option-rel-assn*])
apply (*drule remove-component-right*[**where** *X* = *sint64-nat-assn*])
apply (*drule remove-component-right*[**where** *X* = *watchlist-fast-assn*])
apply (*drule remove-component-right*[**where** *X* = *heuristic-bump-assn*])
apply (*drule remove-component-right*[**where** *X* = *uint32-nat-assn*])
apply (*drule remove-component-right*[**where** *X* = *cach-refinement-l-assn*])
apply (*drule remove-component-right*[**where** *X* = *lbd-assn*])
apply (*drule remove-component-right*[**where** *X* = *out-learned-assn*])
apply (*drule remove-component-right*[**where** *X* = *isasat-stats-assn*])
apply (*drule remove-component-right*[**where** *X* = *heuristic-assn*])
apply (*drule remove-component-right*[**where** *X* = *aivdom-assn*])
apply (*drule remove-component-right*[**where** *X* = *lcount-assn*])
apply (*drule remove-component-right*[**where** *X* = *opts-assn*])
apply (*drule remove-component-right*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-right*[**where** *X* = *occs-assn*])
apply (*rule hfref-cong*, *assumption*)
apply (*auto simp add*: *uncurry-def*)
done

lemmas *refine* = *read-all-code-refine*[*OF not-deleted-code-refine′*]

lemma *mop-refine*:
‹(*read-trail-wl-heur-code f*, *mop*) ∈ *isasat-bounded-assn*^{k}→$_a$ *x-assn*›
unfolding *mop-def*
apply (*rule refine-ASSERT-move-to-pre0*)
apply (*rule hfref-cong*[*OF refine*])
apply (*auto intro*!: *ext split*: *isasat-int-splits*)
done

end

locale *read-all-param-adder-ops* =
  fixes $f' :: \langle 'a \Rightarrow$ *trail-pol* $\Rightarrow$ *arena* $\Rightarrow$
    conflict-option-rel $\Rightarrow$ *nat* $\Rightarrow$ (*nat watcher*) *list list* $\Rightarrow$ *bump-heuristics* $\Rightarrow$
    nat $\Rightarrow$ *conflict-min-cach-l* $\Rightarrow$ *lbd* $\Rightarrow$ *out-learned* $\Rightarrow$ *isasat-stats* $\Rightarrow$ *isasat-restart-heuristics* $\Rightarrow$
   isasat-aivdom $\Rightarrow$ *clss-size* $\Rightarrow$ *opts* $\Rightarrow$ *arena* $\Rightarrow$ *occurences-ref* $\Rightarrow$ $'e$ *nres*$\rangle$ **and**
   $P :: \langle 'a \Rightarrow$ *trail-pol* $\Rightarrow$ *arena* $\Rightarrow$
    conflict-option-rel $\Rightarrow$ *nat* $\Rightarrow$ (*nat watcher*) *list list* $\Rightarrow$ *bump-heuristics* $\Rightarrow$
    nat $\Rightarrow$ *conflict-min-cach-l* $\Rightarrow$ *lbd* $\Rightarrow$ *out-learned* $\Rightarrow$ *isasat-stats* $\Rightarrow$ *isasat-restart-heuristics* $\Rightarrow$
   isasat-aivdom $\Rightarrow$ *clss-size* $\Rightarrow$ *opts* $\Rightarrow$ *arena* $\Rightarrow$ *occurences-ref* $\Rightarrow$ *bool*$\rangle$
begin
definition *mop* **where**
  $\langle$*mop S C* = *do* {
  ASSERT (*P C* (*get-trail-wl-heur S*)
  (*get-clauses-wl-heur S*)
  (*get-conflict-wl-heur S*)
  (*literals-to-update-wl-heur S*)
  (*get-watched-wl-heur S*)
  (*get-vmtf-heur S*)
  (*get-count-max-lvls-heur S*)
  (*get-conflict-cach S*)
  (*get-lbd S*)
  (*get-outlearned-heur S*)
  (*get-stats-heur S*)
  (*get-heur S*)
  (*get-aivdom S*)
  (*get-learned-count S*)
  (*get-opts S*)
  (*get-old-arena S*)
  (*get-occs S*));
   read-all-st (*f' C*) *S*
  }$\rangle$
 end

locale *read-all-param-adder* = *read-all-param-adder-ops f' P*
  for $f' :: \langle 'a \Rightarrow$ *trail-pol* $\Rightarrow$ *arena* $\Rightarrow$
    conflict-option-rel $\Rightarrow$ *nat* $\Rightarrow$ (*nat watcher*) *list list* $\Rightarrow$ *bump-heuristics* $\Rightarrow$
    nat $\Rightarrow$ *conflict-min-cach-l* $\Rightarrow$ *lbd* $\Rightarrow$ *out-learned* $\Rightarrow$ *isasat-stats* $\Rightarrow$ *isasat-restart-heuristics* $\Rightarrow$
   isasat-aivdom $\Rightarrow$ *clss-size* $\Rightarrow$ *opts* $\Rightarrow$ *arena* $\Rightarrow$ *occurences-ref* $\Rightarrow$ $'r$ *nres*$\rangle$ **and**
   $P :: \langle 'a \Rightarrow$ *trail-pol* $\Rightarrow$ *arena* $\Rightarrow$
    conflict-option-rel $\Rightarrow$ *nat* $\Rightarrow$ (*nat watcher*) *list list* $\Rightarrow$ *bump-heuristics* $\Rightarrow$
    nat $\Rightarrow$ *conflict-min-cach-l* $\Rightarrow$ *lbd* $\Rightarrow$ *out-learned* $\Rightarrow$ *isasat-stats* $\Rightarrow$ *isasat-restart-heuristics* $\Rightarrow$
   isasat-aivdom $\Rightarrow$ *clss-size* $\Rightarrow$ *opts* $\Rightarrow$ *arena* $\Rightarrow$ *occurences-ref* $\Rightarrow$ *bool*$\rangle$ +
  fixes *R* **and** *f* **and** *x-assn* $:: \langle 'r \Rightarrow 'q \Rightarrow$ *assn*$\rangle$
  assumes *not-deleted-code-refine*:
   $\langle$(*uncurry17* (λ*a b c d e kf g h i j k l m n ko p q C*. *f C a b c d e kf g h i j k l m n ko p q*),
   uncurry17 (λ*a b c d e kf g h i j k l m n ko p q C*. *f' C a b c d e kf g h i j k l m n ko p q*))
   $\in$ [*uncurry17* (λ*a b c d e kf g h i j k l m n ko p q C*. *P C a b c d e kf g h i j k l m n ko p q*)]$_a$
   trail-pol-fast-assn^k $*_a$
   arena-fast-assn^k $*_a$
  conflict-option-rel-assn^k $*_a$
  sint64-nat-assn^k $*_a$
  watchlist-fast-assn^k $*_a$
  heuristic-bump-assn^k $*_a$
  uint32-nat-assn^k $*_a$

cach-refinement-l-assn^k $*_a$
lbd-assn^k $*_a$
out-learned-assn^k $*_a$
isasat-stats-assn^k $*_a$
heuristic-assn^k $*_a$
aivdom-assn^k $*_a$
lcount-assn^k $*_a$
opts-assn^k $*_a$
arena-fast-assn^k $*_a$ *occs-assn*^{k} $*_a$ (*pure R*)^k $\to$ *x-assn*›
begin

context
begin
lemma *not-deleted-code-refine-tmp*:
‹$\bigwedge C\ C'$. $(C,\ C') \in R \Longrightarrow$ (*uncurry16* (*f C*), *uncurry16* (*f' C'*)) $\in$ [*uncurry16* (*P C'*)]$_a$
trail-pol-fast-assn^k $*_a$
arena-fast-assn^k $*_a$
conflict-option-rel-assn^k $*_a$
sint64-nat-assn^k $*_a$
watchlist-fast-assn^k $*_a$
heuristic-bump-assn^k $*_a$
uint32-nat-assn^k $*_a$
cach-refinement-l-assn^k $*_a$
lbd-assn^k $*_a$
out-learned-assn^k $*_a$
isasat-stats-assn^k $*_a$
heuristic-assn^k $*_a$
aivdom-assn^k $*_a$
lcount-assn^k $*_a$
opts-assn^k $*_a$
arena-fast-assn^k $*_a$ *occs-assn*^{k} $\to$ *x-assn*›
apply (*rule remove-pure-parameter2'*[**where** *R*=*R*])
apply (*rule hfref-cong*[*OF not-deleted-code-refine*])
apply (*auto simp add*: *uncurry-def*)
done
end

lemma (**in** −) *case-isasat-int-split-getter*: ‹*P* (*get-trail-wl-heur S*)
(*get-clauses-wl-heur S*)
(*get-conflict-wl-heur S*)
(*literals-to-update-wl-heur S*)
(*get-watched-wl-heur S*)
(*get-vmtf-heur S*)
(*get-count-max-lvls-heur S*)
(*get-conflict-cach S*)
(*get-lbd S*)
(*get-outlearned-heur S*)
(*get-stats-heur S*)
(*get-heur S*)
(*get-aivdom S*)
(*get-learned-count S*)
(*get-opts S*)
(*get-old-arena S*) (*get-occs S*) = *case-isasat-int P S*›
by (*auto split*: *isasat-int-splits*)

lemma *refine*:

‹(*uncurry* (λ*N C*. *read-all-st-code* (*f C*) *N*),
uncurry (λ*N C'*. *read-all-st* (*f' C'*) *N*))
∈ [*uncurry* (λ*S C*. *P C* (*get-trail-wl-heur S*)
(*get-clauses-wl-heur S*)
(*get-conflict-wl-heur S*)
(*literals-to-update-wl-heur S*)
(*get-watched-wl-heur S*)
(*get-vmtf-heur S*)
(*get-count-max-lvls-heur S*)
(*get-conflict-cach S*)
(*get-lbd S*)
(*get-outlearned-heur S*)
(*get-stats-heur S*)
(*get-heur S*)
(*get-aivdom S*)
(*get-learned-count S*)
(*get-opts S*)
(*get-old-arena S*) (*get-occs S*))]$_a$ *isasat-bounded-assn*^{k} $*_a$ (*pure R*)$^k\to$ *x-assn*›
apply (*rule add-pure-parameter2*)
unfolding *tuple17.case-distrib case-isasat-int-split-getter*
apply (*rule read-all-code-refine*)
apply (*rule not-deleted-code-refine-tmp*)
apply *assumption*
done

lemma *mop-refine*:
‹(*uncurry* (λ*N C*. *read-all-st-code* (*f C*) *N*),
uncurry mop)
∈ *isasat-bounded-assn*^{k} $*_a$ (*pure R*)$^k\to_a$ *x-assn*›
unfolding *mop-def*
apply (*rule refine-ASSERT-move-to-pre*)
apply (*rule refine*)
done
end

locale *read-trail-param-adder* =
fixes *R* **and** *f* **and** *f'* **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** *P*
assumes *not-deleted-code-refine*: ‹(*uncurry* (λ*S C*. *f C S*), *uncurry* (λ*S C*. *f' C S*)) ∈ [*uncurry* (λ*S C*. *P C S*)]$_a$ *trail-pol-fast-assn*^{k} $*_a$ (*pure R*)^k → *x-assn*›
begin

lemma *not-deleted-code-refine'*:
‹(*uncurry17* (λ*M* - - - - - - - - - - - - - - - - - *C*. *f C M*), *uncurry17* (λ*M* - - - - - - - - - - - - - - - - - *C'*. *f' C' M*)) ∈
[*uncurry17* (λ*M* - - - - - - - - - - - - - - - - - *C'*. *P C' M*)]$_a$
trail-pol-fast-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ *conflict-option-rel-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$
watchlist-fast-assn^k $*_a$ *heuristic-bump-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $*_a$ *cach-refinement-l-assn*^{k} $*_a$
lbd-assn^k $*_a$ *out-learned-assn*^{k} $*_a$ *isasat-stats-assn*^{k} $*_a$ *heuristic-assn*^{k} $*_a$
aivdom-assn^k $*_a$ *lcount-assn*^{k} $*_a$ *opts-assn*^{k} $*_a$ *arena-fast-assn*^{k} $*_a$ *occs-assn*^{k} $*_a$ (*pure R*)^k → *x-assn*›
apply (*rule add-pure-parameter2*)
apply (*drule remove-pure-parameter2'*[**where** *f* =*f* **and** *f'*=*f'*, *OF not-deleted-code-refine*])
apply (*drule remove-component-right*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-right*[**where** *X* = *conflict-option-rel-assn*])
apply (*drule remove-component-right*[**where** *X* = *sint64-nat-assn*])
apply (*drule remove-component-right*[**where** *X* = *watchlist-fast-assn*])
apply (*drule remove-component-right*[**where** *X* = *heuristic-bump-assn*])

apply (*drule remove-component-right*[**where** X = *uint32-nat-assn*])
apply (*drule remove-component-right*[**where** X = *cach-refinement-l-assn*])
apply (*drule remove-component-right*[**where** X = *lbd-assn*])
apply (*drule remove-component-right*[**where** X = *out-learned-assn*])
apply (*drule remove-component-right*[**where** X = *isasat-stats-assn*])
apply (*drule remove-component-right*[**where** X = *heuristic-assn*])
apply (*drule remove-component-right*[**where** X = *aivdom-assn*])
apply (*drule remove-component-right*[**where** X = *lcount-assn*])
apply (*drule remove-component-right*[**where** X = *opts-assn*])
apply (*drule remove-component-right*[**where** X = *arena-fast-assn*])
apply (*drule remove-component-right*[**where** X = *occs-assn*])
by (*rule hfref-cong*, *assumption*) *auto*

sublocale *XX* : *read-all-param-adder* **where**
f = ‹($\lambda C\ M$ - - - - - - - - - - - - - - - - -. $f\ C\ M$)› **and**
f' = ‹($\lambda C\ M$ - - - - - - - - - - - - - - - - -. $f'\ C\ M$)› **and**
P = ‹($\lambda C\ M$ - - - - - - - - - - - - - - - - -. $P\ C\ M$)›
by *unfold-locales*
(*rule not-deleted-code-refine'*)

lemmas *refine* = *XX.refine*
lemmas *mop-refine* = *XX.mop-refine*
end

locale *read-arena-param-adder-ops* =
fixes P :: ‹'b ⇒ *arena* ⇒ *bool*› **and** f' :: ‹'b ⇒ *arena-el list* ⇒ 'r *nres*›
begin
definition *mop* **where**
‹*mop* $N\ C$ = *do* {
ASSERT ($P\ C$ (*get-clauses-wl-heur* N));
read-all-st (λ- N - - - - - - - - - - - - - - - -. $f'\ C\ N$) N }›

end

locale *read-arena-param-adder* = *read-arena-param-adder-ops* $P\ f'$
for P :: ‹'b ⇒ *arena* ⇒ *bool*› **and** f' :: ‹'b ⇒ *arena-el list* ⇒ 'r *nres*› +
fixes R :: ‹('a × 'b) *set*› **and** f **and** *x-assn* :: ‹'r ⇒ 'q ⇒ *assn*›
assumes *not-deleted-code-refine*: ‹(*uncurry* ($\lambda S\ C$. $f\ C\ S$), *uncurry* ($\lambda S\ C'$. $f'\ C'\ S$)) ∈ [*uncurry* (λS C. $P\ C\ S$)]$_a$ *arena-fast-assn*^{k} *$_a$ (*pure* R)^k → *x-assn*›
begin

lemma *not-deleted-code-refine'*:
‹(*uncurry17* (λ- M - - - - - - - - - - - - - - - - C. $f\ C\ M$), *uncurry17* (λ- M - - - - - - - - - - - - - - - - C'. $f'\ C'\ M$)) ∈
[*uncurry17* (λ- M - - - - - - - - - - - - - - - - C'. $P\ C'\ M$)]$_a$
trail-pol-fast-assn^k *$_a$ arena-fast-assn*^{k} *$_a$ conflict-option-rel-assn*^{k} *$_a$ sint64-nat-assn*^{k} *$_a$
watchlist-fast-assn^k *$_a$ heuristic-bump-assn*^{k} *$_a$ uint32-nat-assn*^{k} *$_a$ cach-refinement-l-assn*^{k} *$_a$
lbd-assn^k *$_a$ out-learned-assn*^{k} *$_a$ isasat-stats-assn*^{k} *$_a$ heuristic-assn*^{k} *$_a$
aivdom-assn^k *$_a$ lcount-assn*^{k} *$_a$ opts-assn*^{k} *$_a$ arena-fast-assn*^{k} *$_a$ occs-assn*^{k} *$_a$ (*pure* R)^k → *x-assn*›
apply (*rule add-pure-parameter2*)
apply (*drule remove-pure-parameter2'*[**where** f =f **and** f'=f', *OF not-deleted-code-refine*])
apply (*drule remove-component-left*[**where** X = *trail-pol-fast-assn*])
apply (*drule remove-component-right*[**where** X = *conflict-option-rel-assn*])
apply (*drule remove-component-right*[**where** X = *sint64-nat-assn*])

apply (*drule remove-component-right*[**where** X = *watchlist-fast-assn*])
apply (*drule remove-component-right*[**where** X = *heuristic-bump-assn*])
apply (*drule remove-component-right*[**where** X = *uint32-nat-assn*])
apply (*drule remove-component-right*[**where** X = *cach-refinement-l-assn*])
apply (*drule remove-component-right*[**where** X = *lbd-assn*])
apply (*drule remove-component-right*[**where** X = *out-learned-assn*])
apply (*drule remove-component-right*[**where** X = *isasat-stats-assn*])
apply (*drule remove-component-right*[**where** X = *heuristic-assn*])
apply (*drule remove-component-right*[**where** X = *aivdom-assn*])
apply (*drule remove-component-right*[**where** X = *lcount-assn*])
apply (*drule remove-component-right*[**where** X = *opts-assn*])
apply (*drule remove-component-right*[**where** X = *arena-fast-assn*])
apply (*drule remove-component-right*[**where** X = *occs-assn*])
by (*rule hfref-cong*, *assumption*) *auto*

sublocale *XX*: *read-all-param-adder* **where**
$f = \langle(\lambda C$ - M - - - - - - - - - - - - - - - -. $f\ C\ M)\rangle$ **and**
$f' = \langle(\lambda C$ - M - - - - - - - - - - - - - - - -. $f'\ C\ M)\rangle$ **and**
$P = \langle(\lambda C$ - M - - - - - - - - - - - - - - - -. $P\ C\ M)\rangle$
by *unfold-locales*
(*rule not-deleted-code-refine'*)
lemmas *refine* = *XX.refine*

lemma *mop-refine*:
$\langle$(*uncurry* ($\lambda N\ C$. *read-all-st-code* (λ- M - - - - - - - - - - - - - - - -. $f\ C\ M$) N),
uncurry mop)
$\in$ *isasat-bounded-assn*^{k} $*_a$ (*pure* R)$^k \to_a$ *x-assn*$\rangle$
unfolding *mop-def*
apply (*rule refine-ASSERT-move-to-pre*)
apply (*rule refine*)
done

end

locale *read-arena-param-adder0* =
fixes f **and** f' **and** *x-assn* :: $\langle 'r \Rightarrow 'q \Rightarrow$ *assn*$\rangle$ **and** P
assumes *not-deleted-code-refine*: $\langle(f, f') \in [P]_a$ *arena-fast-assn*$^k \to$ *x-assn*$\rangle$
begin

sublocale *XX*: *read-arena-param-adder* **where**
$f = \langle\lambda$- N. $f\ N\rangle$ **and**
$f' = \langle\lambda$- N. $f'\ N\rangle$ **and**
$P = \langle\lambda$-. $P\rangle$ **and**
$R = \langle$*unit-rel*$\rangle$
apply *unfold-locales*
using *not-deleted-code-refine*[*THEN remove-component-right*] **.**

lemmas *refine* = *XX.refine*[*THEN remove-unused-unit-parameter*]
lemmas *mop-refine* = *XX.mop-refine*
end

lemma *merge-second-pure-argument-generalized*:
$\langle$(*uncurry2* f, *uncurry2* f')
$\in$ [*uncurry2* P]$_a$ A $*_a$ (*pure* R)^k $*_a$ (*pure* R')$^k \to$ *x-assn* $\Longrightarrow$
(*uncurry* ($\lambda S\ C$. (*case* C *of* (C, D) $\Rightarrow$ $f\ S\ C\ D$)),
uncurry ($\lambda S\ C'$. (*case* C' *of* (C, D) $\Rightarrow$ $f'\ S\ C\ D$)))

$\in [uncurry$
$(\lambda S\ C.\ (case\ C\ of\ (C,\ D) \Rightarrow P\ S\ C\ D))]_a\ A *_a (pure\ (R \times_f R'))^k \to x\text{-}assn\rangle$
unfolding *hfref-def*
by (*auto simp flip*: *prod-assn-pure-conv*)

lemma *merge-second-pure-argument*:
$\langle(uncurry2\ (\lambda S\ C\ D.\ f\ C\ D\ S),\ uncurry2\ (\lambda S\ C'\ D'.\ f'\ C'\ D'\ S))$
$\in [uncurry2$
$(\lambda S\ C'\ D'.$
$P\ C'\ D'\ S)]_a\ A *_a (pure\ R)^k *_a (pure\ R')^k \to x\text{-}assn \Longrightarrow$
$(uncurry\ (\lambda S\ C.\ (case\ C\ of\ (C,\ D) \Rightarrow f\ C\ D)\ S),$
$uncurry\ (\lambda S\ C'.\ (case\ C'\ of\ (C,\ D) \Rightarrow f'\ C\ D)\ S))$
$\in [uncurry\ (\lambda S\ C.\ (case\ C\ of\ (C,\ D) \Rightarrow P\ C\ D)\ S)]_a\ A *_a (pure\ (R \times_f R'))^k \to x\text{-}assn\rangle$
unfolding *hfref-def*
by (*auto simp flip*: *prod-assn-pure-conv*)

lemma *merge-third-pure-argument'*:
$\langle(uncurry3\ (\lambda S\ C\ D\ E.\ f\ C\ D\ E\ S),\ uncurry3\ (\lambda S\ C'\ D'\ E'.\ f'\ C'\ D'\ E'\ S))$
$\in [uncurry3$
$(\lambda S\ C'\ D'\ E'.$
$P\ C'\ D'\ E'\ S)]_a\ A *_a (pure\ R)^k *_a (pure\ R')^k\ *_a (pure\ R'')^k \to x\text{-}assn \Longrightarrow$
$(uncurry2\ (\lambda S\ E\ C.\ (case\ C\ of\ (C,\ D) \Rightarrow f\ E\ C\ D)\ S),$
$uncurry2\ (\lambda S\ E'\ C'.\ (case\ C'\ of\ (C,\ D) \Rightarrow f'\ E'\ C\ D)\ S))$
$\in [uncurry2$
$(\lambda S\ E\ C.\ (case\ C\ of\ (C,\ D) \Rightarrow P\ E\ C\ D)$
$S)]_a\ A *_a (pure\ R)^k *_a\ (pure\ (R' \times_f R''))^k \to x\text{-}assn\rangle$
unfolding *hfref-def*
by (*auto simp flip*: *prod-assn-pure-conv*)

abbreviation *read-arena-wl-heur-code* :: ‹-› **where**
$\langle read\text{-}arena\text{-}wl\text{-}heur\text{-}code\ f \equiv read\text{-}all\text{-}st\text{-}code\ \ (\lambda\text{-}\ M\ \text{- - - - - - - - - - - - - - - -}.\ f\ M)\rangle$
abbreviation *read-arena-wl-heur* :: ‹-› **where**
$\langle read\text{-}arena\text{-}wl\text{-}heur\ f \equiv read\text{-}all\text{-}st\ \ (\lambda\text{-}\ M\ \text{- - - - - - - - - - - - - - - -}.\ f\ M)\rangle$

locale *read-arena-param-adder2-twoargs-ops* =
fixes
$f' :: \langle 'b \Rightarrow 'd \Rightarrow arena \Rightarrow 'r\ nres\rangle$ **and**
$P :: \langle 'b \Rightarrow 'd \Rightarrow arena \Rightarrow bool\rangle$
begin
definition *mop* **where**
$\langle mop\ N\ C\ C' = do\ \{$
$ASSERT\ (P\ C\ C'\ (get\text{-}clauses\text{-}wl\text{-}heur\ N));$
$read\text{-}arena\text{-}wl\text{-}heur\ (\lambda N.\ f'\ C\ C'\ N)\ N$
$\}\rangle$
end

locale *read-arena-param-adder2-twoargs* =
read-arena-param-adder2-twoargs-ops $f'\ P$
for $f' :: \langle 'b \Rightarrow 'd \Rightarrow arena \Rightarrow 'r\ nres\rangle$ **and** P +
fixes R **and** R' **and** f **and** $x\text{-}assn :: \langle 'r \Rightarrow 'q \Rightarrow assn\rangle$
assumes *not-deleted-code-refine*:
$\langle(uncurry2\ (\lambda S\ C\ D.\ f\ C\ D\ S),\ uncurry2\ (\lambda S\ C'\ D'.\ f'\ C'\ D'\ S)) \in$
$[uncurry2\ (\lambda S\ C'\ D'.\ P\ C'\ D'\ S)]_a\ arena\text{-}fast\text{-}assn^k *_a (pure\ R)^k *_a (pure\ R')^k \to x\text{-}assn\rangle$
begin

sublocale *XX*: *read-arena-param-adder* **where**
f = ‹λ(*C*,*D*) *N*. *f C D N*› **and**
f' = ‹λ(*C*,*D*) *N*. *f' C D N*› **and**
P = ‹λ(*C*,*D*) *N*. *P C D N*› **and**
R = ‹*R* ×*f* *R'*›
apply *unfold-locales*
using *not-deleted-code-refine*[*THEN merge-second-pure-argument*] **.**

lemma *refine*:
‹(*uncurry2* (λ*N C D*. *read-arena-wl-heur-code* (*f C D*) *N*),
uncurry2 (λ*N C' D*. *read-arena-wl-heur* (*f' C' D*) *N*))
∈ [*uncurry2* (λ*S C D*. *P C D* (*get-clauses-wl-heur S*))]*a* *isasat-bounded-assn*^{k} **a* (*pure R*)^k **a* (*pure R'*)^k → *x-assn*›
by (*rule XX.refine*[*THEN split-snd-pure-arg*, *unfolded prod.case*])

lemma *mop-refine*:
‹(*uncurry2* (λ*N C D*. *read-arena-wl-heur-code* (λ*N*. *f C D N*) *N*), *uncurry2 mop*) ∈ *isasat-bounded-assn*^{k} **a* (*pure R*)^k **a* (*pure R'*)^k →*a* *x-assn*›
unfolding *mop-def*
apply (*rule refine-ASSERT-move-to-pre2*)
apply (*rule refine*[*unfolded comp-def*])
done

end

abbreviation *read-conflict-wl-heur-code* :: ‹-› **where**
‹*read-conflict-wl-heur-code f* ≡ *read-all-st-code* (λ- - *M* - - - - - - - - - - - - - - -. *f M*)›
abbreviation *read-conflict-wl-heur* :: ‹-› **where**
‹*read-conflict-wl-heur f* ≡ *read-all-st* (λ- - *M* - - - - - - - - - - - - - - -. *f M*)›

locale *read-conflict-param-adder* =
fixes *R* **and** *f* **and** *f'* **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** *P*
assumes *not-deleted-code-refine*: ‹(*uncurry* (λ*S C*. *f C S*), *uncurry* (λ*S C'*. *f' C' S*)) ∈ [*uncurry* (λ*S C*. *P C S*)]*a* *conflict-option-rel-assn*^{k} **a* (*pure R*)^k → *x-assn*›
begin

lemma *not-deleted-code-refine'*:
shows ‹
(*uncurry17* (λ- - *M* - - - - - - - - - - - - - - - *C*. *f C M*), *uncurry17* (λ- - *M* - - - - - - - - - - - - - - - *C'*. *f' C' M*)) ∈
[*uncurry17* (λ- - *M* - - - - - - - - - - - - - - - *C'*. *P C' M*)]*a*
trail-pol-fast-assn^k **a* *arena-fast-assn*^{k} **a* *conflict-option-rel-assn*^{k} **a* *sint64-nat-assn*^{k} **a*
watchlist-fast-assn^k **a* *heuristic-bump-assn*^{k} **a* *uint32-nat-assn*^{k} **a* *cach-refinement-l-assn*^{k} **a*
lbd-assn^k **a* *out-learned-assn*^{k} **a* *isasat-stats-assn*^{k} **a* *heuristic-assn*^{k} **a*
aivdom-assn^k **a* *lcount-assn*^{k} **a* *opts-assn*^{k} **a* *arena-fast-assn*^{k} **a* *occs-assn*^{k} **a* (*pure R*)^k → *x-assn*›
apply (*rule add-pure-parameter2*)
apply (*drule remove-pure-parameter2'*[**where** *f* =*f* **and** *f'*=*f'*, *OF not-deleted-code-refine*])
apply (*drule remove-component-left*[**where** *X* = *trail-pol-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-right*[**where** *X* = *sint64-nat-assn*])
apply (*drule remove-component-right*[**where** *X* = *watchlist-fast-assn*])
apply (*drule remove-component-right*[**where** *X* = *heuristic-bump-assn*])
apply (*drule remove-component-right*[**where** *X* = *uint32-nat-assn*])
apply (*drule remove-component-right*[**where** *X* = *cach-refinement-l-assn*])

apply (*drule remove-component-right*[**where** X = *lbd-assn*])
apply (*drule remove-component-right*[**where** X = *out-learned-assn*])
apply (*drule remove-component-right*[**where** X = *isasat-stats-assn*])
apply (*drule remove-component-right*[**where** X = *heuristic-assn*])
apply (*drule remove-component-right*[**where** X = *aivdom-assn*])
apply (*drule remove-component-right*[**where** X = *lcount-assn*])
apply (*drule remove-component-right*[**where** X = *opts-assn*])
apply (*drule remove-component-right*[**where** X = *arena-fast-assn*])
apply (*drule remove-component-right*[**where** X = *occs-assn*])
by (*rule hfref-cong*, *assumption*) *auto*

sublocale *XX* : *read-all-param-adder* **where**
f = ‹$(\lambda C$ - - M - - - - - - - - - - - - - - -. $f\ C\ M)$› **and**
f' = ‹$(\lambda C$ - - M - - - - - - - - - - - - - - -. $f'\ C\ M)$› **and**
P = ‹$(\lambda C$ - - M - - - - - - - - - - - - - - -. $P\ C\ M)$›
by *unfold-locales*
(*rule not-deleted-code-refine'*)

lemmas *refine* = *XX.refine*
end

locale *read-conflict-param-adder0* =
fixes f **and** f' **and** *x-assn* :: ‹$'r \Rightarrow 'q \Rightarrow$ *assn*› **and** P
assumes *not-deleted-code-refine*: ‹$(f, f') \in [P]_a$ *conflict-option-rel-assn*$^k \to$ *x-assn*›
begin
sublocale *XX*: *read-conflict-param-adder* **where**
f = ‹λ-. f› **and**
f' = ‹λ-. f'› **and**
P = ‹λ-. P› **and**
R = *unit-rel*
apply *unfold-locales*
using *not-deleted-code-refine*[*THEN remove-component-right*] **.**

lemmas *refine* = *XX.refine*[*THEN remove-unused-unit-parameter*]
end

abbreviation *read-literals-to-update-wl-heur-code* :: ‹-› **where**
‹*read-literals-to-update-wl-heur-code* $f \equiv$ *read-all-st-code* (λ- - - M - - - - - - - - - - - - - -. $f\ M$)›
abbreviation *read-literals-to-update-wl-heur* :: ‹-› **where**
‹*read-literals-to-update-wl-heur* $f \equiv$ *read-all-st* (λ- - - M - - - - - - - - - - - - - -. $f\ M$)›

locale *read-literals-to-update-param-adder* =
fixes R **and** f **and** f' **and** *x-assn* :: ‹$'r \Rightarrow 'q \Rightarrow$ *assn*› **and** P
assumes *not-deleted-code-refine*: ‹(*uncurry* ($\lambda S\ C$. $f\ C\ S$), *uncurry* ($\lambda S\ C'$. $f'\ C'\ S$)) $\in$ [*uncurry* (λS C. $P\ C\ S$)]$_a$ *sint64-nat-assn*^{k} $*_a$ (*pure* R)$^k \to$ *x-assn*›
begin

lemma *not-deleted-code-refine'*:
shows ‹
(*uncurry17* (λ- - - M - - - - - - - - - - - - - - C. $f\ C\ M$), *uncurry17* (λ- - - M - - - - - - - - - - - - - - C'. $f'\ C'\ M$)) $\in$
[*uncurry17* (λ- - - M - - - - - - - - - - - - - - C'. $P\ C'\ M$)]$_a$
trail-pol-fast-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ *conflict-option-rel-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$
watchlist-fast-assn^k $*_a$ *heuristic-bump-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $*_a$ *cach-refinement-l-assn*^{k} $*_a$
lbd-assn^k $*_a$ *out-learned-assn*^{k} $*_a$ *isasat-stats-assn*^{k} $*_a$ *heuristic-assn*^{k} $*_a$

aivdom-assn^k $*_a$ *lcount-assn*^{k} $*_a$ *opts-assn*^{k} $*_a$ *arena-fast-assn*^{k} $*_a$ *occs-assn*^{k} $*_a$ (*pure R*)^k $\to$ *x-assn*›
apply (*rule add-pure-parameter2*)
apply (*drule remove-pure-parameter2'*[**where** *f* =*f* **and** *f'*=*f'*, *OF not-deleted-code-refine*])
apply (*drule remove-component-left*[**where** *X* = *trail-pol-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *conflict-option-rel-assn*])
apply (*drule remove-component-right*[**where** *X* = *watchlist-fast-assn*])
apply (*drule remove-component-right*[**where** *X* = *heuristic-bump-assn*])
apply (*drule remove-component-right*[**where** *X* = *uint32-nat-assn*])
apply (*drule remove-component-right*[**where** *X* = *cach-refinement-l-assn*])
apply (*drule remove-component-right*[**where** *X* = *lbd-assn*])
apply (*drule remove-component-right*[**where** *X* = *out-learned-assn*])
apply (*drule remove-component-right*[**where** *X* = *isasat-stats-assn*])
apply (*drule remove-component-right*[**where** *X* = *heuristic-assn*])
apply (*drule remove-component-right*[**where** *X* = *aivdom-assn*])
apply (*drule remove-component-right*[**where** *X* = *lcount-assn*])
apply (*drule remove-component-right*[**where** *X* = *opts-assn*])
apply (*drule remove-component-right*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-right*[**where** *X* = *occs-assn*])
by (*rule hfref-cong*, *assumption*) *auto*

sublocale *XX* : *read-all-param-adder* **where**
f = ‹(λ*C* - - - *M* - - - - - - - - - - - - -. *f C M*)› **and**
f' = ‹(λ*C* - - - *M* - - - - - - - - - - - - -. *f' C M*)› **and**
P = ‹(λ*C* - - - *M* - - - - - - - - - - - - -. *P C M*)›
by *unfold-locales*
(*rule not-deleted-code-refine'*)

lemmas *refine* = *XX.refine*
end

locale *read-literals-to-update-param-adder0* =
fixes *f* **and** *f'* **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** *P*
assumes *not-deleted-code-refine*: ‹(*f*, *f'*) ∈ [*P*]$_a$ *sint64-nat-assn*^{k} $\to$ *x-assn*›
begin
sublocale *XX*: *read-literals-to-update-param-adder* **where**
f = ‹λ-. *f*› **and**
f' = ‹λ-. *f'*› **and**
P = ‹λ-. *P*›
apply *unfold-locales*
using *not-deleted-code-refine*[*THEN remove-component-right*] **.**

lemmas *refine* = *XX.refine*[*THEN remove-unused-unit-parameter*]
end

abbreviation *read-watchlist-wl-heur-code* :: ‹-› **where**
‹*read-watchlist-wl-heur-code f* ≡ *read-all-st-code* (λ- - - - *M* - - - - - - - - - - - -. *f M*)›
abbreviation *read-watchlist-wl-heur* :: ‹-› **where**
‹*read-watchlist-wl-heur f* ≡ *read-all-st* (λ- - - - *M* - - - - - - - - - - - -. *f M*)›

locale *read-watchlist-param-adder* =
fixes *R* **and** *f* **and** *f'* **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** *P*
assumes *not-deleted-code-refine*:
‹(*uncurry* (λ*S C*. *f C S*), *uncurry* (λ*S C'*. *f' C' S*)) ∈ [*uncurry* (λ*S C*. *P C S*)]$_a$ *watchlist-fast-assn*^{k}

$*_a$ (*pure R*)^k → *x-assn*›
begin

lemma *not-deleted-code-refine′*:
shows ‹
(*uncurry17* (λ- - - - *M* - - - - - - - - - - - - *C*. *f C M*), *uncurry17* (λ- - - - *M* - - - - - - - - - - - - *C′*. *f′ C′ M*)) ∈
[*uncurry17* (λ- - - - *M* - - - - - - - - - - - - *C′*. *P C′ M*)]$_a$
trail-pol-fast-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ *conflict-option-rel-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$
watchlist-fast-assn^k $*_a$ *heuristic-bump-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $*_a$ *cach-refinement-l-assn*^{k} $*_a$
lbd-assn^k $*_a$ *out-learned-assn*^{k} $*_a$ *isasat-stats-assn*^{k} $*_a$ *heuristic-assn*^{k} $*_a$
aivdom-assn^k $*_a$ *lcount-assn*^{k} $*_a$ *opts-assn*^{k} $*_a$ *arena-fast-assn*^{k} $*_a$ *occs-assn*^{k} $*_a$ (*pure R*)^k → *x-assn*›
apply (*rule add-pure-parameter2*)
apply (*drule remove-pure-parameter2′*[**where** *f* =*f* **and** *f′*=*f′*, *OF not-deleted-code-refine*])
apply (*drule remove-component-left*[**where** *X* = *trail-pol-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *conflict-option-rel-assn*])
apply (*drule remove-component-middle*[**where** *X* = *sint64-nat-assn*])
apply (*drule remove-component-right*[**where** *X* = *heuristic-bump-assn*])
apply (*drule remove-component-right*[**where** *X* = *uint32-nat-assn*])
apply (*drule remove-component-right*[**where** *X* = *cach-refinement-l-assn*])
apply (*drule remove-component-right*[**where** *X* = *lbd-assn*])
apply (*drule remove-component-right*[**where** *X* = *out-learned-assn*])
apply (*drule remove-component-right*[**where** *X* = *isasat-stats-assn*])
apply (*drule remove-component-right*[**where** *X* = *heuristic-assn*])
apply (*drule remove-component-right*[**where** *X* = *aivdom-assn*])
apply (*drule remove-component-right*[**where** *X* = *lcount-assn*])
apply (*drule remove-component-right*[**where** *X* = *opts-assn*])
apply (*drule remove-component-right*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-right*[**where** *X* = *occs-assn*])
by (*rule hfref-cong*, *assumption*) *auto*

sublocale *XX* : *read-all-param-adder* **where**
f = ‹(λ*C* - - - - *M* - - - - - - - - - - - -. *f C M*)› **and**
f′ = ‹(λ*C* - - - - *M* - - - - - - - - - - - -. *f′ C M*)› **and**
P = ‹(λ*C* - - - - *M* - - - - - - - - - - - -. *P C M*)›
by *unfold-locales*
(*rule not-deleted-code-refine′*)

lemmas *refine* = *XX.refine*
lemmas *mop-refine* = *XX.mop-refine*
end

locale *read-watchlist-param-adder0* =
fixes *f* **and** *f′* **and** *x-assn* :: ‹′*r* ⇒ ′*q* ⇒ *assn*› **and** *P*
assumes *not-deleted-code-refine*: ‹(*f*, *f′*) ∈ [*P*]$_a$ *watchlist-fast-assn*^{k} → *x-assn*›
begin

sublocale *XX*: *read-watchlist-param-adder* **where**
f = ‹λ-. *f*› **and**
f′ = ‹λ-. *f′*› **and**
P = ‹λ-. *P*›
apply *unfold-locales*
using *not-deleted-code-refine*[*THEN remove-component-right*] **.**

lemmas *refine* = *XX.refine*[*THEN remove-unused-unit-parameter*]

end

locale *read-watchlist-param-adder-twoargs* =
  fixes R **and** R' **and** f **and** f' **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** P
  assumes *not-deleted-code-refine*:
    ‹(*uncurry2* (λ*S C D*. *f C D S*), *uncurry2* (λ*S C' D'*. *f' C' D' S*)) ∈
    [*uncurry2* (λ*S C' D'*. *P C' D' S*)]$_a$ *watchlist-fast-assn*^{k} $*_a$ (*pure R*)^k $*_a$ (*pure R'*)^k → *x-assn*›
begin

sublocale *XX*: *read-watchlist-param-adder* **where**
  f = ‹λ(*C*,*D*). *f C D*› **and**
  f' = ‹λ(*C*,*D*). *f' C D*› **and**
  P = ‹λ(*C*,*D*). *P C D*› **and**
  R = ‹R ×$_f$ R'›
  apply *unfold-locales*
  using *not-deleted-code-refine*[*THEN merge-second-pure-argument*] **.**

lemma *refine*:
  ‹(*uncurry2* (λ*N C D*. *read-watchlist-wl-heur-code* (*f C D*) *N*),
    uncurry2 (λ*N C' D'*. *read-watchlist-wl-heur* (*f' C' D'*) *N*))
  ∈ [*uncurry2* (λ*S C D*. *P C D* (*get-watched-wl-heur S*))]$_a$
    isasat-bounded-assn^k $*_a$ (*pure R*)^k $*_a$ (*pure R'*)^k→ *x-assn*›
  by (*rule XX.refine*[*THEN split-snd-pure-arg*, *unfolded prod.case*])

lemmas *mop-refine* = *XX.XX.mop-refine*
end

abbreviation *read-vmtf-wl-heur-code* :: ‹-› **where**
  ‹*read-vmtf-wl-heur-code f* ≡ *read-all-st-code* (λ- - - - - *M* - - - - - - - - - - -. *f M*)›
abbreviation *read-vmtf-wl-heur* :: ‹-› **where**
  ‹*read-vmtf-wl-heur f* ≡ *read-all-st* (λ- - - - - *M* - - - - - - - - - - -. *f M*)›

locale *read-vmtf-param-adder* =
  fixes f **and** f' **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** P **and** R
  assumes *not-deleted-code-refine*: ‹(*uncurry* (λ*S C*. *f C S*), *uncurry* (λ*S C'*. *f' C' S*)) ∈ [*uncurry* (λ*S C*. *P C S*)]$_a$ *heuristic-bump-assn*^{k} $*_a$ (*pure R*)^k → *x-assn*›
begin

lemma *not-deleted-code-refine'*:
  shows ‹
  (*uncurry17* (λ- - - - - *M* - - - - - - - - - - - *C*. *f C M*), *uncurry17* (λ- - - - - *M* - - - - - - - - - - - *C'*. *f' C' M*)) ∈
  [*uncurry17* (λ- - - - - *M* - - - - - - - - - - - *C'*. *P C' M*)]$_a$
   trail-pol-fast-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ *conflict-option-rel-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$
  watchlist-fast-assn^k $*_a$ *heuristic-bump-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $*_a$ *cach-refinement-l-assn*^{k} $*_a$
  lbd-assn^k $*_a$ *out-learned-assn*^{k} $*_a$ *isasat-stats-assn*^{k} $*_a$ *heuristic-assn*^{k} $*_a$
  aivdom-assn^k $*_a$ *lcount-assn*^{k} $*_a$ *opts-assn*^{k} $*_a$ *arena-fast-assn*^{k} $*_a$ *occs-assn*^{k} $*_a$ (*pure R*)^k → *x-assn*›
  apply (*rule add-pure-parameter2*)
  apply (*drule remove-pure-parameter2'*[**where** f =f **and** f'=f', *OF not-deleted-code-refine*])
  apply (*drule remove-component-left*[**where** *X* = *trail-pol-fast-assn*])

apply (*drule remove-component-middle*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *conflict-option-rel-assn*])
apply (*drule remove-component-middle*[**where** *X* = *sint64-nat-assn*])
apply (*drule remove-component-middle*[**where** *X* = *watchlist-fast-assn*])
apply (*drule remove-component-right*[**where** *X* = *uint32-nat-assn*])
apply (*drule remove-component-right*[**where** *X* = *cach-refinement-l-assn*])
apply (*drule remove-component-right*[**where** *X* = *lbd-assn*])
apply (*drule remove-component-right*[**where** *X* = *out-learned-assn*])
apply (*drule remove-component-right*[**where** *X* = *isasat-stats-assn*])
apply (*drule remove-component-right*[**where** *X* = *heuristic-assn*])
apply (*drule remove-component-right*[**where** *X* = *aivdom-assn*])
apply (*drule remove-component-right*[**where** *X* = *lcount-assn*])
apply (*drule remove-component-right*[**where** *X* = *opts-assn*])
apply (*drule remove-component-right*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-right*[**where** *X* = *occs-assn*])
by (*rule hfref-cong*, *assumption*) *auto*

sublocale *XX* : *read-all-param-adder* **where**
f = ‹(λ*C* - - - - - *M* - - - - - - - - - - -. *f C M*)› **and**
f′ = ‹(λ*C* - - - - - *M* - - - - - - - - - - -. *f*′ *C M*)› **and**
P = ‹(λ*C* - - - - - *M* - - - - - - - - - - -. *P C M*)›
by *unfold-locales*
(*rule not-deleted-code-refine*′)

lemmas *refine* = *XX.refine*
end

locale *read-vmtf-param-adder0* =
fixes *f* **and** *f*′ **and** *x-assn* :: ‹′*r* ⇒ ′*q* ⇒ *assn*› **and** *P*
assumes *not-deleted-code-refine*: ‹(*f*, *f*′) ∈ [*P*]$_a$ *heuristic-bump-assn*^{k} → *x-assn*›
begin

sublocale *XX*: *read-vmtf-param-adder* **where**
f = ‹λ-. *f*› **and**
f′ = ‹λ-. *f*′› **and**
P = ‹λ-. *P*›
apply *unfold-locales*
using *not-deleted-code-refine*[*THEN remove-component-right*] **.**

lemmas *refine* = *XX.refine*[*THEN remove-unused-unit-parameter*]

end

abbreviation *read-ccount-wl-heur-code* :: ‹- ⇒ *twl-st-wll-trail-fast2* ⇒ -› **where**
‹*read-ccount-wl-heur-code f* ≡ *read-all-st-code* (λ- - - - - *M* - - - - - - - - - - -. *f M*)›
abbreviation *read-ccount-wl-heur* :: ‹- ⇒ *isasat* ⇒ -› **where**
‹*read-ccount-wl-heur f* ≡ *read-all-st* (λ- - - - - *M* - - - - - - - - - - -. *f M*)›

locale *read-ccount-param-adder* =
fixes *f* **and** *f*′ **and** *x-assn* :: ‹′*r* ⇒ ′*q* ⇒ *assn*› **and** *P* **and** *R*
assumes *not-deleted-code-refine*: ‹(*uncurry* (λ*S C*. *f C S*), *uncurry* (λ*S C*′. *f*′ *C*′ *S*)) ∈ [*uncurry* (λ*S C*. *P C S*)]$_a$ *uint32-nat-assn*^{k} ∗$_a$ (*pure R*)^k → *x-assn*›
begin

lemma *not-deleted-code-refine′*:
shows ‹
(*uncurry17* (λ- - - - - - *M* - - - - - - - - - - C. *f C M*), *uncurry17* (λ- - - - - - *M* - - - - - - - - - - C′. *f′ C′ M*)) ∈
[*uncurry17* (λ- - - - - - *M* - - - - - - - - - - C′. *P C′ M*)]$_a$
trail-pol-fast-assn^k *$*_a$ *arena-fast-assn*^{k} *$*_a$ *conflict-option-rel-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} *$*_a$
watchlist-fast-assn^k *$*_a$ *heuristic-bump-assn*^{k} *$*_a$ *uint32-nat-assn*^{k} *$*_a$ *cach-refinement-l-assn*^{k} *$*_a$
lbd-assn^k *$*_a$ *out-learned-assn*^{k} *$*_a$ *isasat-stats-assn*^{k} *$*_a$ *heuristic-assn*^{k} *$*_a$
aivdom-assn^k *$*_a$ *lcount-assn*^{k} *$*_a$ *opts-assn*^{k} *$*_a$ *arena-fast-assn*^{k} *$*_a$ *occs-assn*^{k} *$*_a$ (*pure R*)^k → *x-assn*›
apply (*rule add-pure-parameter2*)
apply (*drule remove-pure-parameter2′*[**where** *f* =*f* **and** *f′*=*f′*, *OF not-deleted-code-refine*])
apply (*drule remove-component-left*[**where** *X* = *trail-pol-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *conflict-option-rel-assn*])
apply (*drule remove-component-middle*[**where** *X* = *sint64-nat-assn*])
apply (*drule remove-component-middle*[**where** *X* = *watchlist-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *heuristic-bump-assn*])
apply (*drule remove-component-right*[**where** *X* = *cach-refinement-l-assn*])
apply (*drule remove-component-right*[**where** *X* = *lbd-assn*])
apply (*drule remove-component-right*[**where** *X* = *out-learned-assn*])
apply (*drule remove-component-right*[**where** *X* = *isasat-stats-assn*])
apply (*drule remove-component-right*[**where** *X* = *heuristic-assn*])
apply (*drule remove-component-right*[**where** *X* = *aivdom-assn*])
apply (*drule remove-component-right*[**where** *X* = *lcount-assn*])
apply (*drule remove-component-right*[**where** *X* = *opts-assn*])
apply (*drule remove-component-right*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-right*[**where** *X* = *occs-assn*])
by (*rule hfref-cong*, *assumption*)
(*auto intro*!: *ext*)

sublocale *XX* : *read-all-param-adder* **where**
f = ‹(λ*C* - - - - - - *M* - - - - - - - - - -. *f C M*)› **and**
f′ = ‹(λ*C* - - - - - - *M* - - - - - - - - - -. *f′ C M*)› **and**
P = ‹(λ*C* - - - - - - *M* - - - - - - - - - -. *P C M*)›
by *unfold-locales*
(*rule not-deleted-code-refine′*)

lemmas *refine* = *XX.refine*
end

locale *read-ccount-param-adder0* =
fixes *f* **and** *f′* **and** *x-assn* :: ‹′*r* ⇒ ′*q* ⇒ *assn*› **and** *P*
assumes *not-deleted-code-refine*: ‹(*f*, *f′*) ∈ [*P*]$_a$ *uint32-nat-assn*^{k} → *x-assn*›
begin

sublocale *XX*: *read-ccount-param-adder* **where**
f = ‹λ-. *f*› **and**
f′ = ‹λ-. *f′*› **and**
P = ‹λ-. *P*›
apply *unfold-locales*
using *not-deleted-code-refine*[*THEN remove-component-right*] **.**

lemmas *refine* = *XX.refine*[*THEN remove-unused-unit-parameter*]
end

abbreviation *read-ccach-wl-heur-code* :: ‹- ⇒ *twl-st-wll-trail-fast2* ⇒ -› **where**
‹*read-ccach-wl-heur-code f* ≡ *read-all-st-code* (λ- - - - - - - *M* - - - - - - - - - -. *f M*)›
abbreviation *read-ccach-wl-heur* :: ‹- ⇒ *isasat* ⇒ -› **where**
‹*read-ccach-wl-heur f* ≡ *read-all-st* (λ- - - - - - - *M* - - - - - - - - - -. *f M*)›

locale *read-ccach-param-adder* =
fixes *f* **and** *f'* **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** *P* **and** *R*
assumes *not-deleted-code-refine*: ‹(*uncurry* (λ*S C*. *f C S*), *uncurry* (λ*S C'*. *f' C' S*)) ∈ [*uncurry* (λ*S C*. *P C S*)]$_a$ *cach-refinement-l-assn*^{k} *$*_a$ (*pure R*)^k → *x-assn*›
begin

lemma *not-deleted-code-refine'*:
shows ‹
(*uncurry17* (λ- - - - - - - *M* - - - - - - - - - *C*. *f C M*), *uncurry17* (λ- - - - - - - *M* - - - - - - - - - *C'*. *f' C' M*)) ∈
[*uncurry17* (λ- - - - - - - *M* - - - - - - - - - *C'*. *P C' M*)]$_a$
trail-pol-fast-assn^k *$*_a$ *arena-fast-assn*^{k} *$*_a$ *conflict-option-rel-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} *$*_a$
watchlist-fast-assn^k *$*_a$ *heuristic-bump-assn*^{k} *$*_a$ *uint32-nat-assn*^{k} *$*_a$ *cach-refinement-l-assn*^{k} *$*_a$
lbd-assn^k *$*_a$ *out-learned-assn*^{k} *$*_a$ *isasat-stats-assn*^{k} *$*_a$ *heuristic-assn*^{k} *$*_a$
aivdom-assn^k *$*_a$ *lcount-assn*^{k} *$*_a$ *opts-assn*^{k} *$*_a$ *arena-fast-assn*^{k} *$*_a$ *occs-assn*^{k} *$*_a$ (*pure R*)^k → *x-assn*›
apply (*rule add-pure-parameter2*)
apply (*drule remove-pure-parameter2'*[**where** *f* =*f* **and** *f'*=*f'*, *OF not-deleted-code-refine*])
apply (*drule remove-component-left*[**where** *X* = *trail-pol-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *conflict-option-rel-assn*])
apply (*drule remove-component-middle*[**where** *X* = *sint64-nat-assn*])
apply (*drule remove-component-middle*[**where** *X* = *watchlist-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *heuristic-bump-assn*])
apply (*drule remove-component-middle*[**where** *X* = *uint32-nat-assn*])
apply (*drule remove-component-right*[**where** *X* = *lbd-assn*])
apply (*drule remove-component-right*[**where** *X* = *out-learned-assn*])
apply (*drule remove-component-right*[**where** *X* = *isasat-stats-assn*])
apply (*drule remove-component-right*[**where** *X* = *heuristic-assn*])
apply (*drule remove-component-right*[**where** *X* = *aivdom-assn*])
apply (*drule remove-component-right*[**where** *X* = *lcount-assn*])
apply (*drule remove-component-right*[**where** *X* = *opts-assn*])
apply (*drule remove-component-right*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-right*[**where** *X* = *occs-assn*])
by (*rule hfref-cong*, *assumption*) *auto*

sublocale *XX* : *read-all-param-adder* **where**
f = ‹(λ*C* - - - - - - - *M* - - - - - - - - - -. *f C M*)› **and**
f' = ‹(λ*C* - - - - - - - *M* - - - - - - - - - -. *f' C M*)› **and**
P = ‹(λ*C* - - - - - - - *M* - - - - - - - - - -. *P C M*)›
by *unfold-locales*
(*rule not-deleted-code-refine'*)

lemmas *refine* = *XX.refine*
end

abbreviation *read-lbd-wl-heur-code* :: ‹- ⇒ *twl-st-wll-trail-fast2* ⇒ -› **where**
‹*read-lbd-wl-heur-code f* ≡ *read-all-st-code* (λ- - - - - - - - *M* - - - - - - - - -. *f M*)›
abbreviation *read-lbd-wl-heur* :: ‹- ⇒ *isasat* ⇒ -› **where**
‹*read-lbd-wl-heur f* ≡ *read-all-st* (λ- - - - - - - - *M* - - - - - - - - -. *f M*)›

locale *read-lbd-param-adder* =
fixes f **and** f' **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** P **and** R
assumes *not-deleted-code-refine*: ‹(*uncurry* (λ*S C*. *f C S*), *uncurry* (λ*S C'*. *f' C' S*)) ∈ [*uncurry* (λ*S C*. *P C S*)]$_a$ *lbd-assn*^{k} *$*_a$ (*pure R*)^k → *x-assn*›
begin

lemma *not-deleted-code-refine'*:
shows ‹
(*uncurry17* (λ- - - - - - - - *M* - - - - - - - - *C*. *f C M*), *uncurry17* (λ- - - - - - - - *M* - - - - - - - - *C'*. *f' C' M*)) ∈
[*uncurry17* (λ- - - - - - - - *M* - - - - - - - - *C'*. *P C' M*)]$_a$
trail-pol-fast-assn^k *$*_a$ *arena-fast-assn*^{k} *$*_a$ *conflict-option-rel-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} *$*_a$
watchlist-fast-assn^k *$*_a$ *heuristic-bump-assn*^{k} *$*_a$ *uint32-nat-assn*^{k} *$*_a$ *cach-refinement-l-assn*^{k} *$*_a$
lbd-assn^k *$*_a$ *out-learned-assn*^{k} *$*_a$ *isasat-stats-assn*^{k} *$*_a$ *heuristic-assn*^{k} *$*_a$
aivdom-assn^k *$*_a$ *lcount-assn*^{k} *$*_a$ *opts-assn*^{k} *$*_a$ *arena-fast-assn*^{k} *$*_a$ *occs-assn*^{k} *$*_a$ (*pure R*)^k → *x-assn*›
apply (*rule add-pure-parameter2*)
apply (*drule remove-pure-parameter2'*[**where** *f* =*f* **and** *f'*=*f'*, *OF not-deleted-code-refine*])
apply (*drule remove-component-left*[**where** *X* = *trail-pol-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *conflict-option-rel-assn*])
apply (*drule remove-component-middle*[**where** *X* = *sint64-nat-assn*])
apply (*drule remove-component-middle*[**where** *X* = *watchlist-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *heuristic-bump-assn*])
apply (*drule remove-component-middle*[**where** *X* = *uint32-nat-assn*])
apply (*drule remove-component-middle*[**where** *X* = *cach-refinement-l-assn*])
apply (*drule remove-component-right*[**where** *X* = *out-learned-assn*])
apply (*drule remove-component-right*[**where** *X* = *isasat-stats-assn*])
apply (*drule remove-component-right*[**where** *X* = *heuristic-assn*])
apply (*drule remove-component-right*[**where** *X* = *aivdom-assn*])
apply (*drule remove-component-right*[**where** *X* = *lcount-assn*])
apply (*drule remove-component-right*[**where** *X* = *opts-assn*])
apply (*drule remove-component-right*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-right*[**where** *X* = *occs-assn*])
by (*rule hfref-cong*, *assumption*) *auto*

sublocale *XX* : *read-all-param-adder* **where**
f = ‹(λ*C* - - - - - - - - *M* - - - - - - - - -. *f C M*)› **and**
f' = ‹(λ*C* - - - - - - - - *M* - - - - - - - - -. *f' C M*)› **and**
P = ‹(λ*C* - - - - - - - - *M* - - - - - - - - -. *P C M*)›
by *unfold-locales*
(*rule not-deleted-code-refine'*)

lemmas *refine* = *XX.refine*
end

locale *read-lbd-param-adder0* =
fixes *f* **and** *f'* **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** *P*
assumes *not-deleted-code-refine*: ‹(*f*, *f'*) ∈ [*P*]$_a$ *lbd-assn*^{k} → *x-assn*›
begin
sublocale *XX*: *read-lbd-param-adder* **where**
f = ‹λ-. *f*› **and**
f' = ‹λ-. *f'*› **and**
P = ‹λ-. *P*›
apply *unfold-locales*
using *not-deleted-code-refine*[*THEN remove-component-right*] **.**

lemmas *refine* = *XX.refine*[*THEN remove-unused-unit-parameter*]
end

abbreviation *read-outl-wl-heur-code* :: ‹- ⇒ *twl-st-wll-trail-fast2* ⇒ -› **where**
‹*read-outl-wl-heur-code f* ≡ *read-all-st-code* (λ- - - - - - - - - *M* - - - - - - -. *f M*)›
abbreviation *read-outl-wl-heur* :: ‹- ⇒ *isasat* ⇒ -› **where**
‹*read-outl-wl-heur f* ≡ *read-all-st* (λ- - - - - - - - - *M* - - - - - - -. *f M*)›

locale *read-outl-param-adder* =
fixes *f* **and** *f'* **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** *P* **and** *R*
assumes *not-deleted-code-refine*: ‹(*uncurry* (λ*S C*. *f C S*), *uncurry* (λ*S C'*. *f' C' S*)) ∈ [*uncurry* (λ*S C*. *P C S*)]$_a$ *out-learned-assn*^{k} *$*_a$ (*pure R*)^k → *x-assn*›
begin

lemma *not-deleted-code-refine'*:
shows ‹
(*uncurry17* (λ- - - - - - - - - *M* - - - - - - - *C*. *f C M*), *uncurry17* (λ- - - - - - - - - *M* - - - - - - - *C'*. *f' C' M*)) ∈
[*uncurry17* (λ- - - - - - - - - *M* - - - - - - - *C'*. *P C' M*)]$_a$
trail-pol-fast-assn^k *$*_a$ *arena-fast-assn*^{k} *$*_a$ *conflict-option-rel-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} *$*_a$
watchlist-fast-assn^k *$*_a$ *heuristic-bump-assn*^{k} *$*_a$ *uint32-nat-assn*^{k} *$*_a$ *cach-refinement-l-assn*^{k} *$*_a$
lbd-assn^k *$*_a$ *out-learned-assn*^{k} *$*_a$ *isasat-stats-assn*^{k} *$*_a$ *heuristic-assn*^{k} *$*_a$
aivdom-assn^k *$*_a$ *lcount-assn*^{k} *$*_a$ *opts-assn*^{k} *$*_a$ *arena-fast-assn*^{k} *$*_a$ *occs-assn*^{k} *$*_a$ (*pure R*)^k → *x-assn*›
apply (*rule add-pure-parameter2*)
apply (*drule remove-pure-parameter2'*[**where** *f* =*f* **and** *f'*=*f'*, *OF not-deleted-code-refine*])
apply (*drule remove-component-left*[**where** *X* = *trail-pol-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *conflict-option-rel-assn*])
apply (*drule remove-component-middle*[**where** *X* = *sint64-nat-assn*])
apply (*drule remove-component-middle*[**where** *X* = *watchlist-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *heuristic-bump-assn*])
apply (*drule remove-component-middle*[**where** *X* = *uint32-nat-assn*])
apply (*drule remove-component-middle*[**where** *X* = *cach-refinement-l-assn*])
apply (*drule remove-component-middle*[**where** *X* = *lbd-assn*])
apply (*drule remove-component-right*[**where** *X* = *isasat-stats-assn*])
apply (*drule remove-component-right*[**where** *X* = *heuristic-assn*])
apply (*drule remove-component-right*[**where** *X* = *aivdom-assn*])
apply (*drule remove-component-right*[**where** *X* = *lcount-assn*])
apply (*drule remove-component-right*[**where** *X* = *opts-assn*])
apply (*drule remove-component-right*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-right*[**where** *X* = *occs-assn*])
by (*rule hfref-cong*, *assumption*) *auto*

sublocale *XX* : *read-all-param-adder* **where**
f = ‹(λ*C* - - - - - - - - - *M* - - - - - - -. *f C M*)› **and**
f' = ‹(λ*C* - - - - - - - - - *M* - - - - - - -. *f' C M*)› **and**
P = ‹(λ*C* - - - - - - - - - *M* - - - - - - -. *P C M*)›
by *unfold-locales*
(*rule not-deleted-code-refine'*)

lemmas *refine* = *XX.refine*

end

locale *read-outl-param-adder0* =

fixes f **and** f' **and** *x-assn* :: ‹$'r \Rightarrow 'q \Rightarrow$ *assn*› **and** P
assumes *not-deleted-code-refine*: ‹$(f, f') \in [P]_a$ *out-learned-assn*$^k \to$ *x-assn*›
begin

sublocale *XX*: *read-outl-param-adder* **where**
$f = ‹\lambda\text{-}.\ f›$ **and**
$f' = ‹\lambda\text{-}.\ f'›$ **and**
$P = ‹\lambda\text{-}.\ P›$
apply *unfold-locales*
using *not-deleted-code-refine*[*THEN remove-component-right*] **.**

lemmas *refine* = *XX.refine*[*THEN remove-unused-unit-parameter*]
end

abbreviation *read-stats-wl-heur-code* :: ‹- $\Rightarrow$ *twl-st-wll-trail-fast2* $\Rightarrow$ -› **where**
‹*read-stats-wl-heur-code f* $\equiv$ *read-all-st-code* $(\lambda$- - - - - - - - - - *M* - - - - - -. *f M*)›
abbreviation *read-stats-wl-heur* :: ‹- $\Rightarrow$ *isasat* $\Rightarrow$ -› **where**
‹*read-stats-wl-heur f* $\equiv$ *read-all-st* $(\lambda$- - - - - - - - - - *M* - - - - - -. *f M*)›

locale *read-stats-param-adder* =
fixes f **and** f' **and** *x-assn* :: ‹$'r \Rightarrow 'q \Rightarrow$ *assn*› **and** P
assumes *not-deleted-code-refine*: ‹(*uncurry* $(\lambda S\ C.\ f\ C\ S)$, *uncurry* $(\lambda S\ C'.\ f'\ C'\ S)) \in [$*uncurry* $(\lambda S\ C.\ P\ C\ S)]_a$ *isasat-stats-assn*^{k} $*_a$ (*pure R*)$^k \to$ *x-assn*›
begin

lemma *not-deleted-code-refine'*:
shows ‹
(*uncurry17* $(\lambda$- - - - - - - - - - *M* - - - - - - *C*. *f C M*), *uncurry17* $(\lambda$- - - - - - - - - - *M* - - - - - - *C'*. *f' C' M*)) $\in$
[*uncurry17* $(\lambda$- - - - - - - - - - *M* - - - - - - *C'*. *P C' M*)]$_a$
trail-pol-fast-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ *conflict-option-rel-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$
watchlist-fast-assn^k $*_a$ *heuristic-bump-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $*_a$ *cach-refinement-l-assn*^{k} $*_a$
lbd-assn^k $*_a$ *out-learned-assn*^{k} $*_a$ *isasat-stats-assn*^{k} $*_a$ *heuristic-assn*^{k} $*_a$
aivdom-assn^k $*_a$ *lcount-assn*^{k} $*_a$ *opts-assn*^{k} $*_a$ *arena-fast-assn*^{k} $*_a$ *occs-assn*^{k} $*_a$ (*pure R*)$^k \to$ *x-assn*›
apply (*rule add-pure-parameter2*)
apply (*drule remove-pure-parameter2'*[**where** $f = f$ **and** $f' = f'$, *OF not-deleted-code-refine*])
apply (*drule remove-component-left*[**where** X = *trail-pol-fast-assn*])
apply (*drule remove-component-middle*[**where** X = *arena-fast-assn*])
apply (*drule remove-component-middle*[**where** X = *conflict-option-rel-assn*])
apply (*drule remove-component-middle*[**where** X = *sint64-nat-assn*])
apply (*drule remove-component-middle*[**where** X = *watchlist-fast-assn*])
apply (*drule remove-component-middle*[**where** X = *heuristic-bump-assn*])
apply (*drule remove-component-middle*[**where** X = *uint32-nat-assn*])
apply (*drule remove-component-middle*[**where** X = *cach-refinement-l-assn*])
apply (*drule remove-component-middle*[**where** X = *lbd-assn*])
apply (*drule remove-component-middle*[**where** X = *out-learned-assn*])
apply (*drule remove-component-right*[**where** X = *heuristic-assn*])
apply (*drule remove-component-right*[**where** X = *aivdom-assn*])
apply (*drule remove-component-right*[**where** X = *lcount-assn*])
apply (*drule remove-component-right*[**where** X = *opts-assn*])
apply (*drule remove-component-right*[**where** X = *arena-fast-assn*])
apply (*drule remove-component-right*[**where** X = *occs-assn*])
by (*rule hfref-cong*, *assumption*) *auto*

sublocale *XX* : *read-all-param-adder* **where**

$f = \ \langle(\lambda C$ - - - - - - - - - - M - - - - - -. $f\ C\ M)\rangle$ **and**
$f' = \langle(\lambda C$ - - - - - - - - - - M - - - - - -. $f'\ C\ M)\rangle$ **and**
$P = \ \langle(\lambda C$ - - - - - - - - - - M - - - - - -. $P\ C\ M)\rangle$
by *unfold-locales*
(*rule not-deleted-code-refine'*)

lemmas *refine* = *XX.refine*
end

locale *read-stats-param-adder0* =
fixes f **and** f' **and** *x-assn* :: $\langle 'r \Rightarrow 'q \Rightarrow assn\rangle$ **and** P
assumes *not-deleted-code-refine*: $\langle(f, f') \in [P]_a\ isasat\text{-}stats\text{-}assn^k \to x\text{-}assn\rangle$
begin
sublocale *XX*: *read-stats-param-adder* **where**
$f = \langle\lambda$-. $f\rangle$ **and**
$f' = \langle\lambda$-. $f'\rangle$ **and**
$P = \langle\lambda$-. $P\rangle$
apply *unfold-locales*
using *not-deleted-code-refine*[*THEN remove-component-right*] **.**

lemmas *refine* = *XX.refine*[*THEN remove-unused-unit-parameter*]
end

abbreviation *read-heur-wl-heur-code* :: $\langle$- $\Rightarrow$ *twl-st-wll-trail-fast2* $\Rightarrow$ -$\rangle$ **where**
$\langle$*read-heur-wl-heur-code f* $\equiv$ *read-all-st-code* $(\lambda$- - - - - - - - - - - M - - - - -. $f\ M)\rangle$
abbreviation *read-heur-wl-heur* :: $\langle$- $\Rightarrow$ *isasat* $\Rightarrow$ -$\rangle$ **where**
$\langle$*read-heur-wl-heur f* $\equiv$ *read-all-st* $(\lambda$- - - - - - - - - - - M - - - - -. $f\ M)\rangle$

locale *read-heur-param-adder* =
fixes f **and** f' **and** *x-assn* :: $\langle 'r \Rightarrow 'q \Rightarrow assn\rangle$ **and** P **and** R
assumes *not-deleted-code-refine*: $\langle(uncurry\ (\lambda S\ C.\ f\ C\ S), uncurry\ (\lambda S\ C'.\ f'\ C'\ S)) \in [uncurry\ (\lambda S\ C.\ P\ C\ S)]_a\ heuristic\text{-}assn^k *_a (pure\ R)^k \to x\text{-}assn\rangle$
begin

lemma *not-deleted-code-refine'*:
shows $\langle$
$(uncurry17\ (\lambda$- - - - - - - - - - - M - - - - - $C.\ f\ C\ M)$, $uncurry17\ (\lambda$- - - - - - - - - - - M - - - - - $C'.\ f'\ C'\ M)) \in$
$[uncurry17\ (\lambda$- - - - - - - - - - - M - - - - - $C'.\ P\ C'\ M)]_a$
trail-pol-fast-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ *conflict-option-rel-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$
watchlist-fast-assn^k $*_a$ *heuristic-bump-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $*_a$ *cach-refinement-l-assn*^{k} $*_a$
lbd-assn^k $*_a$ *out-learned-assn*^{k} $*_a$ *isasat-stats-assn*^{k} $*_a$ *heuristic-assn*^{k} $*_a$
aivdom-assn^k $*_a$ *lcount-assn*^{k} $*_a$ *opts-assn*^{k} $*_a$ *arena-fast-assn*^{k} $*_a$ *occs-assn*^{k} $*_a$ $(pure\ R)^k \to x\text{-}assn\rangle$
apply (*rule add-pure-parameter2*)
apply (*drule remove-pure-parameter2'*[**where** $f = f$ **and** $f' = f'$, *OF not-deleted-code-refine*])
apply (*drule remove-component-left*[**where** X = *trail-pol-fast-assn*])
apply (*drule remove-component-middle*[**where** X = *arena-fast-assn*])
apply (*drule remove-component-middle*[**where** X = *conflict-option-rel-assn*])
apply (*drule remove-component-middle*[**where** X = *sint64-nat-assn*])
apply (*drule remove-component-middle*[**where** X = *watchlist-fast-assn*])
apply (*drule remove-component-middle*[**where** X = *heuristic-bump-assn*])
apply (*drule remove-component-middle*[**where** X = *uint32-nat-assn*])
apply (*drule remove-component-middle*[**where** X = *cach-refinement-l-assn*])
apply (*drule remove-component-middle*[**where** X = *lbd-assn*])
apply (*drule remove-component-middle*[**where** X = *out-learned-assn*])

apply (*drule remove-component-middle*[**where** X = *isasat-stats-assn*])
apply (*drule remove-component-right*[**where** X = *aivdom-assn*])
apply (*drule remove-component-right*[**where** X = *lcount-assn*])
apply (*drule remove-component-right*[**where** X = *opts-assn*])
apply (*drule remove-component-right*[**where** X = *arena-fast-assn*])
apply (*drule remove-component-right*[**where** X = *occs-assn*])
by (*rule hfref-cong*, *assumption*) *auto*

sublocale *XX* : *read-all-param-adder* **where**
f = ‹(λ*C* - - - - - - - - - - - *M* - - - - -. *f C M*)› **and**
f' = ‹(λ*C* - - - - - - - - - - - *M* - - - - -. *f' C M*)› **and**
P = ‹(λ*C* - - - - - - - - - - - *M* - - - - -. *P C M*)›
by *unfold-locales*
(*rule not-deleted-code-refine'*)

lemmas *refine* = *XX.refine*
end

locale *read-heur-param-adder0* =
fixes f **and** f' **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** P
assumes *not-deleted-code-refine*: ‹(*f*, *f'*) ∈ [*P*]$_a$ *heuristic-assn*^{k} → *x-assn*›
begin
sublocale *XX*: *read-heur-param-adder* **where**
f = ‹λ-. *f*› **and**
f' = ‹λ-. *f'*› **and**
P = ‹λ-. *P*›
apply *unfold-locales*
using *not-deleted-code-refine*[*THEN remove-component-right*] .

lemmas *refine* = *XX.refine*[*THEN remove-unused-unit-parameter*]
end

locale *read-heur-param-adder2* =
fixes f **and** f' **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** P **and** R **and** R'
assumes *not-deleted-code-refine*: ‹(*uncurry2* (λ*S C D*. *f C D S*), *uncurry2* (λ*S C' D'*. *f' C' D' S*)) ∈ [*uncurry2* (λ*S C D*. *P C D S*)]$_a$ *heuristic-assn*^{k} *$*_a$ (*pure R*)^k*$*_a$ (*pure R'*)^k → *x-assn*›
begin
sublocale *XX*: *read-heur-param-adder* **where**
f = ‹λ(*C*,*D*) *N*. *f C D N*› **and**
f' = ‹λ(*C*,*D*) *N*. *f' C D N*› **and**
P = ‹λ(*C*,*D*) *N*. *P C D N*› **and**
R = ‹$R \times_f R'$›
apply *unfold-locales*
using *not-deleted-code-refine*[*THEN merge-second-pure-argument*] .

lemma *refine*:
‹(*uncurry2* (λ*N C D*. *read-heur-wl-heur-code* (*f C D*) *N*),
uncurry2 (λ*N C'* *D*. *read-heur-wl-heur* (*f' C'* *D*) *N*))
∈ [*uncurry2* (λ*S C D*. *P C D* (*get-heur S*))]$_a$ *isasat-bounded-assn*^{k} *$*_a$ (*pure R*)^k *$*_a$ (*pure R'*)^k → *x-assn*›
by (*rule XX.refine*[*THEN split-snd-pure-arg*, *unfolded prod.case*])

end

abbreviation *read-vdom-wl-heur-code* :: ‹- ⇒ *twl-st-wll-trail-fast2* ⇒ -› **where**

‹*read-vdom-wl-heur-code f* ≡ *read-all-st-code* (λ- - - - - - - - - - - - - *M* - - - -. *f M*)›
abbreviation *read-vdom-wl-heur* :: ‹- ⇒ *isasat* ⇒ -› **where**
‹*read-vdom-wl-heur f* ≡ *read-all-st* (λ- - - - - - - - - - - - - *M* - - - -. *f M*)›

locale *read-vdom-param-adder* =
fixes *f* **and** *f'* **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** *P* **and** *R*
assumes *not-deleted-code-refine*: ‹(*uncurry* (λ*S C*. *f C S*), *uncurry* (λ*S C'*. *f' C' S*)) ∈ [*uncurry* (λ*S C*. *P C S*)]$_a$ *aivdom-assn*^{k} *$*_a$ (*pure R*)^k → *x-assn*›
begin

lemma *not-deleted-code-refine'*:
shows ‹
(*uncurry17* (λ- - - - - - - - - - - - - *M* - - - - *C*. *f C M*), *uncurry17* (λ- - - - - - - - - - - - - *M* - - - - *C'*. *f' C' M*)) ∈
[*uncurry17* (λ- - - - - - - - - - - - - *M* - - - - *C'*. *P C' M*)]$_a$
trail-pol-fast-assn^k *$*_a$ *arena-fast-assn*^{k} *$*_a$ *conflict-option-rel-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} *$*_a$
watchlist-fast-assn^k *$*_a$ *heuristic-bump-assn*^{k} *$*_a$ *uint32-nat-assn*^{k} *$*_a$ *cach-refinement-l-assn*^{k} *$*_a$
lbd-assn^k *$*_a$ *out-learned-assn*^{k} *$*_a$ *isasat-stats-assn*^{k} *$*_a$ *heuristic-assn*^{k} *$*_a$
aivdom-assn^k *$*_a$ *lcount-assn*^{k} *$*_a$ *opts-assn*^{k} *$*_a$ *arena-fast-assn*^{k} *$*_a$ *occs-assn*^{k} *$*_a$ (*pure R*)^k → *x-assn*›
apply (*rule add-pure-parameter2*)
apply (*drule remove-pure-parameter2'*[**where** *f* =*f* **and** *f'*=*f'*, *OF not-deleted-code-refine*])
apply (*drule remove-component-left*[**where** *X* = *trail-pol-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *conflict-option-rel-assn*])
apply (*drule remove-component-middle*[**where** *X* = *sint64-nat-assn*])
apply (*drule remove-component-middle*[**where** *X* = *watchlist-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *heuristic-bump-assn*])
apply (*drule remove-component-middle*[**where** *X* = *uint32-nat-assn*])
apply (*drule remove-component-middle*[**where** *X* = *cach-refinement-l-assn*])
apply (*drule remove-component-middle*[**where** *X* = *lbd-assn*])
apply (*drule remove-component-middle*[**where** *X* = *out-learned-assn*])
apply (*drule remove-component-middle*[**where** *X* = *isasat-stats-assn*])
apply (*drule remove-component-middle*[**where** *X* = *heuristic-assn*])
apply (*drule remove-component-right*[**where** *X* = *lcount-assn*])
apply (*drule remove-component-right*[**where** *X* = *opts-assn*])
apply (*drule remove-component-right*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-right*[**where** *X* = *occs-assn*])
by (*rule hfref-cong*, *assumption*) *auto*

sublocale *XX* : *read-all-param-adder* **where**
f = ‹(λ*C* - - - - - - - - - - - - *M* - - - -. *f C M*)› **and**
f' = ‹(λ*C* - - - - - - - - - - - - *M* - - - -. *f' C M*)› **and**
P = ‹(λ*C* - - - - - - - - - - - - *M* - - - -. *P C M*)›
by *unfold-locales*
(*rule not-deleted-code-refine'*)

lemmas *refine* = *XX.refine*
end

locale *read-vdom-param-adder0* =
fixes *f* **and** *f'* **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** *P*
assumes *not-deleted-code-refine*: ‹(*f*, *f'*) ∈ [*P*]$_a$ *aivdom-assn*^{k} → *x-assn*›
begin

sublocale *XX*: *read-vdom-param-adder* **where**
f = ‹λ-. *f*› **and**

$f' = ‹\lambda\text{-}.\ f'›$ **and**
$P = ‹\lambda\text{-}.\ P›$
apply *unfold-locales*
using *not-deleted-code-refine*[*THEN remove-component-right*] **.**

lemmas *refine* = *XX.refine*[*THEN remove-unused-unit-parameter*]
end

abbreviation *read-lcount-wl-heur-code* :: ‹- ⇒ *twl-st-wll-trail-fast2* ⇒ -› **where**
‹*read-lcount-wl-heur-code f* ≡ *read-all-st-code* (λ- - - - - - - - - - - - - *M* - - -. *f M*)›
abbreviation *read-lcount-wl-heur* :: ‹- ⇒ *isasat* ⇒ -› **where**
‹*read-lcount-wl-heur f* ≡ *read-all-st* (λ- - - - - - - - - - - - - *M* - - -. *f M*)›

locale *read-lcount-param-adder* =
fixes *f* **and** *f'* **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** *P* **and** *R*
assumes *not-deleted-code-refine*: ‹(*uncurry* (λ*S C*. *f C S*), *uncurry* (λ*S C'*. *f' C' S*)) ∈ [*uncurry* (λ*S C*. *P C S*)]$_a$ *lcount-assn*^{k} $*_a$ (*pure R*)^k → *x-assn*›
begin

lemma *not-deleted-code-refine'*:
shows ‹
(*uncurry17* (λ- - - - - - - - - - - - - *M* - - - *C*. *f C M*), *uncurry17* (λ- - - - - - - - - - - - - *M* - - - *C'*. *f' C' M*)) ∈
[*uncurry17* (λ- - - - - - - - - - - - - *M* - - - *C'*. *P C' M*)]$_a$
trail-pol-fast-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ *conflict-option-rel-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$
watchlist-fast-assn^k $*_a$ *heuristic-bump-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $*_a$ *cach-refinement-l-assn*^{k} $*_a$
lbd-assn^k $*_a$ *out-learned-assn*^{k} $*_a$ *isasat-stats-assn*^{k} $*_a$ *heuristic-assn*^{k} $*_a$
aivdom-assn^k $*_a$ *lcount-assn*^{k} $*_a$ *opts-assn*^{k} $*_a$ *arena-fast-assn*^{k} $*_a$ *occs-assn*^{k} $*_a$ (*pure R*)^k → *x-assn*›
apply (*rule add-pure-parameter2*)
apply (*drule remove-pure-parameter2'*[**where** *f* =*f* **and** *f'*=*f'*, *OF not-deleted-code-refine*])
apply (*drule remove-component-left*[**where** *X* = *trail-pol-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *conflict-option-rel-assn*])
apply (*drule remove-component-middle*[**where** *X* = *sint64-nat-assn*])
apply (*drule remove-component-middle*[**where** *X* = *watchlist-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *heuristic-bump-assn*])
apply (*drule remove-component-middle*[**where** *X* = *uint32-nat-assn*])
apply (*drule remove-component-middle*[**where** *X* = *cach-refinement-l-assn*])
apply (*drule remove-component-middle*[**where** *X* = *lbd-assn*])
apply (*drule remove-component-middle*[**where** *X* = *out-learned-assn*])
apply (*drule remove-component-middle*[**where** *X* = *isasat-stats-assn*])
apply (*drule remove-component-middle*[**where** *X* = *heuristic-assn*])
apply (*drule remove-component-middle*[**where** *X* = *aivdom-assn*])
apply (*drule remove-component-right*[**where** *X* = *opts-assn*])
apply (*drule remove-component-right*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-right*[**where** *X* = *occs-assn*])
by (*rule hfref-cong*, *assumption*) *auto*

sublocale *XX* : *read-all-param-adder* **where**
f = ‹(λ*C* - - - - - - - - - - - - - *M* - - -. *f C M*)› **and**
f' = ‹(λ*C* - - - - - - - - - - - - - *M* - - -. *f' C M*)› **and**
P = ‹(λ*C* - - - - - - - - - - - - - *M* - - -. *P C M*)›
by *unfold-locales*

(*rule not-deleted-code-refine'*)

lemmas *refine* = *XX.refine*
end

locale *read-lcount-param-adder0* =
fixes *f* **and** *f'* **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** *P*
assumes *not-deleted-code-refine*: ‹(*f*, *f'*) ∈ [*P*]$_a$ *lcount-assn*^{k} → *x-assn*›
begin
sublocale *XX*: *read-lcount-param-adder* **where**
f = ‹λ-. *f*› **and**
f' = ‹λ-. *f'*› **and**
P = ‹λ-. *P*›
apply *unfold-locales*
using *not-deleted-code-refine*[*THEN remove-component-right*] **.**

lemmas *refine* = *XX.refine*[*THEN remove-unused-unit-parameter*]
end

abbreviation *read-opts-wl-heur-code* :: ‹- ⇒ *twl-st-wll-trail-fast2* ⇒ -› **where**
‹*read-opts-wl-heur-code f* ≡ *read-all-st-code* (λ- - - - - - - - - - - - - - - *M* - -. *f M*)›
abbreviation *read-opts-wl-heur* :: ‹- ⇒ *isasat* ⇒ -› **where**
‹*read-opts-wl-heur f* ≡ *read-all-st* (λ- - - - - - - - - - - - - - - *M* - -. *f M*)›

locale *read-opts-param-adder* =
fixes *f* **and** *f'* **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** *P*
assumes *not-deleted-code-refine*: ‹(*uncurry* (λ*S C*. *f C S*), *uncurry* (λ*S C'*. *f' C' S*)) ∈ [*uncurry* (λ*S C*. *P C S*)]$_a$ *opts-assn*^{k} *$*_a$ (*pure R*)^k → *x-assn*›
begin

lemma *not-deleted-code-refine'*:
shows ‹
(*uncurry17* (λ- - - - - - - - - - - - - - - *M* - - *C*. *f C M*), *uncurry17* (λ- - - - - - - - - - - - - - - *M* - - *C'*. *f' C' M*)) ∈
[*uncurry17* (λ- - - - - - - - - - - - - - - *M* - - *C'*. *P C' M*)]$_a$
trail-pol-fast-assn^k *$*_a$ *arena-fast-assn*^{k} *$*_a$ *conflict-option-rel-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} *$*_a$
watchlist-fast-assn^k *$*_a$ *heuristic-bump-assn*^{k} *$*_a$ *uint32-nat-assn*^{k} *$*_a$ *cach-refinement-l-assn*^{k} *$*_a$
lbd-assn^k *$*_a$ *out-learned-assn*^{k} *$*_a$ *isasat-stats-assn*^{k} *$*_a$ *heuristic-assn*^{k} *$*_a$
aivdom-assn^k *$*_a$ *lcount-assn*^{k} *$*_a$ *opts-assn*^{k} *$*_a$ *arena-fast-assn*^{k} *$*_a$ *occs-assn*^{k} *$*_a$ (*pure R*)^k → *x-assn*›
apply (*rule add-pure-parameter2*)
apply (*drule remove-pure-parameter2'*[**where** *f* =*f* **and** *f'*=*f'*, *OF not-deleted-code-refine*])
apply (*drule remove-component-left*[**where** *X* = *trail-pol-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *conflict-option-rel-assn*])
apply (*drule remove-component-middle*[**where** *X* = *sint64-nat-assn*])
apply (*drule remove-component-middle*[**where** *X* = *watchlist-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *heuristic-bump-assn*])
apply (*drule remove-component-middle*[**where** *X* = *uint32-nat-assn*])
apply (*drule remove-component-middle*[**where** *X* = *cach-refinement-l-assn*])
apply (*drule remove-component-middle*[**where** *X* = *lbd-assn*])
apply (*drule remove-component-middle*[**where** *X* = *out-learned-assn*])
apply (*drule remove-component-middle*[**where** *X* = *isasat-stats-assn*])
apply (*drule remove-component-middle*[**where** *X* = *heuristic-assn*])
apply (*drule remove-component-middle*[**where** *X* = *aivdom-assn*])

apply (*drule remove-component-middle*[**where** X = *lcount-assn*])
apply (*drule remove-component-right*[**where** X = *arena-fast-assn*])
apply (*drule remove-component-right*[**where** X = *occs-assn*])
by (*rule hfref-cong*, *assumption*) *auto*

sublocale *XX* : *read-all-param-adder* **where**
f = ‹(λ*C* - - - - - - - - - - - - - - - *M* - -. *f C M*)› **and**
f' = ‹(λ*C* - - - - - - - - - - - - - - - *M* - -. *f' C M*)› **and**
P = ‹(λ*C* - - - - - - - - - - - - - - - *M* - -. *P C M*)›
by *unfold-locales*
(*rule not-deleted-code-refine'*)

lemmas *refine* = *XX.refine*
end

locale *read-opts-param-adder0* =
fixes f **and** f' **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** P
assumes *not-deleted-code-refine*: ‹(*f*, *f'*) ∈ [*P*]$_a$ *opts-assn*^{k} → *x-assn*›
begin

sublocale *XX*: *read-opts-param-adder* **where**
f = ‹λ-. *f*› **and**
f' = ‹λ-. *f'*› **and**
P = ‹λ-. *P*›
apply *unfold-locales*
using *not-deleted-code-refine*[*THEN remove-component-right*] **.**

lemmas *refine* = *XX.refine*[*THEN remove-unused-unit-parameter*]
end

abbreviation *read-old-arena-wl-heur-code* :: ‹- ⇒ *twl-st-wll-trail-fast2* ⇒ -› **where**
‹*read-old-arena-wl-heur-code f* ≡ *read-all-st-code* (λ- - - - - - - - - - - - - - - *M*. *f M*)›
abbreviation *read-old-arena-wl-heur* :: ‹- ⇒ *isasat* ⇒ -› **where**
‹*read-old-arena-wl-heur f* ≡ *read-all-st* (λ- - - - - - - - - - - - - - - *M*. *f M*)›

locale *read-old-arena-param-adder* =
fixes f **and** f' **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** P
assumes *not-deleted-code-refine*: ‹(*uncurry* (λ*S C*. *f C S*), *uncurry* (λ*S C'*. *f' C' S*)) ∈ [*uncurry* (λ*S C*. *P C S*)]$_a$ *arena-fast-assn*^{k} ∗$_a$ (*pure R*)^k → *x-assn*›
begin

lemma *not-deleted-code-refine'*:
shows ‹
(*uncurry17* (λ- - - - - - - - - - - - - - - *M* - *C*. *f C M*), *uncurry17* (λ- - - - - - - - - - - - - - - *M* - *C'*. *f' C' M*)) ∈
[*uncurry17* (λ- - - - - - - - - - - - - - - *M* - *C'*. *P C' M*)]$_a$
trail-pol-fast-assn^k ∗$_a$ *arena-fast-assn*^{k} ∗$_a$ *conflict-option-rel-assn*^{k} ∗$_a$ *sint64-nat-assn*^{k} ∗$_a$
watchlist-fast-assn^k ∗$_a$ *heuristic-bump-assn*^{k} ∗$_a$ *uint32-nat-assn*^{k} ∗$_a$ *cach-refinement-l-assn*^{k} ∗$_a$
lbd-assn^k ∗$_a$ *out-learned-assn*^{k} ∗$_a$ *isasat-stats-assn*^{k} ∗$_a$ *heuristic-assn*^{k} ∗$_a$
aivdom-assn^k ∗$_a$ *lcount-assn*^{k} ∗$_a$ *opts-assn*^{k} ∗$_a$ *arena-fast-assn*^{k} ∗$_a$ *occs-assn*^{k} ∗$_a$ (*pure R*)^k → *x-assn*›
apply (*rule add-pure-parameter2*)
apply (*drule remove-pure-parameter2'*[**where** f =*f* **and** f'=*f'*, *OF not-deleted-code-refine*])
apply (*drule remove-component-left*[**where** X = *trail-pol-fast-assn*])
apply (*drule remove-component-middle*[**where** X = *arena-fast-assn*])

apply (*drule remove-component-middle*[**where** X = *conflict-option-rel-assn*])
apply (*drule remove-component-middle*[**where** X = *sint64-nat-assn*])
apply (*drule remove-component-middle*[**where** X = *watchlist-fast-assn*])
apply (*drule remove-component-middle*[**where** X = *heuristic-bump-assn*])
apply (*drule remove-component-middle*[**where** X = *uint32-nat-assn*])
apply (*drule remove-component-middle*[**where** X = *cach-refinement-l-assn*])
apply (*drule remove-component-middle*[**where** X = *lbd-assn*])
apply (*drule remove-component-middle*[**where** X = *out-learned-assn*])
apply (*drule remove-component-middle*[**where** X = *isasat-stats-assn*])
apply (*drule remove-component-middle*[**where** X = *heuristic-assn*])
apply (*drule remove-component-middle*[**where** X = *aivdom-assn*])
apply (*drule remove-component-middle*[**where** X = *lcount-assn*])
apply (*drule remove-component-middle*[**where** X = *opts-assn*])
apply (*drule remove-component-right*[**where** X = *occs-assn*])
by (*rule hfref-cong*, *assumption*) *auto*

sublocale *XX* : *read-all-param-adder* **where**
f = ‹(λC - - - - - - - - - - - - - - - - M -. f C M)› **and**
f' = ‹(λC - - - - - - - - - - - - - - - - M -. f' C M)› **and**
P = ‹(λC - - - - - - - - - - - - - - - - M -. P C M)›
by *unfold-locales*
(*rule not-deleted-code-refine'*)

lemmas *refine* = *XX.refine*
end

locale *read-old-arena-param-adder0* =
fixes f **and** f' **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** P
assumes *not-deleted-code-refine*: ‹(f, f') ∈ $[P]_a$ *arena-fast-assn*^{k} → *x-assn*›
begin

sublocale *XX*: *read-old-arena-param-adder* **where**
f = ‹λ-. f› **and**
f' = ‹λ-. f'› **and**
P = ‹λ-. P›
apply *unfold-locales*
using *not-deleted-code-refine*[*THEN remove-component-right*] **.**

lemmas *refine* = *XX.refine*[*THEN remove-unused-unit-parameter*]
end

abbreviation *read-occs-wl-heur-code* :: ‹- ⇒ *twl-st-wll-trail-fast2* ⇒ -› **where**
‹*read-occs-wl-heur-code f* ≡ *read-all-st-code* (λ- - - - - - - - - - - - - - - - M. f M)›
abbreviation *read-occs-wl-heur* :: ‹- ⇒ *isasat* ⇒ -› **where**
‹*read-occs-wl-heur f* ≡ *read-all-st* (λ- - - - - - - - - - - - - - - - M. f M)›

locale *read-occs-param-adder* =
fixes f **and** f' **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** P **and** R
assumes *not-deleted-code-refine*: ‹(*uncurry* (λS C. f C S), *uncurry* (λS C'. f' C' S)) ∈ [*uncurry* (λS C. P C S)]$_a$ *occs-assn*^{k} *$*_a$ (*pure* R)^k → *x-assn*›
begin

lemma *not-deleted-code-refine'*:
shows ‹

(*uncurry17* (λ- - - - - - - - - - - - - - - - - *M C*. *f C M*), *uncurry17* (λ- - - - - - - - - - - - - - - - - *M C'*. *f' C' M*)) ∈
[*uncurry17* (λ- - - - - - - - - - - - - - - - - *M C'*. *P C' M*)]$_a$
trail-pol-fast-assn^k **$_a$ arena-fast-assn*^{k} **$_a$ conflict-option-rel-assn*^{k} **$_a$ sint64-nat-assn*^{k} **$_a$*
watchlist-fast-assn^k **$_a$ heuristic-bump-assn*^{k} **$_a$ uint32-nat-assn*^{k} **$_a$ cach-refinement-l-assn*^{k} **$_a$*
lbd-assn^k **$_a$ out-learned-assn*^{k} **$_a$ isasat-stats-assn*^{k} **$_a$ heuristic-assn*^{k} **$_a$*
aivdom-assn^k **$_a$ lcount-assn*^{k} **$_a$ opts-assn*^{k} **$_a$ arena-fast-assn*^{k} **$_a$ occs-assn*^{k} **$_a$ (pure R)*^{k} → *x-assn*›
apply (*rule add-pure-parameter2*)
apply (*drule remove-pure-parameter2'*[**where** *f* =*f* **and** *f'*=*f'*, *OF not-deleted-code-refine*])
apply (*drule remove-component-left*[**where** *X* = *trail-pol-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *conflict-option-rel-assn*])
apply (*drule remove-component-middle*[**where** *X* = *sint64-nat-assn*])
apply (*drule remove-component-middle*[**where** *X* = *watchlist-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *heuristic-bump-assn*])
apply (*drule remove-component-middle*[**where** *X* = *uint32-nat-assn*])
apply (*drule remove-component-middle*[**where** *X* = *cach-refinement-l-assn*])
apply (*drule remove-component-middle*[**where** *X* = *lbd-assn*])
apply (*drule remove-component-middle*[**where** *X* = *out-learned-assn*])
apply (*drule remove-component-middle*[**where** *X* = *isasat-stats-assn*])
apply (*drule remove-component-middle*[**where** *X* = *heuristic-assn*])
apply (*drule remove-component-middle*[**where** *X* = *aivdom-assn*])
apply (*drule remove-component-middle*[**where** *X* = *lcount-assn*])
apply (*drule remove-component-middle*[**where** *X* = *opts-assn*])
apply (*drule remove-component-middle*[**where** *X* = *arena-fast-assn*])
by (*rule hfref-cong*, *assumption*) *auto*

sublocale *XX* : *read-all-param-adder* **where**
f = ‹(λ*C* - - - - - - - - - - - - - - - - - *M*. *f C M*)› **and**
f' = ‹(λ*C* - - - - - - - - - - - - - - - - - *M*. *f' C M*)› **and**
P = ‹(λ*C* - - - - - - - - - - - - - - - - - *M*. *P C M*)›
by *unfold-locales*
(*rule not-deleted-code-refine'*)

lemmas *refine* = *XX.refine*
end

locale *read-occs-param-adder0* =
fixes *f* **and** *f'* **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** *P*
assumes *not-deleted-code-refine*: ‹(*f*, *f'*) ∈ [*P*]$_a$ *occs-assn*^{k} → *x-assn*›
begin

sublocale *XX*: *read-occs-param-adder* **where**
f = ‹λ-. *f*› **and**
f' = ‹λ-. *f'*› **and**
P = ‹λ-. *P*›
apply *unfold-locales*
using *not-deleted-code-refine*[*THEN remove-component-right*] **.**

lemmas *refine* = *XX.refine*[*THEN remove-unused-unit-parameter*]
end

locale *read-occs-param-adder2* =
fixes *f* **and** *f'* **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** *P* **and** *R* **and** *R'*
assumes *not-deleted-code-refine*: ‹(*uncurry2* (λ*S C D*. *f C D S*), *uncurry2* (λ*S C' D'*. *f' C' D' S*)) ∈

$[uncurry2\ (\lambda S\ C\ D.\ P\ C\ D\ S)]_a\ occs\text{-}assn^k *_a (pure\ R)^k *_a (pure\ R')^k \to x\text{-}assn\rangle$
begin

sublocale *XX*: *read-occs-param-adder* **where**
$f = \langle\lambda(C,D)\ N.\ f\ C\ D\ N\rangle$ **and**
$f' = \langle\lambda(C,D)\ N.\ f'\ C\ D\ N\rangle$ **and**
$P = \langle\lambda(C,D)\ N.\ P\ C\ D\ N\rangle$ **and**
$R = \langle R \times_f R'\rangle$
apply *unfold-locales*
using *not-deleted-code-refine*[*THEN merge-second-pure-argument*] **.**

lemma *refine*:
$\langle(uncurry2\ (\lambda N\ C\ D.\ read\text{-}occs\text{-}wl\text{-}heur\text{-}code\ (f\ C\ D)\ N),$
$uncurry2\ (\lambda N\ C'\ D.\ read\text{-}occs\text{-}wl\text{-}heur\ (f'\ C'\ D)\ N))$
$\in [uncurry2\ (\lambda S\ C\ D.\ P\ C\ D\ (get\text{-}occs\ S))]_a\ isasat\text{-}bounded\text{-}assn^k\ *_a (pure\ R)^k *_a (pure\ R')^k \to x\text{-}assn\rangle$
by (*rule XX.refine*[*THEN split-snd-pure-arg*, *unfolded prod.case*])

lemma *mop-refine*:
$\langle(uncurry2\ (\lambda N\ C\ D.\ read\text{-}occs\text{-}wl\text{-}heur\text{-}code\ (f\ C\ D)\ N),\ uncurry2\ (\lambda N\ C\ D.\ XX.XX.mop\ N\ (C,D)))$
$\in$
$isasat\text{-}bounded\text{-}assn^k *_a (pure\ R)^k *_a (pure\ R')^k \to_a x\text{-}assn\rangle$
unfolding *mop-def XX.XX.mop-def*
apply (*rule refine-ASSERT-move-to-pre2*)
unfolding *prod.simps*
apply (*rule refine*[*unfolded comp-def*])
done

end

locale *read-trail-arena-param-adder-ops* =
fixes $P :: \langle'b \Rightarrow trail\text{-}pol \Rightarrow arena \Rightarrow bool\rangle$ **and** $f' :: \langle'b \Rightarrow trail\text{-}pol \Rightarrow arena\text{-}el\ list \Rightarrow 'r\ nres\rangle$
begin

definition *mop* **where**
$\langle mop\ N\ C = do\ \{$
$ASSERT\ (P\ C\ (get\text{-}trail\text{-}wl\text{-}heur\ N)\ (get\text{-}clauses\text{-}wl\text{-}heur\ N));$
$read\text{-}all\text{-}st\ (\lambda M\ N\ \text{- - - - - - - - - - - - - - - - -}.\ f'\ C\ M\ N)\ N$
$\}\rangle$

end

locale *read-trail-arena-param-adder* = *read-trail-arena-param-adder-ops P f'*
for $P :: \langle'b \Rightarrow trail\text{-}pol \Rightarrow arena \Rightarrow bool\rangle$ **and** $f' :: \langle'b \Rightarrow trail\text{-}pol \Rightarrow arena\text{-}el\ list \Rightarrow 'r\ nres\rangle$ +
fixes $R :: \langle('a \times 'b)\ set\rangle$ **and** f **and** $x\text{-}assn :: \langle'r \Rightarrow 'q \Rightarrow assn\rangle$
assumes *not-deleted-code-refine*: $\langle(uncurry2\ (\lambda S\ T\ C.\ f\ C\ S\ T),\ uncurry2\ (\lambda S\ T\ C'.\ f'\ C'\ S\ T)) \in$
$[uncurry2\ (\lambda S\ T\ C.\ P\ C\ S\ T)]_a\ trail\text{-}pol\text{-}fast\text{-}assn^k *_a arena\text{-}fast\text{-}assn^k *_a (pure\ R)^k \to x\text{-}assn\rangle$
begin

lemma *not-deleted-code-refine'*:
$\langle(uncurry17\ (\lambda M\ N\ \text{- - - - - - - - - - - - - - - -}\ C.\ f\ C\ M\ N),\ uncurry17\ (\lambda M\ N\ \text{- - - - - - - - - - - - - - - -}$
$C'.\ f'\ C'\ M\ N)) \in$
$[uncurry17\ (\lambda M\ N\ \text{- - - - - - - - - - - - - - - -}\ C'.\ P\ C'\ M\ N)]_a$
$trail\text{-}pol\text{-}fast\text{-}assn^k *_a arena\text{-}fast\text{-}assn^k *_a conflict\text{-}option\text{-}rel\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a$
$watchlist\text{-}fast\text{-}assn^k *_a heuristic\text{-}bump\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k *_a cach\text{-}refinement\text{-}l\text{-}assn^k *_a$

lbd-assn^k $*_a$ *out-learned-assn*^{k} $*_a$ *isasat-stats-assn*^{k} $*_a$ *heuristic-assn*^{k} $*_a$
aivdom-assn^k $*_a$ *lcount-assn*^{k} $*_a$ *opts-assn*^{k} $*_a$ *arena-fast-assn*^{k} $*_a$ *occs-assn*^{k} $*_a$ (*pure R*)^k $\to$ *x-assn*›
apply (*rule add-pure-parameter2*)
apply (*drule remove-pure-parameter2'*[*OF not-deleted-code-refine*])
apply (*drule remove-component-right*[**where** *X* = *conflict-option-rel-assn*])
apply (*drule remove-component-right*[**where** *X* = *sint64-nat-assn*])
apply (*drule remove-component-right*[**where** *X* = *watchlist-fast-assn*])
apply (*drule remove-component-right*[**where** *X* = *heuristic-bump-assn*])
apply (*drule remove-component-right*[**where** *X* = *uint32-nat-assn*])
apply (*drule remove-component-right*[**where** *X* = *cach-refinement-l-assn*])
apply (*drule remove-component-right*[**where** *X* = *lbd-assn*])
apply (*drule remove-component-right*[**where** *X* = *out-learned-assn*])
apply (*drule remove-component-right*[**where** *X* = *isasat-stats-assn*])
apply (*drule remove-component-right*[**where** *X* = *heuristic-assn*])
apply (*drule remove-component-right*[**where** *X* = *aivdom-assn*])
apply (*drule remove-component-right*[**where** *X* = *lcount-assn*])
apply (*drule remove-component-right*[**where** *X* = *opts-assn*])
apply (*drule remove-component-right*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-right*[**where** *X* = *occs-assn*])
by (*rule hfref-cong*, *assumption*) *auto*

sublocale *XX* : *read-all-param-adder* **where**
f = ‹(λ*C M N* - - - - - - - - - - - - - - - -. *f C M N*)› **and**
f' = ‹(λ*C M N* - - - - - - - - - - - - - - - -. *f' C M N*)› **and**
P = ‹(λ*C M N* - - - - - - - - - - - - - - - -. *P C M N*)›
by *unfold-locales*
(*rule not-deleted-code-refine'*)

lemmas *refine* = *XX.refine*

lemma *mop-refine*:
‹(*uncurry* (λ*N C*. *read-all-st-code* (λ*M N* - - - - - - - - - - - - - - - -. *f C M N*) *N*),
uncurry mop)
∈ *isasat-bounded-assn*^{k} $*_a$ (*pure R*)$^k \to_a$ *x-assn*›
unfolding *mop-def*
apply (*rule refine-ASSERT-move-to-pre*)
apply (*rule refine*)
done
end

locale *read-trail-arena-param-adder2-twoargs-ops* =
fixes
f' :: ‹'*b* ⇒ '*d* ⇒ *trail-pol* ⇒ *arena* ⇒ '*r nres*› **and**
P :: ‹'*b* ⇒ '*d* ⇒ *trail-pol* ⇒ *arena* ⇒ *bool*›
begin
definition *mop* **where**
‹*mop N C C'* = *do* {
ASSERT (*P C C'* (*get-trail-wl-heur N*) (*get-clauses-wl-heur N*));
read-all-st (λ*M N* - - - - - - - - - - - - - - - -. *f' C C' M N*) *N*
}›
end

locale *read-trail-arena-param-adder2-twoargs* =
read-trail-arena-param-adder2-twoargs-ops f' P
for *f'* :: ‹'*b* ⇒ '*d* ⇒ *trail-pol* ⇒ *arena* ⇒ '*r nres*› **and** *P* +

fixes R **and** R' **and** f **and** *x-assn* :: ‹$'r \Rightarrow 'q \Rightarrow$ *assn*›
assumes *not-deleted-code-refine*:
‹(*uncurry3* ($\lambda S\ T\ C\ D$. $f\ C\ D\ S\ T$), *uncurry3* ($\lambda S\ T\ C'\ D'$. $f'\ C'\ D'\ S\ T$)) ∈
[*uncurry3* ($\lambda S\ T\ C'\ D'$. $P\ C'\ D'\ S\ T$)]$_a$ *trail-pol-fast-assn*^{k} $*_a$ *arena-fast-assn*^{k} $*_a$ (*pure* R)^k $*_a$ (*pure* R')^k $\to$ *x-assn*›
begin

sublocale *XX*: *read-trail-arena-param-adder* **where**
f = ‹$\lambda(C,D)\ N.\ f\ C\ D\ N$› **and**
f' = ‹$\lambda(C,D)\ N.\ f'\ C\ D\ N$› **and**
P = ‹$\lambda(C,D)\ N.\ P\ C\ D\ N$› **and**
R = ‹$R \times_f R'$›
apply *unfold-locales*
by (*rule hfref-cong*[*OF not-deleted-code-refine*[*THEN merge-second-pure-argument*]]) *auto*

lemmas *refine* = *XX.refine*[*unfolded case-isasat-int-split-getter*]

lemma *mop-refine*:
‹(*uncurry2* ($\lambda N\ C\ D$. *read-all-st-code*
($\lambda M\ N$ - - - - - - - - - - - - - - - -. $f\ C\ D\ M\ N$) N), *uncurry2 mop*) ∈ *isasat-bounded-assn*^{k} $*_a$ (*pure* R)^k $*_a$ (*pure* R')^k $\to_a$ *x-assn*›
unfolding *mop-def*
apply (*rule refine-ASSERT-move-to-pre2*)
apply (*rule refine*[*THEN split-snd-pure-arg*, *unfolded prod.case*])
done
end

locale *read-trail-arena-param-adder2-threeargs-ops* =
fixes
f' :: ‹$'b \Rightarrow 'd \Rightarrow 'e \Rightarrow$ *trail-pol* $\Rightarrow$ *arena* $\Rightarrow 'r$ *nres*› **and**
P :: ‹$'b \Rightarrow 'd \Rightarrow 'e \Rightarrow$ *trail-pol* $\Rightarrow$ *arena* $\Rightarrow$ *bool*›
begin
definition *mop* **where**
‹*mop* $N\ C\ D\ E$ = *do* {
ASSERT ($P\ C\ D\ E$ (*get-trail-wl-heur* N) (*get-clauses-wl-heur* N));
read-all-st ($\lambda M\ N$ - - - - - - - - - - - - - - - -. $f'\ C\ D\ E\ M\ N$) N
}›
end

lemma *refine-ASSERT-move-to-pre3*:
assumes ‹(*uncurry3* g, *uncurry3* h) ∈ [*uncurry3* P]$_a$ A $*_a$ B $*_a$ C $*_a$ $D \to$ *x-assn*›
shows
‹(*uncurry3* g, *uncurry3* ($\lambda N\ C\ D\ E$. *do* {*ASSERT* ($P\ N\ C\ D\ E$); $h\ N\ C\ D\ E$}))
∈ A $*_a$ B $*_a$ C $*_a$ $D \to_a$ *x-assn*›
apply *sepref-to-hoare*
apply *vcg*
apply (*subst POSTCOND-def hn-ctxt-def sep-conj-empty' pure-true-conv EXTRACT-def*)+
apply (*auto simp*: *nofail-ASSERT-bind*)
apply (*rule assms*[*to-hnr*, *simplified*, *unfolded hn-ctxt-def hn-refine-def htriple-def sep-conj-empty' pure-true-conv sep.add-assoc*, *rule-format*])
apply *auto*
done

locale *read-trail-arena-param-adder2-threeargs* =
read-trail-arena-param-adder2-threeargs-ops f' P

for $f' :: \langle 'b \Rightarrow 'd \Rightarrow 'e \Rightarrow$ *trail-pol* $\Rightarrow$ *arena* $\Rightarrow 'r$ *nres*$\rangle$ **and** P +
fixes R **and** R' **and** R'' **and** f **and** *x-assn* $:: \langle 'r \Rightarrow 'q \Rightarrow$ *assn*$\rangle$
assumes *not-deleted-code-refine*:
$\langle$(*uncurry4* ($\lambda S\ T\ C\ D\ E.\ f\ C\ D\ E\ S\ T$), *uncurry4* ($\lambda S\ T\ C'\ D'\ E'.\ f'\ C'\ D'\ E'\ S\ T$)) $\in$
[*uncurry4* ($\lambda S\ T\ C'\ D'\ E'.\ P\ C'\ D'\ E'\ S\ T$)]$_a$ *trail-pol-fast-assn*^{k} $*_a$ *arena-fast-assn*^{k} $*_a$ (*pure* R)^k $*_a$ (*pure* R')^k $*_a$ (*pure* R'')$^k \to$ *x-assn*$\rangle$
begin
sublocale *XX*: *read-trail-arena-param-adder* **where**
$f = \langle\lambda(C,D,E)\ N.\ f\ C\ D\ E\ N\rangle$ **and**
$f' = \langle\lambda(C,D,E)\ N.\ f'\ C\ D\ E\ N\rangle$ **and**
$P = \langle\lambda(C,D,E)\ N.\ P\ C\ D\ E\ N\rangle$ **and**
$R = \langle R \times_r R' \times_r R''\rangle$
apply *unfold-locales*
subgoal
using *not-deleted-code-refine*[*THEN merge-second-pure-argument*]
by (*auto simp flip*: *prod-assn-pure-conv simp*: *hfref-def*)
done

It would be better to this without calling auto, but fighting uncurry is just too hard and not really useful.

lemma *refine*:
$\langle$(*uncurry3* ($\lambda N\ C\ D\ E.$ *read-all-st-code* ($\lambda M\ N$ - - - - - - - - - - - - - - -. $f\ C\ D\ E\ M\ N$) N),
uncurry3 ($\lambda N\ C\ D\ E.$ *read-all-st* ($\lambda M\ N$ - - - - - - - - - - - - - - -. $f'\ C\ D\ E\ M\ N$) N))
$\in$ [*uncurry3* ($\lambda S\ C'\ D'\ E'.\ P\ C'\ D'\ E'$ (*get-trail-wl-heur* S) (*get-clauses-wl-heur* S))]$_a$
isasat-bounded-assn^k $*_a$ (*pure* R)^k $*_a$ (*pure* R')^k $*_a$ (*pure* R'')$^k \to$ *x-assn*$\rangle$
using *XX.refine*[*THEN split-snd-pure-arg*, *unfolded prod.case*] **unfolding** *hfref-def*
by (*auto simp flip*: *prod-assn-pure-conv*)
lemma *mop-refine*:
$\langle$(*uncurry3* ($\lambda N\ C\ D\ E.$ *read-all-st-code*
($\lambda M\ N$ - - - - - - - - - - - - - - -. $f\ C\ D\ E\ M\ N$) N), *uncurry3 mop*) $\in$ *isasat-bounded-assn*^{k} $*_a$ (*pure* R)^k $*_a$ (*pure* R')^k $*_a$ (*pure* R'')$^k \to_a$ *x-assn*$\rangle$
unfolding *mop-def*
apply (*rule refine-ASSERT-move-to-pre3*)
apply (*rule refine*)
done

end

locale *read-trail-vmtf-param-adder* =
fixes $P :: \langle 'b \Rightarrow$ - $\Rightarrow$ *bump-heuristics* $\Rightarrow$ *bool*$\rangle$ **and** $f' :: \langle 'b \Rightarrow$ *trail-pol* $\Rightarrow$ - $\Rightarrow 'r$ *nres*$\rangle$ **and**
$R :: \langle('a \times 'b)$ *set*$\rangle$ **and** f **and** *x-assn* $:: \langle 'r \Rightarrow 'q \Rightarrow$ *assn*$\rangle$
assumes *not-deleted-code-refine*: $\langle$(*uncurry2* ($\lambda S\ T\ C.\ f\ C\ S\ T$), *uncurry2* ($\lambda S\ T\ C'.\ f'\ C'\ S\ T$)) $\in$ [*uncurry2* ($\lambda S\ T\ C.\ P\ C\ S\ T$)]$_a$ *trail-pol-fast-assn*^{k} $*_a$ *heuristic-bump-assn*^{k} $*_a$ (*pure* R)$^k \to$ *x-assn*$\rangle$
begin

lemma *not-deleted-code-refine'*:
$\langle$(*uncurry17* (λM - - - - N - - - - - - - - - - - $C.\ f\ C\ M\ N$), *uncurry17* (λM - - - - N - - - - - - - - - - - $C'.\ f'\ C'\ M\ N$)) $\in$
[*uncurry17* (λM - - - - N - - - - - - - - - - - $C'.\ P\ C'\ M\ N$)]$_a$
trail-pol-fast-assn^k $*_a$ *arena-fast-assn*^{k} $*_a$ *conflict-option-rel-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$
watchlist-fast-assn^k $*_a$ *heuristic-bump-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $*_a$ *cach-refinement-l-assn*^{k} $*_a$
lbd-assn^k $*_a$ *out-learned-assn*^{k} $*_a$ *isasat-stats-assn*^{k} $*_a$ *heuristic-assn*^{k} $*_a$
aivdom-assn^k $*_a$ *lcount-assn*^{k} $*_a$ *opts-assn*^{k} $*_a$ *arena-fast-assn*^{k} $*_a$ *occs-assn*^{k} $*_a$ (*pure* R)$^k \to$ *x-assn*$\rangle$
apply (*rule add-pure-parameter2*)
apply (*drule remove-pure-parameter2'*[*OF not-deleted-code-refine*])

apply (*drule remove-component-middle*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-middle*[**where** *X* = *conflict-option-rel-assn*])
apply (*drule remove-component-middle*[**where** *X* = *sint64-nat-assn*])
apply (*drule remove-component-middle*[**where** *X* = *watchlist-fast-assn*])
apply (*drule remove-component-right*[**where** *X* = *uint32-nat-assn*])
apply (*drule remove-component-right*[**where** *X* = *cach-refinement-l-assn*])
apply (*drule remove-component-right*[**where** *X* = *lbd-assn*])
apply (*drule remove-component-right*[**where** *X* = *out-learned-assn*])
apply (*drule remove-component-right*[**where** *X* = *isasat-stats-assn*])
apply (*drule remove-component-right*[**where** *X* = *heuristic-assn*])
apply (*drule remove-component-right*[**where** *X* = *aivdom-assn*])
apply (*drule remove-component-right*[**where** *X* = *lcount-assn*])
apply (*drule remove-component-right*[**where** *X* = *opts-assn*])
apply (*drule remove-component-right*[**where** *X* = *arena-fast-assn*])
apply (*drule remove-component-right*[**where** *X* = *occs-assn*])
by (*rule hfref-cong*, *assumption*) *auto*

sublocale *XX* : *read-all-param-adder* **where**
f = ‹(λ*C M - - - - N - - - - - - - - - - -*. *f C M N*)› **and**
f′ = ‹(λ*C M - - - - N - - - - - - - - - - -*. *f′ C M N*)› **and**
P = ‹(λ*C M - - - - N - - - - - - - - - - -*. *P C M N*)›
by *unfold-locales*
(*rule not-deleted-code-refine′*)

lemmas *refine* = *XX.refine*
end

lemma *hn-refine-frame′′′*: *hn-refine* (*F****G*) *c* (*F′****G′*) *R CP m* ⟹ *hn-refine* (*F****G****H*) *c* (*F′****G′****H*) *R CP m*
by (*metis hn-refine-frame-mid′′ sep.mult-commute*)

locale *read-trail-vmtf-param-adder0* =
fixes *f* **and** *f′* **and** *x-assn* :: ‹′*r* ⇒ ′*q* ⇒ *assn*› **and** *P*
assumes *not-deleted-code-refine*: ‹(*uncurry* (λ*S T*. *f S T*), *uncurry* (λ*S T*. *f′ S T*)) ∈ [*uncurry* (λ*S T*. *P S T*)]$_a$ *trail-pol-fast-assn*^{k} *$_a$ *heuristic-bump-assn*^{k} → *x-assn*›
begin

sublocale *XX*: *read-trail-vmtf-param-adder* **where**
f = ‹λ-::*unit*. *f*› **and**
f′ = ‹λ-::*unit*. *f′*› **and**
P = ‹λ-::*unit*. *P*› **and**
R = ‹*unit-rel*›
apply *unfold-locales*
using *not-deleted-code-refine* **apply** −
unfolding *hfref-def*
apply *clarsimp*
apply (*rule hn-refine-frame′′′*)
apply *auto*
done

lemmas *refine* = *XX.refine*[*THEN remove-unused-unit-parameter*]
end

lemma *Mreturn-comp-IsaSAT-int*:

‹(*Mreturn* o_{17} *IsaSAT-int*) *a b c d e f g h i j k l m n ko p q* =
Mreturn (*IsaSAT-int a b c d e f g h i j k l m n ko p q*)›
by *auto*

sepref-register
remove-a remove-b remove-c remove-d
remove-e remove-f remove-g remove-h
remove-i remove-j remove-k remove-l
remove-m remove-n remove-o remove-p remove-q

lemmas [*sepref-fr-rules*] =
remove-a-code.refine
remove-b-code.refine
remove-c-code.refine
remove-d-code.refine
remove-e-code.refine
remove-f-code.refine
remove-g-code.refine
remove-h-code.refine
remove-i-code.refine
remove-j-code.refine
remove-k-code.refine
remove-l-code.refine
remove-m-code.refine
remove-n-code.refine
remove-o-code.refine
remove-p-code.refine
remove-q-code.refine

lemma *lambda-comp-true*: ‹(λ*S*. *True*) ∘ *f* = (λ-. *True*)› ‹*uncurry* (λ*a b*. *True*) = (λ-. *True*)› ‹*uncurry2* (λ*a b c*. *True*) = (λ-. *True*)›
‹*case-isasat-int* (λ*M* - - - - - - - - - - - - - - - - -. *True*) = (λ-. *True*)›
by (*auto intro*!: *ext split*: *isasat-int-splits*)

named-theorems *state-extractors* ‹*Definition of all functions modifying the state*›
lemmas [*state-extractors*] =
extract-trail-wl-heur-def
extract-arena-wl-heur-def
extract-conflict-wl-heur-def
extract-watchlist-wl-heur-def
extract-vmtf-wl-heur-def
extract-ccach-wl-heur-def
extract-clvls-wl-heur-def
extract-lbd-wl-heur-def
extract-outl-wl-heur-def
extract-stats-wl-heur-def
extract-heur-wl-heur-def
extract-vdom-wl-heur-def
extract-lcount-wl-heur-def
extract-opts-wl-heur-def
extract-literals-to-update-wl-heur-def
extract-occs-wl-heur-def
tuple17-getters-setters

lemmas [*llvm-inline*] =
DEFAULT-INIT-PHASE-def
FOCUSED-MODE-def

lemma *from-bool1*: *from-bool True* = *1*
by *auto*
lemmas [*llvm-pre-simp*] = *from-bool1*

lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
remove-a-code-alt-def[*unfolded isasat-state.remove-a-code-alt-def Mreturn-comp-IsaSAT-int*]
remove-b-code-alt-def[*unfolded isasat-state.remove-b-code-alt-def Mreturn-comp-IsaSAT-int*]
remove-c-code-alt-def[*unfolded isasat-state.remove-c-code-alt-def Mreturn-comp-IsaSAT-int*]
remove-d-code-alt-def[*unfolded isasat-state.remove-d-code-alt-def Mreturn-comp-IsaSAT-int*]
remove-e-code-alt-def[*unfolded isasat-state.remove-e-code-alt-def Mreturn-comp-IsaSAT-int*]
remove-f-code-alt-def[*unfolded isasat-state.remove-f-code-alt-def Mreturn-comp-IsaSAT-int*]
remove-g-code-alt-def[*unfolded isasat-state.remove-g-code-alt-def Mreturn-comp-IsaSAT-int*]
remove-h-code-alt-def[*unfolded isasat-state.remove-h-code-alt-def Mreturn-comp-IsaSAT-int*]
remove-i-code-alt-def[*unfolded isasat-state.remove-i-code-alt-def Mreturn-comp-IsaSAT-int*]
remove-j-code-alt-def[*unfolded isasat-state.remove-j-code-alt-def Mreturn-comp-IsaSAT-int*]
remove-k-code-alt-def[*unfolded isasat-state.remove-k-code-alt-def Mreturn-comp-IsaSAT-int*]
remove-l-code-alt-def[*unfolded isasat-state.remove-l-code-alt-def Mreturn-comp-IsaSAT-int*]
remove-m-code-alt-def[*unfolded isasat-state.remove-m-code-alt-def Mreturn-comp-IsaSAT-int*]
remove-n-code-alt-def[*unfolded isasat-state.remove-n-code-alt-def Mreturn-comp-IsaSAT-int*]
remove-o-code-alt-def[*unfolded isasat-state.remove-o-code-alt-def Mreturn-comp-IsaSAT-int*]
remove-p-code-alt-def[*unfolded isasat-state.remove-p-code-alt-def Mreturn-comp-IsaSAT-int*]
remove-q-code-alt-def[*unfolded isasat-state.remove-q-code-alt-def Mreturn-comp-IsaSAT-int*]

abbreviation *update-trail-wl-heur* :: ‹*trail-pol* ⇒ *isasat* ⇒ -› **where**
‹*update-trail-wl-heur* ≡ *update-a*›

abbreviation *update-arena-wl-heur* :: ‹*arena* ⇒*isasat* ⇒ -› **where**
‹*update-arena-wl-heur* ≡ *update-b*›

abbreviation *update-conflict-wl-heur* :: ‹*conflict-option-rel* ⇒*isasat* ⇒ -› **where**
‹*update-conflict-wl-heur* ≡ *update-c*›

abbreviation *update-literals-to-update-wl-heur* :: ‹*nat* ⇒*isasat* ⇒ -› **where**
‹*update-literals-to-update-wl-heur* ≡ *update-d*›

abbreviation *update-watchlist-wl-heur* :: ‹*nat watcher list list* ⇒*isasat* ⇒ -› **where**
‹*update-watchlist-wl-heur* ≡ *update-e*›

abbreviation *update-vmtf-wl-heur* :: ‹*bump-heuristics* ⇒*isasat* ⇒ -› **where**
‹*update-vmtf-wl-heur* ≡ *update-f*›

abbreviation *update-clvls-wl-heur* :: ‹*nat* ⇒*isasat* ⇒ -› **where**
‹*update-clvls-wl-heur* ≡ *update-g*›

abbreviation *update-ccach-wl-heur* :: ‹*conflict-min-cach-l* ⇒*isasat* ⇒ -› **where**
‹*update-ccach-wl-heur* ≡ *update-h*›

abbreviation *update-lbd-wl-heur* :: ‹*lbd* ⇒*isasat* ⇒ -› **where**
‹*update-lbd-wl-heur* ≡ *update-i*›

abbreviation *update-outl-wl-heur* :: ‹*out-learned* ⇒*isasat* ⇒ -› **where**

‹*update-outl-wl-heur* ≡ *update-j*›

abbreviation *update-stats-wl-heur* :: ‹*isasat-stats* ⇒ *isasat* ⇒ *isasat*› **where**
‹*update-stats-wl-heur* ≡ *update-k*›

abbreviation *update-heur-wl-heur* :: ‹*isasat-restart-heuristics* ⇒*isasat* ⇒ *isasat*› **where**
‹*update-heur-wl-heur* ≡ *update-l*›

abbreviation *update-vdom-wl-heur* :: ‹*isasat-aivdom* ⇒*isasat* ⇒ -› **where**
‹*update-vdom-wl-heur* ≡ *update-m*›

abbreviation *update-lcount-wl-heur* :: ‹*clss-size* ⇒*isasat* ⇒ -› **where**
‹*update-lcount-wl-heur* ≡ *update-n*›

abbreviation *update-opts-wl-heur* :: ‹*opts* ⇒*isasat* ⇒ -› **where**
‹*update-opts-wl-heur* ≡ *update-o*›

abbreviation *update-old-arena-wl-heur* :: ‹*arena* ⇒*isasat* ⇒ -› **where**
‹*update-old-arena-wl-heur* ≡ *update-p*›

abbreviation *update-occs-wl-heur* :: ‹*occurences-ref* ⇒*isasat* ⇒ -› **where**
‹*update-occs-wl-heur* ≡ *update-q*›
end
theory *IsaSAT-Setup1-LLVM*
imports
IsaSAT-Setup
IsaSAT-Setup0-LLVM
begin

sepref-register *arena-status DELETED*
sepref-definition *not-deleted-code*
is ‹(*uncurry* (λ*N C′*. *do* {*status* ← *RETURN* (*arena-status N C′*); *RETURN* (*status* ≠ *DELETED*)}))›
:: ‹[*uncurry* (λ*N C′*. *arena-is-valid-clause-vdom N C′*)]$_a$ *arena-fast-assn*^{k} *∗$_a$ sint64-nat-assn*^{k} → *bool1-assn*›
by *sepref*

lemma *clause-not-marked-to-delete-heur-alt-def*:
‹*RETURN oo clause-not-marked-to-delete-heur* = (λ *S C′*. *read-arena-wl-heur* (λ*N*. *do* {*status* ← *RETURN* (*arena-status N C′*); *RETURN* (*status* ≠ *DELETED*)}) *S*)›
by (*auto intro*!: *ext simp*: *clause-not-marked-to-delete-heur-def read-all-st-def*
split: *isasat-int-splits*)

definition *clause-not-marked-to-delete-heur-code* :: ‹*twl-st-wll-trail-fast2* ⇒ - ⇒ -› **where**
‹*clause-not-marked-to-delete-heur-code S C′* = *read-arena-wl-heur-code* (λ*N*. *not-deleted-code N C′*) *S*›

global-interpretation *arena-is-valid*: *read-arena-param-adder* **where**
R = ‹*snat-rel′ TYPE*(*64*)› **and**
f = ‹λ*C N*. *not-deleted-code N C*› **and**
f′ = ‹(λ*C′ N*. *do* {*status* ← *RETURN* (*arena-status N C′*); *RETURN* (*status* ≠ *DELETED*)})› **and**
x-assn = *bool1-assn* **and**
P = ‹(λ*C S*. *arena-is-valid-clause-vdom S C*)›
rewrites ‹(λ*S C′*. *read-arena-wl-heur* (λ*N*. *do* {*status* ← *RETURN* (*arena-status N C′*); *RETURN* (*status* ≠ *DELETED*)}) *S*) = *RETURN oo clause-not-marked-to-delete-heur*› **and**
‹(λ*S C′*. *read-arena-wl-heur-code* (λ*N*. *not-deleted-code N C′*) *S*) = *clause-not-marked-to-delete-heur-code*›
and
‹(λ*S*. *arena-is-valid-clause-vdom* (*get-clauses-wl-heur S*)) = *curry clause-not-marked-to-delete-heur-pre*›

apply *unfold-locales*
apply (*rule not-deleted-code.refine*)
unfolding *clause-not-marked-to-delete-heur-alt-def clause-not-marked-to-delete-heur-code-def*
apply (*solves ‹rule refl›*)+
subgoal by (*auto simp*: *clause-not-marked-to-delete-heur-pre-def*)
done

sepref-register *clause-not-marked-to-delete-heur*
lemmas [*sepref-fr-rules*] = *arena-is-valid.refine*[*unfolded uncurry-curry-id*]
lemmas [*llvm-code*] = *clause-not-marked-to-delete-heur-code-def*[*unfolded read-all-st-code-def not-deleted-code-def*]

sepref-def *mop-clause-not-marked-to-delete-heur-impl*
is ‹*uncurry mop-clause-not-marked-to-delete-heur*›
:: ‹*isasat-bounded-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *bool1-assn*›
unfolding *mop-clause-not-marked-to-delete-heur-def*
prod.case clause-not-marked-to-delete-heur-pre-def[*symmetric*]
by *sepref*

definition *conflict-is-None* :: ‹*conflict-option-rel* ⇒ *bool nres*› **where**
‹*conflict-is-None* =(λ*N*. *do* {*let* (*b*, -) = *N*; *RETURN b*})›
definition ‹*conflict-is-None-code* :: *option-lookup-clause-assn* ⇒ *1 word llM* ≡
λ(*a*, -). *do*$_M$ {
return$_M$ (*a*)
}›
lemma *conflict-is-None-code-refine*[*sepref-fr-rules*]:
‹(*conflict-is-None-code*, *conflict-is-None*) ∈ *conflict-option-rel-assn*^{k} $\rightarrow_a$ *bool1-assn*›
unfolding *conflict-is-None-code-def conflict-is-None-def Let-def conflict-option-rel-assn-def*
apply *sepref-to-hoare*
apply *vcg*
done

lemma *get-conflict-wl-is-None-heur-alt-def*: ‹*read-conflict-wl-heur conflict-is-None* = *RETURN* ∘ *get-conflict-wl-is-None*
by (*auto simp*: *read-all-st-def get-conflict-wl-is-None-heur-def conflict-is-None-def*
intro!: *ext split*: *isasat-int-splits*)

definition *get-conflict-wl-is-None-heur2* **where**
‹*get-conflict-wl-is-None-heur2* = *RETURN o get-conflict-wl-is-None-heur*›

definition *get-conflict-wl-is-None-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*get-conflict-wl-is-None-fast-code* = *read-conflict-wl-heur-code conflict-is-None-code*›

global-interpretation *conflict-is-None*: *read-conflict-param-adder0* **where**
f = ‹*conflict-is-None-code*› **and**
f' = ‹*conflict-is-None*› **and**
x-assn = *bool1-assn* **and**
P = ‹(λ*S*. *True*)›
rewrites ‹(λ*S*. *read-conflict-wl-heur* (*conflict-is-None*) *S*) = *get-conflict-wl-is-None-heur2*› **and**
‹(λ*S*. *read-conflict-wl-heur-code* (*conflict-is-None-code*) *S*) = *get-conflict-wl-is-None-fast-code*›
apply *unfold-locales*
apply (*rule conflict-is-None-code-refine*; *assumption*)
unfolding *get-conflict-wl-is-None-heur2-def get-conflict-wl-is-None-fast-code-def*
by (*solves ‹rule get-conflict-wl-is-None-heur-alt-def refl›*)+

lemmas [*sepref-fr-rules*] = *conflict-is-None.refine*[*unfolded get-conflict-wl-is-None-heur2-def*]

lemmas [*llvm-code*] = *conflict-is-None-code-def*
lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
get-conflict-wl-is-None-fast-code-def[*unfolded read-all-st-code-def*]

lemma *count-decided-st-heur-alt-def*:
‹*RETURN o count-decided-st-heur* = *read-trail-wl-heur* (*RETURN* ∘ *count-decided-pol*)›
by (*auto intro*!: *ext simp*: *count-decided-st-heur-def count-decided-pol-def*
read-all-st-def split: *isasat-int-splits*)

definition *count-decided-st-heur-impl* **where**
‹*count-decided-st-heur-impl* = *read-trail-wl-heur-code count-decided-pol-impl*›

sepref-register *extract-trail-wl-heur count-decided-pol count-decided-st-heur*

definition *count-decided-st-heur-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*count-decided-st-heur-fast-code* = *read-trail-wl-heur-code count-decided-pol-impl*›

global-interpretation *count-decided*: *read-trail-param-adder0* **where**
f = ‹*count-decided-pol-impl*› **and**
f' = ‹*RETURN o count-decided-pol*› **and**
x-assn = *uint32-nat-assn* **and**
P = ‹(λ*S*. *True*)›
rewrites ‹*read-trail-wl-heur* (*RETURN o count-decided-pol*) = *RETURN o count-decided-st-heur*›
and
‹*read-trail-wl-heur-code count-decided-pol-impl* = *count-decided-st-heur-fast-code*›
apply *unfold-locales*
apply (*rule count-decided-pol-impl.refine*)
subgoal
by (*auto simp*: *read-all-st-def count-decided-st-heur-def intro*!: *ext*
split: *isasat-int-splits*)
subgoal
by (*auto simp*: *count-decided-st-heur-fast-code-def*)
done

lemmas [*sepref-fr-rules*] = *count-decided.refine*[*unfolded lambda-comp-true*]
lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
count-decided-st-heur-fast-code-def[*unfolded read-all-st-code-def*]

definition *polarity-st-heur-pol-fast* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*polarity-st-heur-pol-fast* = (λ*S C*. *read-trail-wl-heur-code* (λ*L*. *polarity-pol-fast L C*) *S*)›

global-interpretation *mop-count-decided*: *read-trail-param-adder* **where**
f = ‹λ*S L*. *polarity-pol-fast L S*› **and**
f' = ‹λ*S L*. *mop-polarity-pol L S*› **and**
x-assn = *tri-bool-assn* **and**
P = ‹λ*S* -. *True*› **and**
R = ‹*unat-lit-rel*›
rewrites ‹(λ*S C*. *read-trail-wl-heur-code* (λ*L*. *polarity-pol-fast L C*) *S*) = *polarity-st-heur-pol-fast*›
and
‹(λ*S C'*. *read-trail-wl-heur* (λ*L*. *mop-polarity-pol L C'*) *S*) = *mop-polarity-st-heur*›
apply *unfold-locales*
apply (*subst lambda-comp-true*)
apply (*rule polarity-pol-fast.refine*)
subgoal

by (*auto simp*: *polarity-st-heur-pol-fast-def*)
subgoal
by (*auto simp*: *mop-polarity-st-heur-def read-all-st-def*
split: *isasat-int-splits intro*!: *ext*)
done

lemmas [*sepref-fr-rules*] = *mop-count-decided.refine*[*unfolded lambda-comp-true*]
lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
polarity-st-heur-pol-fast-def[*unfolded read-all-st-code-def*]

definition *arena-lit2* **where** ‹*arena-lit2 N i j* = *arena-lit N* (*i*+*j*)›

sepref-def *arena-lit2-impl*
is ‹*uncurry2* (*RETURN ooo arena-lit2*)›
:: ‹[*uncurry2* (λ*N i j*. *arena-lit-pre N* (*i*+*j*) ∧ *length N* ≤ *snat64-max*)]$_a$ *arena-fast-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} → *unat-lit-assn*›
supply [*dest*] = *arena-lit-implI*
unfolding *arena-lit-def arena-lit2-def*
by *sepref*

lemma *arena-lit2-impl-arena-lit*:
assumes ‹(*C*, *C'*) ∈ *snat-rel*› **and**
‹(*D*, *D'*) ∈ *snat-rel*›
shows ‹(λ*S*. *arena-lit2-impl S C D*, λ*S*. *RETURN* (*arena-lit S* (*C'* + *D'*)))
∈ [λ*S*. *arena-lit-pre S* (*C'* + *D'*) ∧ *length S* ≤ *snat64-max*]$_a$ (*al-assn arena-el-impl-assn*)^k → *unat-lit-assn*›
proof −
have *arena-lit2*: ‹*RETURN ooo arena-lit2* = (λ*S C' D'*. *RETURN* (*arena-lit2 S C' D'*))› **for** *f*
by (*auto intro*!: *ext*)
show *?thesis*
apply (*rule remove-pure-parameter2-twoargs*[**where** *R* = ‹*snat-rel'* *TYPE*(*64*)› **and** *f* = ‹λ*a C D*. *arena-lit2-impl a C D*› **and** *f'* =
‹λ*S C' D'*. *RETURN* (*arena-lit S* (*C'* + *D'*))›, *OF* - *assms*])
unfolding *arena-lit2-def*[*symmetric*] *arena-lit2*[*symmetric*]
by (*rule arena-lit2-impl.refine*)
qed

lemma *access-lit-in-clauses-heur-alt-def*:
‹*RETURN ooo access-lit-in-clauses-heur* = (λ*N C' D*. *read-arena-wl-heur* (λ*N*. *RETURN* (*arena-lit N* (*C'* + *D*))) *N*)›
by (*auto intro*!: *ext simp*: *read-all-st-def access-lit-in-clauses-heur-def split*: *isasat-int-splits*)

lemma *access-lit-in-clauses-heur-pre*:
‹*uncurry2*
(λ*S C D*.
arena-lit-pre (*get-clauses-wl-heur S*) (*C* + *D*) ∧
length (*get-clauses-wl-heur S*) ≤ *snat64-max*) = *uncurry2* (λ*S i j*. *access-lit-in-clauses-heur-pre* ((*S*, *i*), *j*) ∧
length (*get-clauses-wl-heur S*) ≤ *snat64-max*)›
by (*auto simp*: *access-lit-in-clauses-heur-pre-def*)

definition *access-lit-in-clauses-heur-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ - ⇒ - ⇒ -› **where**
‹*access-lit-in-clauses-heur-fast-code* = (λ*N C D*. *read-arena-wl-heur-code* (λ*N*. *arena-lit2-impl N C D*) *N*)›

definition *mop-arena-lit2-st* **where**

‹*mop-arena-lit2-st S = mop-arena-lit2 (get-clauses-wl-heur S)*›

global-interpretation *access-arena*: *read-arena-param-adder2-twoargs* **where**
R = ‹(*snat-rel′ TYPE*(*64*))› **and**
R′ = ‹*snat-rel′ TYPE*(*64*)› **and**
f′ = ‹*λi j N. RETURN* (*arena-lit N* (*i*+*j*))› **and**
f = ‹*λi j N. arena-lit2-impl N i j*› **and**
x-assn = *unat-lit-assn* **and**
P = ‹(*λi j S. arena-lit-pre* (*S*) (*i*+*j*) ∧ *length S* ≤ *snat64-max*)›
rewrites
‹(*λN C′ D. read-arena-wl-heur* (*λN. RETURN* (*arena-lit N* (*C′* + *D*))) *N*) = *RETURN ooo access-lit-in-clauses-heur*› **and**
‹(*λN C D. read-arena-wl-heur-code* (*λN. arena-lit2-impl N C D*) *N*) = *access-lit-in-clauses-heur-fast-code*› **and**
‹*uncurry2* (*λS C D. arena-lit-pre* (*get-clauses-wl-heur S*) (*C* + *D*) ∧ *length* (*get-clauses-wl-heur S*) ≤ *snat64-max*) =
uncurry2 (*λS i j. access-lit-in-clauses-heur-pre* ((*S*, *i*), *j*) ∧ *length* (*get-clauses-wl-heur S*) ≤ *snat64-max*)›

apply *unfold-locales*
apply (*rule arena-lit2-impl.refine*[*unfolded comp-def arena-lit2-def*])
apply (*subst access-lit-in-clauses-heur-alt-def*; *rule refl*)
apply (*subst access-lit-in-clauses-heur-fast-code-def*; *rule refl*)
apply (*rule access-lit-in-clauses-heur-pre*)
done

lemma *refine-ASSERT-move-to-pre2′*:
‹(*uncurry2 g*, *uncurry2 h*) ∈ [*uncurry2* (*λa b c. P a b c* ∧ *Q a b c*)]$_a$ *A* ∗$_a$ *B* ∗$_a$ *C* → *x-assn* ⟷ (*uncurry2 g*, *uncurry2* (*λN C D. do* {*ASSERT* (*P N C D*); *h N C D*}))
∈ [*uncurry2 Q*]$_a$ *A* ∗$_a$ *B* ∗$_a$ *C* → *x-assn*›
apply (*rule iffI*)
subgoal premises *p*
apply *sepref-to-hoare*
apply *vcg*
apply (*subst POSTCOND-def hn-ctxt-def sep-conj-empty′ pure-true-conv EXTRACT-def*)+
apply (*auto simp*: *nofail-ASSERT-bind hn-ctxt-def*)
apply (*rule p*[*to-hnr*, *simplified*, *unfolded hn-ctxt-def hn-refine-def htriple-def sep-conj-empty′ pure-true-conv sep.add-assoc*, *rule-format*])
apply *auto*
done
subgoal premises *p*
apply *sepref-to-hoare*
apply *vcg*
subgoal for *b bi ba bia a ai asf s*
apply (*subst POSTCOND-def hn-ctxt-def sep-conj-empty′ pure-true-conv EXTRACT-def*)+
using *p*[*to-hnr*, *simplified*, *unfolded hn-ctxt-def hn-refine-def htriple-def sep-conj-empty′ pure-true-conv sep.add-assoc*, *rule-format*, *of a ba b*]
apply *auto*
done
done
done

lemma *arena-lit-arena-lit-read-arena-wl-heur-arena-lit*:
‹*RETURN* (*arena-lit* (*get-clauses-wl-heur N*) (*C* + *C′*)) = *read-arena-wl-heur* (*λN. RETURN* (*arena-lit N* (*C* + *C′*))) *N*›
by (*auto intro*!: *ext simp*: *read-all-st-def access-lit-in-clauses-heur-def split*: *isasat-int-splits*)

sepref-register *mop-access-lit-in-clauses-heur*
lemma *mop-access-lit-in-clauses-heur-refine*[*sepref-fr-rules*]:
‹(*uncurry2 access-lit-in-clauses-heur-fast-code*, *uncurry2 mop-access-lit-in-clauses-heur*)
∈ [*uncurry2* (λ*S i j*. *length* (*get-clauses-wl-heur S*) ≤ *snat64-max*)]$_a$ *isasat-bounded-assn*^{k} *$*_a$ *snat-assn*^{k} *$*_a$ *snat-assn*^{k} → *unat-lit-assn*›
using *access-arena.mop-refine*[*unfolded access-arena.mop-def refine-ASSERT-move-to-pre2'*[*symmetric*, **where** *Q* = ‹λ- - -. *True*›, *unfolded simp-thms lambda-comp-true*]]
unfolding *mop-access-lit-in-clauses-heur-def mop-access-lit-in-clauses-heur-def mop-arena-lit2-def Let-def access-arena.mop-def refine-ASSERT-move-to-pre2'*[*symmetric*] *access-lit-in-clauses-heur-alt-def arena-lit-arena-lit-read-arena-wl-heur-arena-lit*[*symmetric*]
by *auto*

lemma *al-assn-boundD2*: ‹*al-assn arena-el-impl-assn x2* (*d*:: *'a* :: *len2 word* × *'a word* × *32 word ptr*) *c* ⟹ *length x2* < *max-snat LENGTH*(*'a*)›
using *al-assn-boundD*[*unfolded rdomp-def*, *of arena-el-impl-assn* ‹*x2*›, **where** *'l* = *'a*]
by (*cases d*) *auto*

lemma *isasat-bounded-assn-length-arenaD*: ‹*rdomp isasat-bounded-assn a* ⟹ *length* (*get-clauses-wl-heur a*) ≤ *snat64-max*› **apply** −
unfolding *rdomp-def*
apply *normalize-goal+*
by (*cases a*, *case-tac xa*)
(*auto simp*: *isasat-bounded-assn-def rdomp-def snat64-max-def max-snat-def split*: *isasat-int-splits dest*!: *al-assn-boundD2 mod-starD*)

sepref-def *mop-access-lit-in-clauses-heur-impl*
is ‹*uncurry2 mop-access-lit-in-clauses-heur*›
:: ‹*isasat-bounded-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} →$_a$ *unat-lit-assn*›
supply [*dest*] = *isasat-bounded-assn-length-arenaD*
by *sepref*

lemmas [*sepref-fr-rules*] = *access-arena.refine*

lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
access-lit-in-clauses-heur-fast-code-def[*unfolded read-all-st-code-def*]

sepref-register *mop-arena-lit2 mop-arena-length mop-append-ll*
sepref-def *rewatch-heur-vdom-fast-code*
is ‹*uncurry2* (*rewatch-heur-vdom*)›
:: ‹[λ((*vdom*, *arena*), *W*). (∀ *x* ∈ *set* (*get-tvdom-aivdom vdom*). *x* ≤ *snat64-max*) ∧ *length arena* ≤ *snat64-max* ∧
length (*get-tvdom-aivdom vdom*) ≤ *snat64-max*]$_a$
aivdom-assn^k *$*_a$ *arena-fast-assn*^{k} *$*_a$ *watchlist-fast-assn*^{d} → *watchlist-fast-assn*›
supply [[*goals-limit*=*1*]]
arena-lit-pre-le-snat64-max[*dest*] *arena-is-valid-clause-idx-le-unat64-max*[*dest*]
supply [*simp*] = *append-ll-def length-tvdom-aivdom-def*
supply [*dest*] = *arena-lit-implI*(*1*)
unfolding *rewatch-heur-alt-def Let-def PR-CONST-def rewatch-heur-vdom-def tvdom-aivdom-at-def*[*symmetric*] *length-tvdom-aivdom-def*[*symmetric*]
unfolding *while-eq-nfoldli*[*symmetric*]
apply (*subst while-upt-while-direct*, *simp*)
unfolding *if-not-swap FOREACH-cond-def FOREACH-body-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)

by *sepref*

lemma *rewatch-heur-st-fast-alt-def*:
‹*rewatch-heur-st-fast* = (λS_0. *do* {
let (*N*, *S*) = *extract-arena-wl-heur* S_0;
let (*W*, *S*) = *extract-watchlist-wl-heur S*;
let (*vdom*, *S*) = *extract-vdom-wl-heur S*;
ASSERT(*length* (*get-tvdom-aivdom vdom*) ≤ *length N*);
ASSERT (*vdom* = *get-aivdom* S_0);
ASSERT (*N* = *get-clauses-wl-heur* S_0);
ASSERT (*W* = *get-watched-wl-heur* S_0);
W ← *rewatch-heur* (*get-tvdom-aivdom vdom*) *N W*;
let S = *update-arena-wl-heur N S*;
let S = *update-watchlist-wl-heur W S*;
let S = *update-vdom-wl-heur vdom S*;
RETURN S
})›
by (*auto simp*: *rewatch-heur-st-fast-def rewatch-heur-st-def state-extractors*
split: *isasat-int-splits*
intro!: *ext*)

sepref-def *rewatch-heur-st-fast-code*
is ‹(*rewatch-heur-st-fast*)›
:: ‹[*rewatch-heur-st-fast-pre*]$_a$
isasat-bounded-assn^d → *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *rewatch-heur-st-fast-alt-def rewatch-heur-st-def rewatch-heur-vdom-def*[*symmetric*] *rewatch-heur-st-fast-pre-a*
by *sepref*

definition *length-ivdom-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*length-ivdom-fast-code* = *read-vdom-wl-heur-code length-ivdom-aivdom-impl*›
global-interpretation *length-ivdom-aivdom*: *read-vdom-param-adder0* **where**
f = ‹*length-ivdom-aivdom-impl*› **and**
f' = ‹*RETURN o length-ivdom-aivdom*› **and**
x-assn = *sint64-nat-assn* **and**
P = ‹(λ*S*. *True*)›
rewrites ‹*read-vdom-wl-heur* (*RETURN o length-ivdom-aivdom*) = *RETURN o length-ivdom*› **and**
‹*read-vdom-wl-heur-code length-ivdom-aivdom-impl* = *length-ivdom-fast-code*›
apply *unfold-locales*
apply (*rule vdom-ref*)
subgoal
by (*auto simp*: *read-all-st-def length-ivdom-aivdom-def length-ivdom-def intro*!: *ext*
split: *isasat-int-splits*)
subgoal
by (*auto simp*: *length-ivdom-fast-code-def*)
done

definition *length-avdom-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*length-avdom-fast-code* = *read-vdom-wl-heur-code length-avdom-aivdom-impl*›

global-interpretation *length-avdom-aivdom*: *read-vdom-param-adder0* **where**
f = ‹*length-avdom-aivdom-impl*› **and**
f' = ‹*RETURN o length-avdom-aivdom*› **and**
x-assn = *sint64-nat-assn* **and**
P = ‹(λ*S*. *True*)›
rewrites ‹*read-vdom-wl-heur* (*RETURN o length-avdom-aivdom*) = *RETURN o length-avdom*› **and**

‹*read-vdom-wl-heur-code length-avdom-aivdom-impl = length-avdom-fast-code*›
apply *unfold-locales*
apply (*rule vdom-ref*)
subgoal
by (*auto simp*: *read-all-st-def length-avdom-aivdom-def length-avdom-def intro*!: *ext*
split: *isasat-int-splits*)
subgoal
by (*auto simp*: *length-avdom-fast-code-def*)
done

definition *length-tvdom-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*length-tvdom-fast-code = read-vdom-wl-heur-code length-tvdom-aivdom-impl*›

global-interpretation *length-tvdom-aivdom*: *read-vdom-param-adder0* **where**
f = ‹*length-tvdom-aivdom-impl*› **and**
f' = ‹*RETURN o length-tvdom-aivdom*› **and**
x-assn = *sint64-nat-assn* **and**
P = ‹(λ*S*. *True*)›
rewrites ‹*read-vdom-wl-heur* (*RETURN o length-tvdom-aivdom*) = *RETURN o length-tvdom*› **and**
‹*read-vdom-wl-heur-code length-tvdom-aivdom-impl = length-tvdom-fast-code*›
apply *unfold-locales*
apply (*rule vdom-ref*)
subgoal
by (*auto simp*: *read-all-st-def length-tvdom-aivdom-def length-tvdom-def intro*!: *ext*
split: *isasat-int-splits*)
subgoal
by (*auto simp*: *length-tvdom-fast-code-def*)
done

lemmas [*sepref-fr-rules*] = *length-ivdom-aivdom.refine*[*unfolded lambda-comp-true*]
length-avdom-aivdom.refine[*unfolded lambda-comp-true*]
length-tvdom-aivdom.refine[*unfolded lambda-comp-true*]

lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
length-ivdom-fast-code-def[*unfolded read-all-st-code-def*]
length-avdom-fast-code-def[*unfolded read-all-st-code-def*]
length-tvdom-fast-code-def[*unfolded read-all-st-code-def*]

sepref-register *length-avdom length-ivdom length-tvdom*

definition *is-learned* **where**
‹*is-learned N C* = (*arena-status N C* = *LEARNED*)›

sepref-definition *is-learned-impl*
is ‹*uncurry* (*RETURN oo is-learned*)›
:: ‹[*uncurry arena-is-valid-clause-vdom*]$_a$ *arena-fast-assn*^{k} ∗$_a$ *sint64-nat-assn*^{k} → *bool1-assn*›
unfolding *is-learned-def*
by *sepref*

definition *clause-is-learned-heur-code2* :: ‹*twl-st-wll-trail-fast2* ⇒ - ⇒ -› **where**
‹*clause-is-learned-heur-code2 N C = read-arena-wl-heur-code* (λ*Ca*. *is-learned-impl Ca C*) *N*›

lemma *clause-is-learned-heur-alt-def*: ‹*RETURN oo clause-is-learned-heur* = (λ*N C'*. *read-arena-wl-heur*
(λ*C*. (*RETURN* ∘∘ *is-learned*) *C C'*) *N*)›

by (*auto simp*: *clause-is-learned-heur-def read-all-st-def is-learned-def*
intro!: *ext split*: *isasat-int-splits*)

global-interpretation *arena-is-learned*: *read-arena-param-adder* **where**
$R = \langle(snat\text{-}rel'\ TYPE(64))\rangle$ **and**
$f' = \langle\lambda N\ C.\ (RETURN\ oo\ is\text{-}learned)\ C\ N\rangle$ **and**
$f = \langle(\lambda N\ C.\ is\text{-}learned\text{-}impl\ C\ N)\rangle$ **and**
x-assn = *bool1-assn* **and**
$P = \langle\lambda C\ N.\ arena\text{-}is\text{-}valid\text{-}clause\text{-}vdom\ N\ C\rangle$
rewrites
$\langle(\lambda N\ C.\ read\text{-}arena\text{-}wl\text{-}heur\text{-}code\ (\lambda Ca.\ is\text{-}learned\text{-}impl\ Ca\ C)\ N) = clause\text{-}is\text{-}learned\text{-}heur\text{-}code2\rangle$
and
$\langle(\lambda N\ C'.\ read\text{-}arena\text{-}wl\text{-}heur\ (\lambda C.\ (RETURN \circ\circ is\text{-}learned)\ C\ C')\ N) = RETURN\ oo\ clause\text{-}is\text{-}learned\text{-}heur\rangle$

apply *unfold-locales*
apply (*rule is-learned-impl.refine*)
subgoal by (*auto simp*: *clause-is-learned-heur-code2-def intro*!: *ext*)
subgoal by (*subst clause-is-learned-heur-alt-def*, *rule refl*)
done

definition *clause-lbd-heur-code2* :: $\langle twl\text{-}st\text{-}wll\text{-}trail\text{-}fast2 \Rightarrow \text{-}\rangle$ **where**
$\langle clause\text{-}lbd\text{-}heur\text{-}code2 = (\lambda N\ C.\ read\text{-}arena\text{-}wl\text{-}heur\text{-}code\ (\lambda Ca.\ arena\text{-}lbd\text{-}impl\ Ca\ C)\ N)\rangle$

lemma *clause-lbd-heur-alt-def*:
$\langle RETURN \circ\circ clause\text{-}lbd\text{-}heur = (\lambda N\ C'.\ read\text{-}arena\text{-}wl\text{-}heur\ (\lambda C.\ (RETURN \circ\circ arena\text{-}lbd)\ C\ C')\ N)\rangle$
by (*auto simp*: *clause-lbd-heur-def read-all-st-def arena-lbd-def split*: *isasat-int-splits intro*!: *ext*)

global-interpretation *arena-get-lbd*: *read-arena-param-adder* **where**
$R = \langle(snat\text{-}rel'\ TYPE(64))\rangle$ **and**
$f' = \langle\lambda N\ C.\ (RETURN\ oo\ arena\text{-}lbd)\ C\ N\rangle$ **and**
$f = \langle(\lambda N\ C.\ arena\text{-}lbd\text{-}impl\ C\ N)\rangle$ **and**
x-assn = *uint32-nat-assn* **and**
$P = \langle\lambda C\ N.\ get\text{-}clause\text{-}LBD\text{-}pre\ N\ C\rangle$
rewrites
$\langle(\lambda N\ C.\ read\text{-}arena\text{-}wl\text{-}heur\text{-}code\ (\lambda Ca.\ arena\text{-}lbd\text{-}impl\ Ca\ C)\ N) = clause\text{-}lbd\text{-}heur\text{-}code2\rangle$ **and**
$\langle(\lambda N\ C'.\ read\text{-}arena\text{-}wl\text{-}heur\ (\lambda C.\ (RETURN \circ\circ arena\text{-}lbd)\ C\ C')\ N) = RETURN\ oo\ clause\text{-}lbd\text{-}heur\rangle$
and
$\langle arena\text{-}get\text{-}lbd.mop = mop\text{-}arena\text{-}lbd\text{-}st\rangle$
apply *unfold-locales*
apply (*rule arena-lbd-impl.refine*)
subgoal by (*auto simp*: *clause-lbd-heur-code2-def intro*!: *ext*)
subgoal by (*subst clause-lbd-heur-alt-def*, *rule refl*)
subgoal by (*auto simp*: *mop-arena-lbd-st-def mop-arena-lbd-def read-arena-param-adder-ops.mop-def*
read-all-st-def split: *isasat-int-splits*
intro!: *ext*)
done

definition *clause-pos-heur-code2* :: $\langle twl\text{-}st\text{-}wll\text{-}trail\text{-}fast2 \Rightarrow \text{-}\rangle$ **where**
$\langle clause\text{-}pos\text{-}heur\text{-}code2 = (\lambda N\ C.\ read\text{-}arena\text{-}wl\text{-}heur\text{-}code\ (\lambda Ca.\ arena\text{-}pos\text{-}impl\ Ca\ C)\ N)\rangle$

definition *mop-arena-pos-st* :: $\langle \text{-}\rangle$ **where**
$\langle mop\text{-}arena\text{-}pos\text{-}st\ S = mop\text{-}arena\text{-}pos\ (get\text{-}clauses\text{-}wl\text{-}heur\ S)\rangle$

global-interpretation *arena-get-pos*: *read-arena-param-adder* **where**

$R = ‹(\textit{snat-rel}'\ \textit{TYPE}(64))›$ **and**
$f' = ‹\lambda N\ C.\ (\textit{RETURN oo arena-pos})\ C\ N›$ **and**
$f = ‹(\lambda N\ C.\ \textit{arena-pos-impl}\ C\ N)›$ **and**
$\textit{x-assn} = \textit{sint64-nat-assn}$ **and**
$P = ‹\lambda C\ N.\ \textit{get-saved-pos-pre}\ N\ C›$
rewrites
$‹(\lambda N\ C.\ \textit{read-arena-wl-heur-code}\ (\lambda Ca.\ \textit{arena-pos-impl}\ Ca\ C)\ N) = \textit{clause-pos-heur-code2}›$ **and**
$‹\textit{arena-get-pos.mop} = \textit{mop-arena-pos-st}›$
apply *unfold-locales*
apply (*rule arena-pos-impl.refine*)
subgoal by (*subst clause-pos-heur-code2-def*, *rule refl*)
subgoal by (*auto simp*: *mop-arena-pos-st-def mop-arena-pos-def read-arena-param-adder-ops.mop-def read-all-st-def split*: *isasat-int-splits intro*!: *ext*)
done

lemmas [*sepref-fr-rules*] = *arena-get-lbd.refine arena-get-pos.mop-refine arena-get-lbd.mop-refine*

lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
clause-lbd-heur-code2-def[*unfolded read-all-st-code-def*]
clause-is-learned-heur-code2-def[*unfolded read-all-st-code-def*]
clause-pos-heur-code2-def[*unfolded read-all-st-code-def*]
is-learned-impl-def

sepref-register *clause-lbd-heur*

sepref-register *mark-garbage-heur*
lemma *mop-mark-garbage-heur-alt-def*:
$‹\textit{mop-mark-garbage-heur}\ C\ i = (\lambda S_0.\ \textit{do}\ \{$
$\textit{ASSERT}(\textit{mark-garbage-pre}\ (\textit{get-clauses-wl-heur}\ S_0,\ C) \wedge \textit{clss-size-lcount}\ (\textit{get-learned-count}\ S_0) \geq 1 \wedge i < \textit{length}\ (\textit{get-avdom}\ S_0))$;
$\textit{let}\ (N,\ S) = \textit{extract-arena-wl-heur}\ S_0$;
$\textit{ASSERT}\ (N = \textit{get-clauses-wl-heur}\ S_0)$;
$\textit{let}\ N' = \textit{extra-information-mark-to-delete}\ N\ C$;
$\textit{let}\ S = \textit{update-arena-wl-heur}\ N'\ S$;
$\textit{let}\ (\textit{lcount},\ S) = \textit{extract-lcount-wl-heur}\ S$;
$\textit{ASSERT}\ (\textit{lcount} = \textit{get-learned-count}\ S_0)$;
$\textit{let}\ \textit{lcount} = \textit{clss-size-decr-lcount}\ (\textit{lcount})$;
$\textit{let}\ (\textit{vdom},\ S) = \textit{extract-vdom-wl-heur}\ S$;
$\textit{ASSERT}\ (\textit{vdom} = \textit{get-aivdom}\ S_0)$;
$\textit{let}\ S = \textit{update-vdom-wl-heur}\ (\textit{remove-inactive-aivdom}\ i\ \textit{vdom})\ S$;
$\textit{RETURN}\ (\textit{update-lcount-wl-heur}\ \textit{lcount}\ S)$
$\})›$
unfolding *mop-mark-garbage-heur-def mark-garbage-heur-def*
by (*auto intro*!: *ext simp*: *state-extractors split*: *isasat-int-splits*)

sepref-def *mop-mark-garbage-heur-impl*
is $‹\textit{uncurry2 mop-mark-garbage-heur}›$
:: $‹[\lambda((C,\ i),\ S).\ \textit{length}\ (\textit{get-clauses-wl-heur}\ S) \leq \textit{snat64-max}]_a$
$\textit{sint64-nat-assn}^k *_a \textit{sint64-nat-assn}^k *_a \textit{isasat-bounded-assn}^d \to \textit{isasat-bounded-assn}›$
supply [[*goals-limit=1*]]
unfolding *mop-mark-garbage-heur-alt-def clause-not-marked-to-delete-heur-pre-def*

by *sepref*

lemma *mark-garbage-heur-alt-def*: ‹*RETURN ooo mark-garbage-heur* =
(λ*C i* S_0. *do* {
let (*N*, *S*) = *extract-arena-wl-heur* S_0;
ASSERT (*N* = *get-clauses-wl-heur* S_0);
let N' = *extra-information-mark-to-delete N C*;
let S = *update-arena-wl-heur N' S*;
let (*lcount*, *S*) = *extract-lcount-wl-heur S*;
ASSERT (*lcount* = *get-learned-count* S_0);
let lcount = *clss-size-decr-lcount* (*lcount*);
let (*vdom*, *S*) = *extract-vdom-wl-heur S*;
ASSERT (*vdom* = *get-aivdom* S_0);
let S = *update-vdom-wl-heur* (*remove-inactive-aivdom i vdom*) *S*;
RETURN (*update-lcount-wl-heur lcount S*)})›
unfolding *mop-mark-garbage-heur-def mark-garbage-heur-def*
by (*auto intro*!: *ext simp*: *state-extractors*
split: *isasat-int-splits*)

sepref-def *mark-garbage-heur-code2*
is ‹*uncurry2* (*RETURN ooo mark-garbage-heur*)›
:: ‹[λ((*C*, *i*), *S*). *mark-garbage-pre* (*get-clauses-wl-heur S*, *C*) ∧ *i* < *length-avdom S* ∧
clss-size-lcount (*get-learned-count S*) ≥ 1]$_a$
sint64-nat-assn^k *$*_a$ *sint64-nat-assn*^{k} *$*_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
supply [[*goals-limit* = 1]]
unfolding *mark-garbage-heur-alt-def length-avdom-def*
by *sepref*

lemma *mop-mark-garbage-heur3-alt-def*:
‹*mop-mark-garbage-heur3 C i* = (λS_0. *do* {
ASSERT(*mark-garbage-pre* (*get-clauses-wl-heur* S_0, *C*) ∧ *clss-size-lcount* (*get-learned-count* S_0) ≥ 1
∧ *i* < *length* (*get-tvdom* S_0));
let (*N*, *S*) = *extract-arena-wl-heur* S_0;
ASSERT (*N* = *get-clauses-wl-heur* S_0);
let N' = *extra-information-mark-to-delete N C*;
let S = *update-arena-wl-heur N' S*;
let (*vdom*, *S*) = *extract-vdom-wl-heur S*;
ASSERT (*vdom* = *get-aivdom* S_0);
let vdom = *remove-inactive-aivdom-tvdom i vdom*;
let S = *update-vdom-wl-heur vdom S*;
let (*lcount*, *S*) = *extract-lcount-wl-heur S*;
ASSERT (*lcount* = *get-learned-count* S_0);
let lcount = *clss-size-decr-lcount lcount*;
let S = *update-lcount-wl-heur lcount S*;
RETURN S
})›
unfolding *mop-mark-garbage-heur3-def mark-garbage-heur3-def*
by (*auto intro*!: *ext simp*: *state-extractors*
split: *isasat-int-splits*)

sepref-def *mop-mark-garbage-heur3-impl*
is ‹*uncurry2 mop-mark-garbage-heur3*›
:: ‹[λ((*C*, *i*), *S*). *length* (*get-clauses-wl-heur S*) ≤ *snat64-max*]$_a$
sint64-nat-assn^k *$*_a$ *sint64-nat-assn*^{k} *$*_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *mop-mark-garbage-heur3-alt-def*

by *sepref*

lemma *delete-index-vdom-heur-alt-def*: ‹*RETURN oo delete-index-vdom-heur* = (λ*i* S_0. *do* {
let (*vdom*, *S*) = *extract-vdom-wl-heur* S_0;
ASSERT (*vdom* = *get-aivdom* S_0);
let vdom = *remove-inactive-aivdom-tvdom i vdom*;
RETURN (*update-vdom-wl-heur vdom S*)
})›
by (*auto simp*: *state-extractors delete-index-vdom-heur-def*
intro!: *ext split*: *isasat-int-splits*)

sepref-def *delete-index-vdom-heur-fast-code2*
is ‹*uncurry* (*RETURN oo delete-index-vdom-heur*)›
:: ‹[λ(*i*, *S*). *i* < *length-tvdom S*]$_a$
sint64-nat-assn^k *$*_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
supply [[*goals-limit* = *1*]]
supply [*simp*] = *length-tvdom-def*
unfolding *delete-index-vdom-heur-alt-def*
by *sepref*

lemma *access-length-heur-alt-def*:
‹*RETURN oo access-length-heur* = (λ*N C'*. *read-arena-wl-heur* (λ*N*. *RETURN* (*arena-length N C'*))
N)›
by (*auto intro*!: *ext simp*: *read-all-st-def access-length-heur-def*
split: *isasat-int-splits*)

definition *access-length-heur-fast-code2* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*access-length-heur-fast-code2* = (λ*N C*. *read-arena-wl-heur-code* (λ*N*. *arena-length-impl N C*) *N*)›

global-interpretation *arena-length*: *read-arena-param-adder* **where**
R = ‹*snat-rel'* *TYPE*(*64*)› **and**
f' = ‹λ*C N*. *RETURN* (*arena-length N C*)› **and**
f = ‹(λ*C' N*. *arena-length-impl N C'*)› **and**
x-assn = ‹*snat-assn'* *TYPE*(*64*)› **and**
P = ‹(λ*C S*. *arena-is-valid-clause-idx S C*)›
rewrites
‹(λ*N C'*. *read-arena-wl-heur* (λ*N*. *RETURN* (*arena-length N C'*)) *N*) = *RETURN oo access-length-heur*›
and
‹(λ*N C*. *read-arena-wl-heur-code* (λ*N*. *arena-length-impl N C*) *N*) = *access-length-heur-fast-code2*›
apply *unfold-locales*
apply (*rule arena-length-impl.refine*[*unfolded comp-def*])
subgoal by (*auto simp*: *access-length-heur-alt-def*)
subgoal by (*auto simp*: *access-length-heur-fast-code2-def*)
done

lemmas [*sepref-fr-rules*] = *arena-length.refine*

lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
access-length-heur-fast-code2-def[*unfolded read-all-st-code-def*]

lemma *get-slow-ema-heur-alt-def*:
‹*RETURN o get-slow-ema-heur* = *read-heur-wl-heur* (*RETURN o slow-ema-of*)› **and**
get-fast-ema-heur-alt-def:
‹*RETURN o get-fast-ema-heur* = *read-heur-wl-heur* (*RETURN o fast-ema-of*)›
by (*auto simp*: *read-all-st-def intro*!: *ext split*: *isasat-int-splits*)

definition *get-slow-ema-heur-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*get-slow-ema-heur-fast-code* = *read-heur-wl-heur-code slow-ema-of-stats-impl*›

global-interpretation *slow-ema*: *read-heur-param-adder0* **where**
f' = ‹*RETURN o slow-ema-of*› **and**
f = *slow-ema-of-stats-impl* **and**
x-assn = ‹*ema-assn*› **and**
P = ‹(λ-. *True*)›
rewrites
‹*read-heur-wl-heur* (*RETURN o slow-ema-of*) = *RETURN o get-slow-ema-heur*› **and**
‹*read-heur-wl-heur-code slow-ema-of-stats-impl* = *get-slow-ema-heur-fast-code*›
apply *unfold-locales*
apply (*rule heur-refine*)
subgoal by (*auto simp*: *get-slow-ema-heur-alt-def*)
subgoal by (*auto simp*: *get-slow-ema-heur-fast-code-def*)
done

definition *get-fast-ema-heur-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*get-fast-ema-heur-fast-code* = *read-heur-wl-heur-code fast-ema-of-stats-impl*›

global-interpretation *fast-ema*: *read-heur-param-adder0* **where**
f' = ‹*RETURN o fast-ema-of*› **and**
f = *fast-ema-of-stats-impl* **and**
x-assn = ‹*ema-assn*› **and**
P = ‹(λ-. *True*)›
rewrites
‹*read-heur-wl-heur* (*RETURN o fast-ema-of*) = *RETURN o get-fast-ema-heur*› **and**
‹*read-heur-wl-heur-code fast-ema-of-stats-impl* = *get-fast-ema-heur-fast-code*›
apply *unfold-locales*
apply (*rule heur-refine*)
subgoal by (*auto simp*: *get-fast-ema-heur-alt-def*)
subgoal by (*auto simp*: *get-fast-ema-heur-fast-code-def*)
done

thm *get-conflict-count-since-last-restart-heur.simps*
find-theorems *get-conflict-count-since-last-restart RETURN*

lemma *get-conflict-count-since-last-restart-heur-alt-def*:
‹*RETURN o get-conflict-count-since-last-restart-heur* =
read-heur-wl-heur (*RETURN* ∘ *get-conflict-count-since-last-restart*)›
by (*auto simp*: *read-all-st-def intro*!: *ext split*: *isasat-int-splits*)

definition *get-conflict-count-since-last-restart-heur-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*get-conflict-count-since-last-restart-heur-fast-code* = *read-heur-wl-heur-code get-conflict-count-since-last-restart-stats-im*

global-interpretation *get-conflict-count-since-last-restart*: *read-heur-param-adder0* **where**
f' = ‹*RETURN o get-conflict-count-since-last-restart*› **and**
f = *get-conflict-count-since-last-restart-stats-impl* **and**
x-assn = ‹*word64-assn*› **and**
P = ‹(λ-. *True*)›
rewrites
‹*read-heur-wl-heur* (*RETURN o get-conflict-count-since-last-restart*) = *RETURN o get-conflict-count-since-last-restart*
and
‹*read-heur-wl-heur-code get-conflict-count-since-last-restart-stats-impl* = *get-conflict-count-since-last-restart-heur-fast-c*
apply *unfold-locales*

apply (*rule heur-refine*)
subgoal by (*auto simp*: *get-conflict-count-since-last-restart-heur-alt-def*)
subgoal by (*auto simp*: *get-conflict-count-since-last-restart-heur-fast-code-def*)
done

lemma *id-lcount-assn*: ‹(*Mreturn*, *RETURN*) ∈ (*lcount-assn*)^k →$_a$ *lcount-assn*›
unfolding *lcount-assn-def*
by *sepref-to-hoare vcg*

lemma *get-learned-count-alt-def*: ‹*RETURN o get-learned-count* = *read-lcount-wl-heur RETURN*›
by (*auto simp*: *read-all-st-def intro*!: *ext split*: *isasat-int-splits*)

definition *get-learned-count-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*get-learned-count-fast-code* = *read-lcount-wl-heur-code Mreturn*›

global-interpretation *get-lcount*: *read-lcount-param-adder0* **where**
f' = ‹*RETURN*› **and**
f = ‹*Mreturn*› **and**
x-assn = ‹*lcount-assn*› **and**
P = ‹(λ-. *True*)›
rewrites
‹*read-lcount-wl-heur* (*RETURN*) = *RETURN o get-learned-count*› **and**
‹*read-lcount-wl-heur-code Mreturn* = *get-learned-count-fast-code*›
apply *unfold-locales*
apply (*rule id-lcount-assn*)
subgoal by (*auto simp*: *get-learned-count-alt-def*)
subgoal by (*auto simp*: *get-learned-count-fast-code-def*)
done

definition *get-learned-count-number-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*get-learned-count-number-fast-code* = *read-lcount-wl-heur-code clss-size-lcount-fast-code*›

global-interpretation *get-learned-count-number*: *read-lcount-param-adder0* **where**
f' = ‹*RETURN o clss-size-lcount*› **and**
f = ‹*clss-size-lcount-fast-code*› **and**
x-assn = ‹*uint64-nat-assn*› **and**
P = ‹(λ-. *True*)›
rewrites
‹*read-lcount-wl-heur* (*RETURN o clss-size-lcount*) = *RETURN o get-learned-count-number*› **and**
‹*read-lcount-wl-heur-code clss-size-lcount-fast-code* = *get-learned-count-number-fast-code*›
apply *unfold-locales*
apply (*rule clss-size-lcount-fast-code.refine*)
subgoal by (*auto simp*: *read-all-st-def split*: *isasat-int-splits intro*!: *ext*)
subgoal by (*auto simp*: *get-learned-count-number-fast-code-def*)
done

definition *get-learned-count-number'* :: ‹*isasat* ⇒ *nat*› **where**
‹*get-learned-count-number' S* ≡ *get-learned-count-number S*›

lemma [*def-pat-rules*]: ‹*get-learned-count-number*$*S*≡*get-learned-count-number'*$*S*›
by (*auto simp*: *get-learned-count-number'-def*)

lemmas [*sepref-fr-rules*] =
slow-ema.refine[*unfolded lambda-comp-true*] *fast-ema.refine*[*unfolded lambda-comp-true*]
get-conflict-count-since-last-restart.refine[*unfolded lambda-comp-true*]

get-lcount.refine[*unfolded lambda-comp-true*]
get-learned-count-number.refine[*unfolded lambda-comp-true get-learned-count-number'-def*[*symmetric*]]

lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
get-slow-ema-heur-fast-code-def[*unfolded read-all-st-code-def*]
get-fast-ema-heur-fast-code-def[*unfolded read-all-st-code-def*]
get-conflict-count-since-last-restart-heur-fast-code-def[*unfolded read-all-st-code-def*]
get-learned-count-fast-code-def[*unfolded read-all-st-code-def*]
get-learned-count-number-fast-code-def[*unfolded read-all-st-code-def*]

sepref-def *learned-clss-count-fast-code*
is ‹*RETURN o learned-clss-count*›
:: ‹[λ*S*. *learned-clss-count S* ≤ *unat64-max*]$_a$ *isasat-bounded-assn*^{k} → *uint64-nat-assn*›
unfolding *clss-size-allcount-alt-def learned-clss-count-def*
by *sepref*

definition *marked-as-used-st-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*marked-as-used-st-fast-code* = (λ*N C*. *read-arena-wl-heur-code* (λ*N*. *marked-as-used-impl N C*) *N*)›
global-interpretation *marked-used*: *read-arena-param-adder* **where**
R = ‹*snat-rel'* *TYPE*(*64*)› **and**
f' = ‹λ*C N*. *RETURN* (*marked-as-used N C*)› **and**
f = ‹(λ*C' N*. *marked-as-used-impl N C'*)› **and**
x-assn = ‹*unat-assn'* *TYPE*(*2*)› **and**
P = ‹(λ*C S*. *marked-as-used-pre S C*)›
rewrites
‹(λ*N C'*. *read-arena-wl-heur* (λ*N*. *RETURN* (*marked-as-used N C'*)) *N*) = *RETURN oo marked-as-used-st*›
and
‹(λ*N C*. *read-arena-wl-heur-code* (λ*N*. *marked-as-used-impl N C*) *N*) = *marked-as-used-st-fast-code*›
apply *unfold-locales*
apply (*rule marked-as-used-impl.refine*[*unfolded comp-def*])
subgoal by (*auto simp*: *marked-as-used-st-def read-all-st-def intro*!: *ext split*: *isasat-int-splits*)
subgoal by (*auto simp*: *marked-as-used-st-fast-code-def*)
done

lemmas [*sepref-fr-rules*] = *marked-used.refine*

lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
marked-as-used-st-fast-code-def[*unfolded read-all-st-code-def*]

lemma *mop-marked-as-used-st-alt-def*: ‹*mop-marked-as-used-st* = *marked-used.mop*›
by (*auto intro*!: *ext split*: *isasat-int-splits simp*: *mop-marked-as-used-st-def marked-used.mop-def*
mop-marked-as-used-def read-all-st-def)

lemmas [*sepref-fr-rules*] =
marked-used.mop-refine[*unfolded mop-marked-as-used-st-alt-def*[*symmetric*]]

sepref-register *get-the-propagation-reason-heur delete-index-vdom-heur access-length-heur marked-as-used-st*

experiment
begin

export-llvm *polarity-st-heur-pol-fast count-decided-st-heur-fast-code get-conflict-wl-is-None-fast-code*
clause-not-marked-to-delete-heur-code access-lit-in-clauses-heur-fast-code length-ivdom-fast-code
length-avdom-fast-code length-tvdom-fast-code
clause-is-learned-heur-code2 clause-lbd-heur-code2 mop-mark-garbage-heur-impl mark-garbage-heur-code2

mop-mark-garbage-heur3-impl delete-index-vdom-heur-fast-code2 access-length-heur-fast-code2 get-fast-ema-heur-fast-code get-slow-ema-heur-fast-code get-conflict-count-since-last-restart-heur-fast-code get-learned-count-fast-code

end
end
theory *IsaSAT-VMTF*
imports *Watched-Literals.WB-Sort IsaSAT-Setup Weidenbach-Book-Base.Explorer*
begin

Chapter 13

VMTF Decision Heuristic

13.1 Code generation for the VMTF decision heuristic and the trail

type-synonym (**in** −) *isa-vmtf-remove-int* = ‹*vmtf* × (*nat list* × *bool list*)›

definition *update-next-search* :: ‹*nat option* ⇒ *vmtf* ⇒ *vmtf*› **where**
‹*update-next-search L* = (λ(*ns*, *m*, *fst-As*, *lst-As*, *next-search*).
((*ns*, *m*, *fst-As*, *lst-As*, *L*)))›

definition *vmtf-enqueue-pre* **where**
‹*vmtf-enqueue-pre* =
(λ((*M*, *L*),(*ns*,*m*,*fst-As*,*lst-As*, *next-search*)). *L* < *length ns* ∧
(*fst-As* ≠ *None* ⟶ *the fst-As* < *length ns*) ∧
(*fst-As* ≠ *None* ⟶ *lst-As* ≠ *None*) ∧
m+1 ≤ *unat64-max*)›

definition *isa-vmtf-enqueue* :: ‹*trail-pol* ⇒ *nat* ⇒ *vmtf-option-fst-As* ⇒ *vmtf nres*› **where**
‹*isa-vmtf-enqueue* = (λ*M L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*). *do* {
ASSERT(*defined-atm-pol-pre M L*);
de ← *RETURN* (*defined-atm-pol M L*);
case fst-As of
None ⇒*RETURN* ((*ns*[*L* := *VMTF-Node m fst-As None*], *m*+1, *L*, *L*,
(*if de then None else Some L*)))
| *Some fst-As* ⇒ *do* {
let fst-As′ = *VMTF-Node* (*stamp* (*ns*!*fst-As*)) (*Some L*) (*get-next* (*ns*!*fst-As*));
RETURN (*ns*[*L* := *VMTF-Node* (*m*+1) *None* (*Some fst-As*), *fst-As* := *fst-As*′],
m+1, *L*, *the lst-As*, (*if de then next-search else Some L*))
}})›

lemma *vmtf-enqueue-alt-def*:
‹*RETURN ooo vmtf-enqueue* = (λ*M L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*). *do* {
let de = *defined-lit M* (*Pos L*);
case fst-As of
None ⇒ *RETURN* (*ns*[*L* := *VMTF-Node m fst-As None*], *m*+1, *L*, *L*,
(*if de then None else Some L*))
| *Some fst-As* ⇒
let fst-As′ = *VMTF-Node* (*stamp* (*ns*!*fst-As*)) (*Some L*) (*get-next* (*ns*!*fst-As*)) *in*
RETURN (*ns*[*L* := *VMTF-Node* (*m*+1) *None* (*Some fst-As*), *fst-As* := *fst-As*′],
m+1, *L*, *the lst-As*, (*if de then next-search else Some L*))})›
unfolding *vmtf-enqueue-def Let-def*

by (*auto intro*!: *ext split*: *option.splits*)

lemma *isa-vmtf-enqueue*:
‹(*uncurry2 isa-vmtf-enqueue*, *uncurry2* (*RETURN ooo vmtf-enqueue*)) ∈
[λ((*M*, *L*), -). *L* ∈# $\mathcal{A}$]$_f$ (*trail-pol* $\mathcal{A}$) ×$_f$ *nat-rel* ×$_f$ *Id* → ⟨*Id*⟩*nres-rel*›
proof −
have *defined-atm-pol*: ‹(*defined-atm-pol x1g x2f*, *defined-lit x1a* (*Pos x2*)) ∈ *Id*›
if
‹*case y of* (*x*, *xa*) ⇒ (*case x of* (*M*, *L*) ⇒ λ-. *L* ∈# $\mathcal{A}$) *xa*› **and**
‹(*x*, *y*) ∈ *trail-pol* $\mathcal{A}$ ×$_f$ *nat-rel* ×$_f$ *Id*› **and** ‹*x1* = (*x1a*, *x2*)› **and**
‹*x2d* = (*x1e*, *x2e*)› **and**
‹*x2c* = (*x1d*, *x2d*)› **and**
‹*x2b* = (*x1c*, *x2c*)› **and**
‹*x2a* = (*x1b*, *x2b*)› **and**
‹*y* = (*x1*, *x2a*)› **and**
‹*x1f* = (*x1g*, *x2f*)› **and**
‹*x2j* = (*x1k*, *x2k*)› **and**
‹*x2i* = (*x1j*, *x2j*)› **and**
‹*x2h* = (*x1i*, *x2i*)› **and**
‹*x2g* = (*x1h*, *x2h*)› **and**
‹*x* = (*x1f*, *x2g*)›
for *x y x1 x1a x2 x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x1g x2f x2g x1h x2h*
x1i x2i x1j x2j x1k x2k
proof −
have [*simp*]: ‹*defined-lit x1a* (*Pos x2*) ⟷ *defined-atm x1a x2*›
using *that* **by** (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *trail-pol-def defined-atm-def*)

show *?thesis*
using *undefined-atm-code*[*THEN fref-to-Down*, *unfolded uncurry-def*, *of* $\mathcal{A}$ ‹(*x1a*, *x2*)› ‹(*x1g*, *x2f*)›]
that **by** (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *RETURN-def*)
qed

show *?thesis*
unfolding *isa-vmtf-enqueue-def vmtf-enqueue-alt-def uncurry-def*
apply (*intro frefI nres-relI*)
apply (*refine-rcg*)
subgoal by (*rule defined-atm-pol-pre*) *auto*
apply (*rule defined-atm-pol*; *assumption*)
apply (*rule same-in-Id-option-rel*)
subgoal for *x y x1 x1a x2 x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x1g x2f x2g x1h x2h*
x1i x2i x1j x2j x1k x2k
by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

definition *partition-vmtf-nth* :: ‹*nat-vmtf-node list* ⇒ *nat* ⇒ *nat* ⇒ *nat list* ⇒ (*nat list* × *nat*) *nres*›
where
‹*partition-vmtf-nth ns* = *partition-main* (≤) (λ*n*. *stamp* (*ns* ! *n*))›

definition *partition-between-ref-vmtf* :: ‹*nat-vmtf-node list* ⇒ *nat* ⇒ *nat* ⇒ *nat list* ⇒ (*nat list* × *nat*) *nres*› **where**
‹*partition-between-ref-vmtf ns* = *partition-between-ref* (≤) (λ*n*. *stamp* (*ns* ! *n*))›

definition *quicksort-vmtf-nth* :: ‹*nat-vmtf-node list* × *'c* ⇒ *nat list* ⇒ *nat list nres*› **where**

‹*quicksort-vmtf-nth* = (λ(*ns*, -). *full-quicksort-ref* (≤) (λ*n*. *stamp* (*ns* ! *n*)))›

definition *quicksort-vmtf-nth-ref*:: ‹*nat-vmtf-node list* ⇒ *nat* ⇒ *nat* ⇒ *nat list* ⇒ *nat list nres*› **where**
‹*quicksort-vmtf-nth-ref ns a b c* =
quicksort-ref (≤) (λ*n*. *stamp* (*ns* ! *n*)) (*a*, *b*, *c*)›

lemma (**in** −) *partition-vmtf-nth-code-helper*:
assumes ‹∀ *x*∈*set ba*. *x* < *length a*› **and**
‹*b* < *length ba*› **and**
mset: ‹*mset ba* = *mset a2'*› **and**
‹*a1'* < *length a2'*›
shows ‹*a2'* ! *b* < *length a*›
using *nth-mem*[*of b a2'*] *mset-eq-setD*[*OF mset*] *mset-eq-length*[*OF mset*] *assms*
by (*auto simp del*: *nth-mem*)

lemma *partition-vmtf-nth-code-helper3*:
‹∀ *x*∈*set b*. *x* < *length a* ⟹
x'e < *length a2'* ⟹
mset a2' = *mset b* ⟹
a2' ! *x'e* < *length a*›
using *mset-eq-setD nth-mem* **by** *blast*

definition (**in** −) *isa-vmtf-en-dequeue* :: ‹*trail-pol* ⇒ *nat* ⇒ *vmtf* ⇒ *vmtf nres*› **where**
‹*isa-vmtf-en-dequeue* = (λ*M L vm*. *isa-vmtf-enqueue M L* (*vmtf-dequeue L vm*))›

lemma *isa-vmtf-en-dequeue*:
‹(*uncurry2 isa-vmtf-en-dequeue*, *uncurry2* (*RETURN ooo vmtf-en-dequeue*)) ∈
[λ((*M*, *L*), -). *L* ∈# $\mathcal{A}$]$_f$ (*trail-pol* $\mathcal{A}$) ×$_f$ *nat-rel* ×$_f$ *Id* → ⟨*Id*⟩*nres-rel*›
unfolding *isa-vmtf-en-dequeue-def vmtf-en-dequeue-def uncurry-def*
apply (*intro frefI nres-relI*)
apply *clarify*
subgoal for *a aa ab ac ad b zeroed ba ae af ag ah bb ai bc aj ak al am bd*
by (*rule order.trans*,
rule isa-vmtf-enqueue[*of* $\mathcal{A}$, *THEN fref-to-Down-curry2*,
of ai bc ‹*vmtf-dequeue bc* (*aj*, *ak*, *al*, *am*, *bd*)›])
auto
done

definition *isa-vmtf-en-dequeue-pre* :: ‹(*trail-pol* × *nat*) × *vmtf* ⇒ *bool*› **where**
‹*isa-vmtf-en-dequeue-pre* = (λ((*M*, *L*),(*ns*,*m*,*fst-As*, *lst-As*, *next-search*)).
L < *length ns* ∧ *vmtf-dequeue-pre* (*L*, *ns*) ∧
fst-As < *length ns* ∧ (*get-next* (*ns* ! *fst-As*) ≠ *None* ⟶ *get-prev* (*ns* ! *lst-As*) ≠ *None*) ∧
(*get-next* (*ns* ! *fst-As*) = *None* ⟶ *fst-As* = *lst-As*) ∧
m+1 ≤ *unat64-max*)›

lemma *isa-vmtf-en-dequeue-preD*:
assumes ‹*isa-vmtf-en-dequeue-pre* ((*M*, *ah*), *a*, *aa*, *ab*, *ac*, *b*)›
shows ‹*ah* < *length a*› **and** ‹*vmtf-dequeue-pre* (*ah*, *a*)›
using *assms* **by** (*auto simp*: *isa-vmtf-en-dequeue-pre-def*)

lemma *isa-vmtf-en-dequeue-pre-vmtf-enqueue-pre*:
‹*isa-vmtf-en-dequeue-pre* ((*M*, *L*), *a*, *st*, *fst-As*, *lst-As*, *next-search*) ⟹
vmtf-enqueue-pre ((*M*, *L*), *vmtf-dequeue L* (*a*, *st*, *fst-As*, *lst-As*, *next-search*))›

unfolding *vmtf-enqueue-pre-def*
apply *clarify*
apply (*intro conjI*)
subgoal
by (*auto simp*: *vmtf-dequeue-pre-def vmtf-enqueue-pre-def vmtf-dequeue-def ns-vmtf-dequeue-def Let-def isa-vmtf-en-dequeue-pre-def split*: *option.splits*)[]
subgoal
by (*auto simp*: *vmtf-dequeue-pre-def vmtf-enqueue-pre-def vmtf-dequeue-def isa-vmtf-en-dequeue-pre-def split*: *option.splits if-splits*)[]
subgoal
by (*auto simp*: *vmtf-dequeue-pre-def vmtf-enqueue-pre-def vmtf-dequeue-def Let-def isa-vmtf-en-dequeue-pre-def split*: *option.splits if-splits*)[]
subgoal
by (*auto simp*: *vmtf-dequeue-pre-def vmtf-enqueue-pre-def vmtf-dequeue-def Let-def isa-vmtf-en-dequeue-pre-def split*: *option.splits if-splits*)[]
done

lemma *insert-sort-reorder-list*:
assumes *trans*: ‹$\bigwedge$ *x y z*. ⟦*R* (*h x*) (*h y*); *R* (*h y*) (*h z*)⟧ ⟹ *R* (*h x*) (*h z*)› **and** *lin*: ‹$\bigwedge$*x y*. *R* (*h x*) (*h y*) ∨ *R* (*h y*) (*h x*)›
shows ‹(*full-quicksort-ref R h*, *reorder-list vm*) ∈ ⟨*Id*⟩*list-rel* →$_f$ ⟨*Id*⟩ *nres-rel*›
proof −
show *?thesis*
apply (*intro frefI nres-relI*)
apply (*rule full-quicksort-ref-full-quicksort*[*THEN fref-to-Down*, *THEN order-trans*])
using *assms* **apply** *fast*
using *assms* **apply** *fast*
apply *fast*
apply *assumption*
using *assms*
apply (*auto 5 5 simp*: *reorder-list-def intro*!: *full-quicksort-correct*[*THEN order-trans*])
done
qed

lemma *quicksort-vmtf-nth-reorder*:
‹(*uncurry quicksort-vmtf-nth*, *uncurry reorder-list*) ∈
Id ×$_r$ ⟨*Id*⟩*list-rel* →$_f$ ⟨*Id*⟩ *nres-rel*›
apply (*intro frefI nres-relI*)
subgoal for *x y*
using *insert-sort-reorder-list*[*unfolded fref-def nres-rel-def*, *of* ‹(≤)› ‹(λ*n*. *stamp* (*fst* (*fst y*) ! *n*) :: *nat*)› ‹*fst y*›]
by (*cases x*, *cases y*)
(*fastforce simp*: *quicksort-vmtf-nth-def uncurry-def fref-def*)
done

lemma *atoms-hash-del-op-set-delete*:
‹(*uncurry* (*RETURN oo atoms-hash-del*),
uncurry (*RETURN oo Set.remove*)) ∈
nat-rel ×$_r$ *atoms-hash-rel* $\mathcal{A}$ →$_f$ ⟨*atoms-hash-rel* $\mathcal{A}$⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*force simp*: *atoms-hash-del-def atoms-hash-rel-def*)

definition *current-stamp* **where**
‹*current-stamp vm* = *fst* (*snd vm*)›

lemma *current-stamp-alt-def*:
‹*current-stamp* = (λ(-, *m*, -). *m*)›
by (*auto simp*: *current-stamp-def intro*!: *ext*)

lemma *vmtf-rescale-alt-def*:
‹*vmtf-rescale* = (λ(*ns*, *m*, *fst-As*, *lst-As* :: *nat*, *next-search*). *do* {
(*ns*, *m*, -) ← *WHILE*$_T$$^{\lambda\text{-}.\ True}$
(λ(*ns*, *n*, *lst-As*). *lst-As* ≠*None*)
(λ(*ns*, *n*, *a*). *do* {
ASSERT(*a* ≠ *None*);
ASSERT(*n*+*1* ≤ *unat32-max*);
ASSERT(*the a* < *length ns*);
let m = *the a*;
let c = *ns* ! *m*;
let nc = *get-next c*;
let pc = *get-prev c*;
RETURN (*ns*[*m* := *VMTF-Node n pc nc*], *n* + *1*, *pc*)
})
(*ns*, *0*, *Some lst-As*);
RETURN ((*ns*, *m*, *fst-As*, *lst-As*, *next-search*))
})›
unfolding *update-stamp.simps Let-def vmtf-rescale-def* **by** *auto*

definition *vmtf-reorder-list-raw* **where**
‹*vmtf-reorder-list-raw* = (λ*vm to-remove*. *do* {
ASSERT(∀ *x*∈*set to-remove*. *x* < *length vm*);
reorder-list vm to-remove
})›

definition *vmtf-reorder-list* :: ‹*vmtf* ⇒ *nat list* ⇒ *nat list nres*› **where**
‹*vmtf-reorder-list* = (λ(*vm*, -) *to-remove*. *do* {
vmtf-reorder-list-raw vm to-remove
})›

definition *isa-vmtf-flush-int* :: ‹*trail-pol* ⇒ *vmtf* ⇒ - ⇒ (*vmtf* × -) *nres*› **where**
‹*isa-vmtf-flush-int* = (λ*M vm* (*to-remove*, *h*). *do* {
ASSERT(∀ *x*∈*set to-remove*. *x* < *length* (*fst vm*));
ASSERT(*length to-remove* ≤ *unat32-max*);
to-remove′ ← *vmtf-reorder-list vm to-remove*;
ASSERT(*length to-remove*′ ≤ *unat32-max*);
vm ← (*if length to-remove*′ ≥ *unat64-max* − *fst* (*snd vm*)
then vmtf-rescale vm else RETURN vm);
ASSERT(*length to-remove*′ + *fst* (*snd vm*) ≤ *unat64-max*);
(-, *vm*, *h*) ← *WHILE*$_T$λ(*i*, *vm*′, *h*). *i* ≤ *length to-remove*′ ∧ *fst* (*snd vm*′) = *i* + *fst* (*snd vm*) ∧ (*i* < *length to-remove*
(λ(*i*, *vm*, *h*). *i* < *length to-remove*′)
(λ(*i*, *vm*, *h*). *do* {
ASSERT(*i* < *length to-remove*′);
ASSERT(*isa-vmtf-en-dequeue-pre* ((*M*, *to-remove*′!*i*), *vm*));
vm ← *isa-vmtf-en-dequeue M* (*to-remove*′!*i*) *vm*;
ASSERT(*atoms-hash-del-pre* (*to-remove*′!*i*) *h*);
RETURN (*i*+*1*, *vm*, *atoms-hash-del* (*to-remove*′!*i*) *h*)})
(*0*, *vm*, *h*);
RETURN (*vm*, (*emptied-list to-remove*′, *h*))
})›

lemma *isa-vmtf-flush-int*:
‹(*uncurry2 isa-vmtf-flush-int*, *uncurry2* (*vmtf-flush-int* $\mathcal{A}$)) ∈ *trail-pol* ($\mathcal{A}$::*nat multiset*) ×$_f$ *Id* ×$_f$ *Id*
→$_f$ ⟨*Id*×$_f$ *Id*⟩*nres-rel*›
proof −
have *vmtf-flush-int-alt-def*:
‹*vmtf-flush-int* $\mathcal{A}_{in}$ = (λ*M vm* (*to-remove*, *h*). *do* {
ASSERT(∀ *x*∈*set to-remove*. *x* < *length* (*fst vm*));
ASSERT(*length to-remove* ≤ *unat32-max*);
to-remove′ ← *reorder-list vm to-remove*;
ASSERT(*length to-remove*′ ≤ *unat32-max*);
vm ← (*if length to-remove*′ + *fst* (*snd vm*) ≥ *unat64-max*
then vmtf-rescale vm else RETURN vm);
ASSERT(*length to-remove*′ + *fst* (*snd vm*) ≤ *unat64-max*);
(-, *vm*, *h*) ← *WHILE*$_T$$^{\lambda(i,\ vm',\ h).\ i \le length\ to\text{-}remove' \wedge fst\ (snd\ vm') = i + fst\ (snd\ vm) \wedge \quad (i < length\ to\text{-}remove' -}$
(λ(*i*, *vm*, *h*). *i* < *length to-remove*′)
(λ(*i*, *vm*, *h*). *do* {
ASSERT(*i* < *length to-remove*′);
ASSERT(*to-remove*′!*i* ∈# $\mathcal{A}_{in}$);
ASSERT(*atoms-hash-del-pre* (*to-remove*′!*i*) *h*);
vm ← *RETURN*(*vmtf-en-dequeue M* (*to-remove*′!*i*) *vm*);
RETURN (*i*+*1*, *vm*, *atoms-hash-del* (*to-remove*′!*i*) *h*)})
(*0*, *vm*, *h*);
RETURN (*vm*, (*emptied-list to-remove*′, *h*))
})› **for** $\mathcal{A}_{in}$
unfolding *vmtf-flush-int-def*
by *auto*

have *reorder-list*: ‹*vmtf-reorder-list x2c x1e*
≤ ⇓ *Id*
(*reorder-list x2 x1b*)›
if
‹(*x*, *y*) ∈ *trail-pol* $\mathcal{A}$ ×$_f$ *Id* ×$_f$ *Id*› **and**
‹*x1* = (*x1a*, *x2*)› **and**
‹*x2a* = (*x1b*, *x2b*)› **and**
‹*y* = (*x1*, *x2a*)› **and**
‹*x1c* = (*x1d*, *x2c*)› **and**
‹*x2d* = (*x1e*, *x2e*)› **and**
‹*x* = (*x1c*, *x2d*)› **and**
‹∀ *x*∈*set x1b*. *x* < *length* (*fst x2*)› **and**
‹*length x1b* ≤ *unat32-max*› **and**
‹∀ *x*∈*set x1e*. *x* < *length* (*fst x2c*)› **and**
‹*length x1e* ≤ *unat32-max*›
for *x y x1 x1a x2 x2a x1b x2b x1c x1d x2c x2d x1e x2e*
using *that* **unfolding** *vmtf-reorder-list-def* **by** (*cases x2*)
(*auto intro*!: *ASSERT-leI simp*: *reorder-list-def vmtf-reorder-list-raw-def*)

have *vmtf-rescale*: ‹*vmtf-rescale x2c*
≤ ⇓ *Id*
(*vmtf-rescale x2*)›
if
‹(*x*, *y*) ∈ *trail-pol* $\mathcal{A}$ ×$_f$ *Id* ×$_f$ *Id*› **and**
‹*x1* = (*x1a*, *x2*)› **and**
‹*x2a* = (*x1b*, *x2b*)› **and**

```
    ‹y = (x1, x2a)› and
    ‹x1c = (x1d, x2c)› and
    ‹x2d = (x1e, x2e)› and
    ‹x = (x1c, x2d)› and
    ‹∀x∈set x1b. x < length (fst x2)› and
    ‹length x1b ≤ unat32-max› and
    ‹∀x∈set x1e. x < length (fst x2c)› and
    ‹length x1e ≤ unat32-max› and
    ‹(to-remove', to-remove'a) ∈ Id› and
    ‹length to-remove'a ≤ unat32-max› and
    ‹length to-remove' ≤ unat32-max› and
    ‹unat64-max − fst (snd x2c) ≤ length to-remove'› and
    ‹unat64-max ≤ length to-remove'a + fst (snd x2)›
  for x y x1 x1a x2 x2a x1b x2b x1c x1d x2c x2d x1e x2e
    to-remove' to-remove'a
  using that by auto
have loop-rel: ‹((0, vm, x2e), 0, vma, x2b)
    ∈ Id›
  if
    True and
    ‹(x, y) ∈ trail-pol A ×f Id ×f Id› and
    ‹x1 = (x1a, x2)› and
    ‹x2a = (x1b, x2b)› and
    ‹y = (x1, x2a)› and
    ‹x1c = (x1d, x2c)› and
    ‹x2d = (x1e, x2e)› and
    ‹x = (x1c, x2d)› and
    ‹∀x∈set x1b. x < length (fst x2)› and
    ‹length x1b ≤ unat32-max› and
    ‹∀x∈set x1e. x < length (fst x2c)› and
    ‹length x1e ≤ unat32-max› and
    ‹(to-remove', to-remove'a) ∈ Id› and
    ‹length to-remove'a ≤ unat32-max› and
    ‹length to-remove' ≤ unat32-max› and
    ‹(vm, vma) ∈ Id› and
    ‹length to-remove'a + fst (snd vma) ≤ unat64-max› and
    ‹length to-remove' + fst (snd vm) ≤ unat64-max› and
    ‹case (0, vma, x2b) of
   (i, vm', h) ⇒
     i ≤ length to-remove'a ∧
     fst (snd vm') = i + fst (snd vma) ∧
     (i < length to-remove'a ⟶
      vmtf-en-dequeue-pre A ((x1a, to-remove'a ! i), vm'))›
  for x y x1 x1a x2 x2a x1b x2b x1c x1d x2c x2d x1e x2e
    to-remove' to-remove'a vm vma
  using that by auto

have isa-vmtf-en-dequeue-pre:
  ‹vmtf-en-dequeue-pre A ((M, L), x) ⟹ isa-vmtf-en-dequeue-pre ((M', L), x)› for x M M' L
  unfolding vmtf-en-dequeue-pre-def isa-vmtf-en-dequeue-pre-def
  by auto

have isa-vmtf-en-dequeue:
  ‹isa-vmtf-en-dequeue x1d (to-remove' ! x1h) x1i
    ≤ Refine-Basic.SPEC
        (λc. (c, vmtf-en-dequeue x1a (to-remove'a ! x1f) x1g)
```

```
        ∈ Id)›
  if
    ‹(x, y) ∈ trail-pol A ×f Id ×f Id› and
    ‹x1 = (x1a, x2)› and
    ‹x2a = (x1b, x2b)› and
    ‹y = (x1, x2a)› and
    ‹x1c = (x1d, x2c)› and
    ‹x2d = (x1e, x2e)› and
    ‹x = (x1c, x2d)› and
    ‹∀ x∈set x1b. x < length (fst x2)› and
    ‹length x1b ≤ unat32-max› and
    ‹∀ x∈set x1e. x < length (fst x2c)› and
    ‹length x1e ≤ unat32-max› and
    ‹(to-remove', to-remove'a) ∈ Id› and
    ‹length to-remove'a ≤ unat32-max› and
    ‹length to-remove' ≤ unat32-max› and
    ‹(vm, vma) ∈ Id› and
    ‹length to-remove'a + fst (snd vma) ≤ unat64-max› and
    ‹length to-remove' + fst (snd vm) ≤ unat64-max› and
    ‹(xa, x') ∈ Id› and
    ‹case xa of (i, vm, h) ⇒ i < length to-remove'› and
    ‹case x' of (i, vm, h) ⇒ i < length to-remove'a› and
    ‹case xa of
   (i, vm', h) ⇒
     i ≤ length to-remove' ∧
     fst (snd vm') = i + fst (snd vm) ∧
     (i < length to-remove' ⟶
      isa-vmtf-en-dequeue-pre ((x1d, to-remove' ! i), vm'))› and
    ‹case x' of
   (i, vm', h) ⇒
     i ≤ length to-remove'a ∧
     fst (snd vm') = i + fst (snd vma) ∧
     (i < length to-remove'a ⟶
      vmtf-en-dequeue-pre A ((x1a, to-remove'a ! i), vm'))› and
    ‹x2f = (x1g, x2g)› and
    ‹x' = (x1f, x2f)› and
    ‹x2h = (x1i, x2i)› and
    ‹xa = (x1h, x2h)› and
    ‹x1f < length to-remove'a› and
    ‹to-remove'a ! x1f ∈# A› and
    ‹atoms-hash-del-pre (to-remove'a ! x1f) x2g› and
    ‹x1h < length to-remove'› and
    ‹isa-vmtf-en-dequeue-pre ((x1d, to-remove' ! x1h), x1i)›
  for x y x1 x1a x2 x2a x1b x2b x1c x1d x2c x2d x1e
    x2e to-remove' to-remove'a vm vma xa x' x1f x2f x1g
    x2g x1h x2h x1i x2i
  using isa-vmtf-en-dequeue[of A, THEN fref-to-Down-curry2, of x1a ‹to-remove'a ! x1f› x1g
      x1d ‹to-remove' ! x1h› x1i] that
  by (auto simp: RETURN-def)

show ?thesis
 unfolding isa-vmtf-flush-int-def uncurry-def vmtf-flush-int-alt-def
  apply (intro frefI nres-relI)
  apply (refine-rcg)
  subgoal
    by auto
```

```
    subgoal
      by auto
      apply (rule reorder-list; assumption)
    subgoal
      by auto
    subgoal
      by auto
    apply (rule vmtf-rescale; assumption)
    subgoal
      by auto
    subgoal
      by auto
    apply (rule loop-rel; assumption)
    subgoal
      by auto
    subgoal
      by auto
    subgoal
      by (auto intro!: isa-vmtf-en-dequeue-pre)
    subgoal
      by auto
    subgoal
      by auto
    subgoal
      by auto
    apply (rule isa-vmtf-en-dequeue; assumption)
    subgoal for x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e to-remove' to-remove'a vm
       vma xa x' x1f x2f x1g x2g x1h x2h x1i x2i vmb vmc
      by auto
    subgoal
      by auto
    subgoal
      by auto
    done
qed

lemma bounded-included-le:
   assumes bounded: ‹isasat-input-bounded 𝒜› and ‹distinct n› and
   ‹set n ⊆ set-mset 𝒜 ›
  shows ‹length n < unat32-max›  ‹length n ≤ 1 + unat32-max div 2›
proof −
  have lits: ‹literals-are-in-ℒ_in 𝒜 (Pos `# mset n)› and
    dist: ‹distinct n›
    using assms
    by (auto simp: literals-are-in-ℒ_in-alt-def distinct-atoms-rel-alt-def inj-on-def atms-of-ℒ_all-𝒜_in)
   have dist: ‹distinct-mset (Pos `# mset n)›
    by (subst distinct-image-mset-inj)
      (use dist in ‹auto simp: inj-on-def›)
  have tauto: ‹¬ tautology (poss (mset n))›
    by (auto simp: tautology-decomp)

  show ‹length n < unat32-max›  ‹length n ≤ 1 + unat32-max div 2›
    using simple-clss-size-upper-div2[OF bounded lits dist tauto]
    by (auto simp: unat32-max-def)
qed
```

lemma *atms-hash-insert-pre*:
assumes ‹$L \in\# \mathcal{A}$› **and** ‹$(x, x') \in$ *distinct-atoms-rel* $\mathcal{A}$› **and** ‹*isasat-input-bounded* $\mathcal{A}$›
shows ‹*atms-hash-insert-pre* L x›
using *assms bounded-included-le*[*OF assms*(3), *of* ‹L # *fst* x›]
by (*auto simp*: *atoms-hash-insert-def atoms-hash-rel-def distinct-atoms-rel-alt-def*
atms-hash-insert-pre-def)

lemma *atoms-hash-del-op-set-insert*:
‹(*uncurry* (*RETURN oo atoms-hash-insert*), *uncurry* (*RETURN oo insert*)) ∈
[$\lambda(i, xs)$. $i \in\# \mathcal{A}_{in} \wedge$ *isasat-input-bounded* $\mathcal{A}$]$_f$
nat-rel $\times_r$ *distinct-atoms-rel* $\mathcal{A}_{in} \to \langle$*distinct-atoms-rel* $\mathcal{A}_{in}\rangle$*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *atoms-hash-insert-def atoms-hash-rel-def distinct-atoms-rel-alt-def intro*!: *ASSERT-leI*)

definition (**in** −) *atoms-hash-set-member* **where**
‹*atoms-hash-set-member* i xs = *do* {*ASSERT*($i <$ *length* xs); *RETURN* (xs ! i)}›

definition *isa-vmtf-mark-to-rescore*
:: ‹*nat* ⇒ *vmtf* ⇒ - ⇒ *isa-vmtf-remove-int*›
where
‹*isa-vmtf-mark-to-rescore* L = (λ(*ns*, *m*, *fst-As*, *next-search*) *to-remove*.
((*ns*, *m*, *fst-As*, *next-search*), *atoms-hash-insert* L *to-remove*))›

definition *isa-vmtf-mark-to-rescore-pre* **where**
‹*isa-vmtf-mark-to-rescore-pre* = (λL ((*ns*, *m*, *fst-As*, *next-search*), *to-remove*).
atms-hash-insert-pre L *to-remove*)›

lemma *size-conflict-int-size-conflict*:
‹(*RETURN o size-conflict-int*, *RETURN o size-conflict*) ∈ [λD. $D \neq$ *None*]$_f$ *option-lookup-clause-rel*
$\mathcal{A} \to$
$\langle$*nat-rel*$\rangle$*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *size-conflict-int-def size-conflict-def option-lookup-clause-rel-def*
lookup-clause-rel-def)

definition *rescore-clause*
:: ‹*nat multiset* ⇒ *nat clause-l* ⇒ (*nat*,*nat*)*ann-lits* ⇒ *vmtf* ⇒
vmtf nres›
where
‹*rescore-clause* $\mathcal{A}$ C M *vm* = *SPEC* (λ(*vm'*). *vm'* ∈ *vmtf* $\mathcal{A}$ M)›

lemma *isa-vmtf-unset-vmtf-unset*:
‹(*uncurry* (*RETURN oo isa-vmtf-unset*), *uncurry* (*RETURN oo vmtf-unset*)) ∈
nat-rel $\times_f$ (*Id*) $\to_f$
$\langle$(*Id*)$\rangle$*nres-rel*›
unfolding *vmtf-unset-def isa-vmtf-unset-def uncurry-def*
by (*intro frefI nres-relI*) *auto*

lemma *isa-vmtf-unset-isa-vmtf*:
assumes ‹*vm* ∈ *vmtf* $\mathcal{A}$ M› **and** ‹$L \in\# \mathcal{A}$›
shows ‹*isa-vmtf-unset* L *vm* ∈ *vmtf* $\mathcal{A}$ M›

proof −
  obtain *vm0 to-remove to-remove'* **where**
    vm: ‹*vm* = (*vm0*, *to-remove*)› **and**
    vm0: ‹(*vm0*, *to-remove'*) ∈ *vmtf* $\mathcal{A}$ *M*›
    using *assms* **by** (*cases vm*) (*auto simp*:)

  then show *?thesis*
    using *assms*
      isa-vmtf-unset-vmtf-unset[*THEN fref-to-Down-unRET-uncurry*, *of L vm L vm*]
    using
      abs-vmtf-ns-unset-vmtf-unset[*of* ‹*fst vm*› ‹*fst* (*snd vm*)› ‹*fst* (*snd* (*snd vm*))›
        ‹*fst* (*snd* (*snd* (*snd vm*)))› ‹*snd* (*snd* (*snd* (*snd vm*)))› $\mathcal{A}$ *M L*]
    by (*auto simp*: *vm atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *intro*: *elim*!: *prod-relE*)
qed

lemma *isa-vmtf-tl-isa-vmtf*:
  assumes ‹*vm* ∈ *vmtf* $\mathcal{A}$ *M*› **and** ‹*M* ≠ []› **and** ‹*lit-of* (*hd M*) ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and**
    ‹*L* = (*atm-of* (*lit-of* (*hd M*)))›
  shows ‹*isa-vmtf-unset L vm* ∈ *vmtf* $\mathcal{A}$ (*tl M*)›
proof −
  let *?L* = ‹*atm-of* (*lit-of* (*hd M*))›
  obtain *vm0 to-remove to-remove'* **where**
    vm: ‹*vm* = (*vm0*, *to-remove*)› **and**
    vm0: ‹(*vm0*, *to-remove'*) ∈ *vmtf* $\mathcal{A}$ *M*›
    using *assms* **by** (*cases vm*) (*auto simp*:)

  then show *?thesis*
    using *assms*
      isa-vmtf-unset-vmtf-unset[*THEN fref-to-Down-unRET-uncurry*, *of ?L vm ?L vm*]
    using *vmtf-unset-vmtf-tl*[*of* ‹*fst vm*› ‹*fst* (*snd vm*)› ‹*fst* (*snd* (*snd vm*))›
        ‹*fst* (*snd* (*snd* (*snd vm*)))› ‹*snd* (*snd* (*snd* (*snd vm*)))› $\mathcal{A}$ *M*]
    by (*cases M*)
     (*auto simp*: *vm atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *elim*!: *prod-relE*)
qed

definition *isa-vmtf-find-next-undef* :: ‹*vmtf* ⇒ *trail-pol* ⇒ (*nat option*) *nres*› **where**
‹*isa-vmtf-find-next-undef* = (λ(*ns*, *m*, *fst-As*, *lst-As*, *next-search*) *M*. *do* {
    WHILE$_T$λ*next-search*. *next-search* ≠ *None* ⟶ *defined-atm-pol-pre M* (*the next-search*)
      (λ*next-search*. *next-search* ≠ *None* ∧ *defined-atm-pol M* (*the next-search*))
      (λ*next-search*. *do* {
        ASSERT(*next-search* ≠ *None*);
        let n = *the next-search*;
        ASSERT (*n* < *length ns*);
        RETURN (*get-next* (*ns*!*n*))
       }
)
      next-search
  })›

lemma *isa-vmtf-find-next-undef-vmtf-find-next-undef*:
  ‹(*uncurry isa-vmtf-find-next-undef*, *uncurry* (*vmtf-find-next-undef* $\mathcal{A}$)) ∈
     Id ×$_r$ *trail-pol* $\mathcal{A}$ →$_f$ ⟨⟨*nat-rel*⟩*option-rel*⟩*nres-rel* ›
  unfolding *isa-vmtf-find-next-undef-def vmtf-find-next-undef-def uncurry-def*
    defined-atm-def[*symmetric*]
  apply (*intro frefI nres-relI*)

apply *refine-rcg*
subgoal by *auto*
subgoal by (*rule defined-atm-pol-pre*) (*auto simp*: *in-$\mathcal{L}_{all}$-atm-of-$\mathcal{A}_{in}$*)
subgoal
by (*auto simp*: *undefined-atm-code*[*THEN fref-to-Down-unRET-uncurry-Id*])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done

end
theory *IsaSAT-Bump-Heuristics*
imports *Watched-Literals-VMTF*
IsaSAT-VMTF
Tuple4
IsaSAT-Bump-Heuristics-State
begin

13.2 Bumping

thm *isa-vmtf-find-next-undef-def*
term *isa-acids-find-next-undef*
definition *isa-acids-find-next-undef* :: ‹(*nat*, *nat*) *acids* ⇒ *trail-pol* ⇒ (*nat option* × (*nat*, *nat*) *acids*) *nres*› **where**
‹*isa-acids-find-next-undef* = (λ*ac M*. *do* {
WHILE$_T$$^{\lambda(L,\ ac).\ True}$
(λ(*nxt*, *ac*). *nxt* = *None* ∧ *acids-mset ac* ≠ {#})
(λ(*a*, *ac*). *do* {
ASSERT (*a* = *None*);
(*L*, *ac*) ← *acids-pop-min ac*;
ASSERT (*defined-atm-pol-pre M L*);
if defined-atm-pol M L then RETURN (*None*, *ac*)
else RETURN (*Some L*, *ac*)
}
)
(*None*, *ac*)
})›

lemma *isa-acids-find-next-undef-acids-find-next-undef*:
‹(*uncurry isa-acids-find-next-undef*, *uncurry* (*acids-find-next-undef* $\mathcal{A}$)) ∈
Id ×$_r$ *trail-pol* $\mathcal{A}$ →$_f$ ⟨⟨*nat-rel*⟩*option-rel* ×$_r$ *Id*⟩*nres-rel* ›
proof −
have [*refine*]: ‹*a*=*b*⟹ *acids-pop-min a* ≤ ⇓ *Id* (*acids-pop-min b*)›**for** *a b*
by *auto*
show *?thesis*
unfolding *isa-acids-find-next-undef-def acids-find-next-undef-def uncurry-def*
defined-atm-def[*symmetric*]
apply (*intro frefI nres-relI*)
apply *refine-rcg*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*rule defined-atm-pol-pre*) (*auto simp*: *in-$\mathcal{L}_{all}$-atm-of-$\mathcal{A}_{in}$*)

```
  subgoal
    by (auto simp: undefined-atm-code[THEN fref-to-Down-unRET-uncurry-Id])
  subgoal by auto
  subgoal by auto
  done
qed

definition vmtf-rescore-body
 :: ‹nat multiset ⇒ nat clause-l ⇒ (nat,nat) ann-lits ⇒ bump-heuristics ⇒
    (nat × bump-heuristics) nres›
where
  ‹vmtf-rescore-body 𝒜_in C - vm = do {
        WHILE_T^λ(i, vm). i ≤ length C
          (λ(i, vm). i < length C)
          (λ(i, vm). do {
              ASSERT(i < length C);
              ASSERT(atm-of (C!i) ∈# 𝒜_in);
              vm' ← isa-bump-mark-to-rescore (atm-of (C!i)) vm;
              RETURN(i+1, vm')
            })
          (0, vm)
    }›

definition vmtf-rescore
 :: ‹nat multiset ⇒ nat clause-l ⇒ (nat,nat) ann-lits ⇒ bump-heuristics ⇒
       (bump-heuristics) nres›
where
  ‹vmtf-rescore 𝒜_in C M vm = do {
      (-, vm) ← vmtf-rescore-body 𝒜_in C M vm;
      RETURN (vm)
    }›

definition isa-bump-rescore-body
 :: ‹nat clause-l ⇒ trail-pol ⇒ bump-heuristics ⇒
    (nat × bump-heuristics) nres›
where
  ‹isa-bump-rescore-body C - vm = do {
        WHILE_T^λ(i, vm). i ≤ length C
          (λ(i, vm). i < length C)
          (λ(i, vm). do {
              ASSERT(i < length C);
              vm' ← isa-bump-mark-to-rescore (atm-of (C!i)) vm;
              RETURN(i+1, vm')
            })
          (0, vm)
    }›

definition isa-bump-rescore
 :: ‹nat clause-l ⇒ trail-pol ⇒ bump-heuristics ⇒ bump-heuristics nres›
where
  ‹isa-bump-rescore C M vm = do {
      (-, vm) ← isa-bump-rescore-body C M vm;
      RETURN (vm)
    }›
```

definition *isa-rescore-clause*
:: ‹*nat multiset* ⇒ *nat clause-l* ⇒ (*nat*,*nat*)*ann-lits* ⇒ *bump-heuristics* ⇒
bump-heuristics nres›
where
‹*isa-rescore-clause* $\mathcal{A}$ *C M vm* = *SPEC* (λ(*vm'*). *vm'* ∈ *bump-heur* $\mathcal{A}$ *M*)›

lemma *isa-bump-mark-to-rescore*:
assumes
‹*L* ∈# $\mathcal{A}$› ‹*b* ∈ *bump-heur* $\mathcal{A}$ *M*› ‹*length* (*fst* (*get-bumped-variables b*)) < *Suc* (*Suc* (*unat32-max div* 2))›
shows ‹
isa-bump-mark-to-rescore L b ≤ *SPEC* (λ*vm'*. *vm'* ∈ *bump-heur* $\mathcal{A}$ *M*)›
proof −
have [*simp*]: ‹(*atoms-hash-insert L x*, *set* (*fst* (*atoms-hash-insert L x*))) ∈ *distinct-atoms-rel* $\mathcal{A}$›
if ‹(*x*, *set* (*fst x*)) ∈ *distinct-atoms-rel* $\mathcal{A}$›
for *x*
unfolding *distinct-atoms-rel-def*
apply (*rule relcompI*[*of* - ‹(*fst* (*atoms-hash-insert L x*), *insert L* (*set* (*fst x*)))›])
using *that assms*(*1*)
by (*auto simp*: *distinct-atoms-rel-def atoms-hash-rel-def atoms-hash-insert-def*
distinct-hash-atoms-rel-def split: *if-splits*)

show *?thesis*
using *assms*
unfolding *isa-bump-mark-to-rescore-def*
apply (*auto split*: *bump-heuristics-splits simp*: *atms-hash-insert-pre-def*
intro!: *ASSERT-leI*)
apply (*auto simp*: *bump-heur-def distinct-atoms-rel-def atoms-hash-rel-def atoms-hash-insert-def*
distinct-hash-atoms-rel-def intro: *relcompI dest*!: *multi-member-split*[*of L*])[]
apply (*auto simp*: *bump-heur-def intro*: *dest*!: *multi-member-split*[*of L*])
done
qed

lemma *length-get-bumped-variables-le*:
assumes ‹*vm* ∈ *bump-heur* $\mathcal{A}$ *M*› ‹*isasat-input-bounded* $\mathcal{A}$›
shows ‹*length* (*fst* (*get-bumped-variables vm*)) < *Suc* (*Suc* (*unat32-max div* 2))›
using *assms bounded-included-le*[*of* $\mathcal{A}$ ‹*fst* (*get-bumped-variables vm*)›]
by (*cases vm*)
(*auto simp*: *bump-heur-def distinct-atoms-rel-def distinct-hash-atoms-rel-def*
atoms-hash-rel-def)

lemma *vmtf-rescore-score-clause*:
‹(*uncurry2* (*vmtf-rescore* $\mathcal{A}$), *uncurry2* (*isa-rescore-clause* $\mathcal{A}$)) ∈
[λ((*C*, *M*), *vm*). *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*) ∧ *vm* ∈ *bump-heur* $\mathcal{A}$ *M* ∧ *isasat-input-bounded* $\mathcal{A}$]$_f$
⟨*Id*⟩*list-rel* ×$_f$ *Id* ×$_f$ *Id* → ⟨*Id*⟩ *nres-rel*›
proof −
have *H*: ‹*vmtf-rescore-body* $\mathcal{A}$ *C M vm* ≤
SPEC (λ(*n* :: *nat*, *vm'*). *vm'* ∈ *bump-heur* $\mathcal{A}$ *M*)›
if *M*: ‹*vm* ∈ *bump-heur* $\mathcal{A}$ *M*› **and** *C*: ‹∀ *c*∈*set C*. *atm-of c* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
for *C vm* φ *M*
unfolding *vmtf-rescore-body-def*
apply (*refine-vcg WHILEIT-rule-stronger-inv*[**where** *R* = ‹*measure* (λ(*i*, -). *length C* − *i*)› **and**
I' = ‹λ(*i*, *vm'*). *vm'* ∈ *bump-heur* $\mathcal{A}$ *M*›] *isa-bump-mark-to-rescore*[**where** $\mathcal{A}$=$\mathcal{A}$ **and** *M*=*M*])
subgoal by *auto*

subgoal by *auto*
subgoal using *C M* **by** (*auto simp*: *vmtf-def phase-saving-def*)
subgoal using *C M* **by** *auto*
subgoal using *C* **by** (*auto simp*: *atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$)
subgoal using *C* **by** *auto*
subgoal using *C length-get-bumped-variables-le*[*OF - bounded*] **by** *auto*
subgoal using *C* **by** *auto*
subgoal using *C* **by** *auto*
subgoal by *auto*
done
have *K*: ‹((*a*, *b*),(*a'*, *b'*)) ∈ *A* ×$_f$ *B* ⟷ (*a*, *a'*) ∈ *A* ∧ (*b*, *b'*) ∈ *B*› **for** *a b a' b' A B*
by *auto*
show *?thesis*
unfolding *vmtf-rescore-def rescore-clause-def uncurry-def*
apply (*intro frefI nres-relI*)
apply *clarify*
apply (*rule bind-refine-spec*)
prefer *2*
apply (*subst* (*asm*) *K*)
apply (*rule H*; *auto*)
subgoal
by (*meson atm-of-lit-in-atms-of contra-subsetD in-all-lits-of-m-ain-atms-of-iff*
in-multiset-in-set literals-are-in-$\mathcal{L}_{in}$*-def*)
subgoal by (*auto simp*: *isa-rescore-clause-def*)
done
qed

lemma *isa-vmtf-rescore-body*:
‹(*uncurry2* (*isa-bump-rescore-body*), *uncurry2* (*vmtf-rescore-body* $\mathcal{A}$)) ∈ [λ-. *isasat-input-bounded* $\mathcal{A}$]$_f$
(*Id* ×$_f$ *trail-pol* $\mathcal{A}$ ×$_f$ *Id*) → ⟨*Id* ×$_r$ *Id*⟩ *nres-rel*›
proof −
show *?thesis*
unfolding *isa-bump-rescore-body-def vmtf-rescore-body-def uncurry-def*
apply (*intro frefI nres-relI*)
apply *refine-rcg*
subgoal by *auto*
subgoal by *auto*
subgoal for *x y x1 x1a x1b x2 x2a x2b x1c x1d x1e x2c x1g x2g*
by (*cases x2g*) *auto*
subgoal by *auto*
apply (*rule refine-IdI*[*OF eq-refl*])
subgoal by *auto*
subgoal by *auto*
done
qed

lemma *isa-vmtf-rescore*:
‹(*uncurry2* (*isa-bump-rescore*), *uncurry2* (*vmtf-rescore* $\mathcal{A}$)) ∈ [λ-. *isasat-input-bounded* $\mathcal{A}$]$_f$
(*Id* ×$_f$ *trail-pol* $\mathcal{A}$ ×$_f$ (*Id*)) → ⟨(*Id*)⟩ *nres-rel*›
proof −
show *?thesis*
unfolding *isa-bump-rescore-def vmtf-rescore-def uncurry-def*
apply (*intro frefI nres-relI*)
apply (*refine-rcg isa-vmtf-rescore-body*[*THEN fref-to-Down-curry2*])
subgoal by *auto*

subgoal by *auto*
done
qed

definition *vmtf-mark-to-rescore-clause* **where**
‹*vmtf-mark-to-rescore-clause* $\mathcal{A}_{in}$ *arena C vm* = *do* {
ASSERT(*arena-is-valid-clause-idx arena C*);
nfoldli
([*C*..<*C* + (*arena-length arena C*)])
(λ-. *True*)
(λ*i vm*. *do* {
ASSERT(*i* < *length arena*);
ASSERT(*arena-lit-pre arena i*);
ASSERT(*atm-of* (*arena-lit arena i*) ∈# $\mathcal{A}_{in}$);
isa-bump-mark-to-rescore (*atm-of* (*arena-lit arena i*)) *vm*
})
vm
}›

definition *isa-bump-mark-to-rescore-clause* **where**
‹*isa-bump-mark-to-rescore-clause arena C vm* = *do* {
ASSERT(*arena-is-valid-clause-idx arena C*);
nfoldli
([*C*..<*C* + (*arena-length arena C*)])
(λ-. *True*)
(λ*i vm*. *do* {
ASSERT(*i* < *length arena*);
ASSERT(*arena-lit-pre arena i*);
isa-bump-mark-to-rescore (*atm-of* (*arena-lit arena i*)) *vm*
})
vm
}›

lemma *isa-bump-mark-to-rescore-clause-vmtf-mark-to-rescore-clause*:
‹(*uncurry2 isa-bump-mark-to-rescore-clause*, *uncurry2* (*vmtf-mark-to-rescore-clause* $\mathcal{A}$)) ∈ [λ-. *isasat-input-bounded* $\mathcal{A}$]$_f$
Id ×$_f$ *nat-rel* ×$_f$ (*Id*) → ⟨*Id*⟩*nres-rel*›
unfolding *isa-bump-mark-to-rescore-clause-def vmtf-mark-to-rescore-clause-def*
uncurry-def
apply (*intro frefI nres-relI*)
apply (*refine-rcg nfoldli-refine*[**where** *R* = ‹*Id*› **and** *S* = *Id*])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal for *x y x1 x1a x2 x2a x1b x1c x2b x2c xi xa si s*
by (*cases s*)
(*auto simp*: *isa-vmtf-mark-to-rescore-pre-def*
intro!: *atms-hash-insert-pre*)
done

lemma *vmtf-mark-to-rescore-clause-spec*:
 ‹*vm* ∈ *bump-heur* $\mathcal{A}$ *M* ⟹ *valid-arena arena N vdom* ⟹ *C* ∈# *dom-m N* ⟹ *isasat-input-bounded* $\mathcal{A}$ ⟹
 (∀ *C* ∈ *set* [*C*..<*C* + *arena-length arena C*]. *arena-lit arena C* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$) ⟹
 vmtf-mark-to-rescore-clause $\mathcal{A}$ *arena C vm* ≤ *RES* (*bump-heur* $\mathcal{A}$ *M*)›
 unfolding *vmtf-mark-to-rescore-clause-def*
 apply (*subst RES-SPEC-conv*)
 apply (*refine-vcg nfoldli-rule*[**where** *I* = ‹λ- - *vm*. *vm* ∈ *bump-heur* $\mathcal{A}$ *M*›])
 subgoal
  unfolding *arena-lit-pre-def arena-is-valid-clause-idx-def*
  apply (*rule exI*[*of* - *N*])
  apply (*rule exI*[*of* - *vdom*])
  apply (*fastforce simp*: *arena-lifting*)
  done
 subgoal for *x it* σ
  using *arena-lifting*(7)[*of arena N vdom C* ‹*x* − *C*›]
  by (*auto simp*: *arena-lifting*(1−6) *dest*!: *in-list-in-setD*)
 subgoal for *x it* σ
  unfolding *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
  apply (*rule exI*[*of* - *C*])
  apply (*intro conjI*)
  apply (*solves* ‹*auto dest*: *in-list-in-setD*›)
  apply (*rule exI*[*of* - *N*])
  apply (*rule exI*[*of* - *vdom*])
  apply (*fastforce simp*: *arena-lifting dest*: *in-list-in-setD*)
  done
 subgoal for *x it* σ
  by *fastforce*
 subgoal for *x it* - σ
  using *length-get-bumped-variables-le*[*of* - $\mathcal{A}$]
  by (*cases* σ)
   (*auto intro*!: *isa-bump-mark-to-rescore*[*THEN order-trans*] *simp*: *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*
    dest: *in-list-in-setD*)
 done

definition *vmtf-mark-to-rescore-also-reasons*
 :: ‹*nat multiset* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *arena* ⇒ *nat literal list* ⇒ *nat literal* ⇒ - ⇒-› **where**
‹*vmtf-mark-to-rescore-also-reasons* $\mathcal{A}$ *M arena outl L vm* = *do* {
  ASSERT(*length outl* ≤ *unat32-max*);
  nfoldli
   ([*0*..<*length outl*])
   (λ-. *True*)
   (λ*i vm*. *do* {
    ASSERT(*i* < *length outl*); *ASSERT*(*length outl* ≤ *unat32-max*);
    ASSERT(−*outl* ! *i* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$);
    if(*outl*!*i* = *L*)
    then
     RETURN vm
    else do {
     C ← *get-the-propagation-reason M* (−(*outl* ! *i*));
     case C of
      None ⇒ *isa-bump-mark-to-rescore* (*atm-of* (*outl* ! *i*)) *vm*
     | *Some C* ⇒ *if C* = *0 then RETURN vm else vmtf-mark-to-rescore-clause* $\mathcal{A}$ *arena C vm*}
   })
   vm

```
  }›

definition isa-vmtf-mark-to-rescore-also-reasons
  :: ‹trail-pol ⇒ arena ⇒ nat literal list ⇒ nat literal ⇒ - ⇒-› where
‹isa-vmtf-mark-to-rescore-also-reasons M arena outl L vm = do {
    ASSERT(length outl ≤ unat32-max);
    nfoldli
      ([0..<length outl])
      (λ-. True)
      (λi vm. do {
        ASSERT(i < length outl); ASSERT(length outl≤ unat32-max);
        if(outl!i = L)
        then
          RETURN vm
        else do {
              C ← get-the-propagation-reason-pol M (−(outl ! i));
              case C of
                None ⇒ do {
                  isa-bump-mark-to-rescore (atm-of (outl ! i)) vm
                }
              | Some C ⇒ if C = 0 then RETURN vm else isa-bump-mark-to-rescore-clause arena C vm
            }
          })
      vm
  }›

lemma isa-vmtf-mark-to-rescore-also-reasons-vmtf-mark-to-rescore-also-reasons:
  ‹(uncurry4 isa-vmtf-mark-to-rescore-also-reasons, uncurry4 (vmtf-mark-to-rescore-also-reasons 𝒜)) ∈
    [λ-. isasat-input-bounded 𝒜]_f
  trail-pol 𝒜 ×_f Id ×_f Id ×_f Id ×_f Id → ⟨Id⟩nres-rel›
  unfolding isa-vmtf-mark-to-rescore-also-reasons-def vmtf-mark-to-rescore-also-reasons-def
    uncurry-def
  apply (intro frefI nres-relI)
  apply (refine-rcg nfoldli-refine[where R = ‹Id› and S = Id]
    get-the-propagation-reason-pol[of 𝒜, THEN fref-to-Down-curry]
     isa-bump-mark-to-rescore[of - 𝒜]
     isa-bump-mark-to-rescore-clause-vmtf-mark-to-rescore-clause[of 𝒜, THEN fref-to-Down-curry2])
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  apply assumption
  subgoal for x y x1 x1a - - - x1b x2 x2a x2b x1c x1d x1e x2c x2d x2e xi xa si s xb x'
    by (cases xb)
     (auto simp: isa-vmtf-mark-to-rescore-pre-def in-𝓛_all-atm-of-in-atms-of-iff
        intro!: atms-hash-insert-pre[of - 𝒜])
  subgoal by auto
  subgoal by auto
  done


lemma vmtf-mark-to-rescore-also-reasons-spec:
```

```
  ‹vm ∈ bump-heur 𝒜 M ⟹ valid-arena arena N vdom ⟹ length outl ≤ unat32-max ⟹ isasat-input-bounded
𝒜 ⟹
    (∀ L ∈ set outl. L ∈# ℒ_all 𝒜) ⟹
    (∀ L ∈ set outl. ∀ C. (Propagated (−L) C ∈ set M ⟶ C ≠ 0 ⟶ (C ∈# dom-m N ∧
        (∀ C ∈ set [C..<C + arena-length arena C]. arena-lit arena C ∈# ℒ_all 𝒜)))) ⟹
     vmtf-mark-to-rescore-also-reasons 𝒜 M arena outl L vm ≤ RES (bump-heur 𝒜 M)›
   unfolding vmtf-mark-to-rescore-also-reasons-def
   apply (subst RES-SPEC-conv)
   apply (refine-vcg nfoldli-rule[where I = ‹λ- - vm. vm ∈ bump-heur 𝒜 M›]
     isa-bump-mark-to-rescore[of - 𝒜])
   subgoal by (auto dest: in-list-in-setD)
   subgoal for x l1 l2 σ
     unfolding all-set-conv-nth
     by (auto simp: uminus-𝒜_in-iff dest!: in-list-in-setD)
   subgoal for x l1 l2 σ
     unfolding get-the-propagation-reason-def
     apply (rule SPEC-rule)
     apply (rename-tac reason, case-tac reason; simp only: option.simps RES-SPEC-conv[symmetric])
     subgoal
       using length-get-bumped-variables-le[of - 𝒜 M]
       by (auto intro!: isa-bump-mark-to-rescore[where M=M and 𝒜=𝒜, THEN order-trans]
         simp: in-ℒ_all-atm-of-in-atms-of-iff[symmetric])
     apply (rename-tac D, case-tac ‹D = 0›; simp)
     subgoal
       by (rule vmtf-mark-to-rescore-clause-spec, assumption, assumption)
        fastforce+
     done
   done

definition vmtf-mark-to-rescore-also-reasons-cl
  :: ‹nat multiset ⇒ (nat, nat) ann-lits ⇒ arena ⇒ nat ⇒ nat literal ⇒ - ⇒-› where
‹vmtf-mark-to-rescore-also-reasons-cl 𝒜 M arena C L vm = do {
    ASSERT(arena-is-valid-clause-idx arena C);
    nfoldli
      ([0..<arena-length arena C])
      (λ-. True)
      (λi vm. do {
        K ← mop-arena-lit2 arena C i;
        ASSERT(−K ∈# ℒ_all 𝒜);
        if(K = L)
        then
           RETURN vm
        else do {
          C ← get-the-propagation-reason M (−K);
          case C of
            None ⇒ isa-bump-mark-to-rescore (atm-of K) vm
          | Some C ⇒ if C = 0 then RETURN vm else vmtf-mark-to-rescore-clause 𝒜 arena C vm}
      })
      vm
  }›

definition isa-vmtf-bump-to-rescore-also-reasons-cl
  :: ‹trail-pol ⇒ arena ⇒ nat ⇒ nat literal ⇒ - ⇒-› where
‹isa-vmtf-bump-to-rescore-also-reasons-cl M arena C L vm = do {
    ASSERT(arena-is-valid-clause-idx arena C);
    nfoldli
```

```
    ([0..<arena-length arena C])
    (λ-. True)
    (λi vm. do {
       K ← mop-arena-lit2 arena C i;
      if(K = L)
      then
        RETURN vm
      else do {
           C ← get-the-propagation-reason-pol M (−K);
           case C of
             None ⇒ do {
               isa-bump-mark-to-rescore (atm-of K) vm
             }
           | Some C ⇒ if C = 0 then RETURN vm else isa-bump-mark-to-rescore-clause arena C vm
         }
       })
    vm
 }›

lemma isa-vmtf-bump-to-rescore-also-reasons-cl-vmtf-mark-to-rescore-also-reasons-cl:
  ‹(uncurry4 isa-vmtf-bump-to-rescore-also-reasons-cl, uncurry4 (vmtf-mark-to-rescore-also-reasons-cl
A)) ∈
   [λ-. isasat-input-bounded A]_f
  trail-pol A ×_f Id ×_f Id ×_f Id ×_f (Id) → ⟨Id⟩nres-rel›
proof −
  have H: ‹f = g ⟹ (f,g) ∈ Id› for f g
    by auto
  show ?thesis
    unfolding isa-vmtf-bump-to-rescore-also-reasons-cl-def vmtf-mark-to-rescore-also-reasons-cl-def
      uncurry-def mop-arena-lit2-def
    apply (intro frefI nres-relI)
    apply (refine-rcg nfoldli-refine[where R = ‹Id› and S = Id]
      get-the-propagation-reason-pol[of A, THEN fref-to-Down-curry]
       isa-bump-mark-to-rescore-clause-vmtf-mark-to-rescore-clause[of A, THEN fref-to-Down-curry2])
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    apply (rule H)
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    apply assumption
    subgoal for x y x1 x1a - - - x1b x2 x2a x2b x1c x1d x1e x2c x2d x2e xi xa si s xb x'
      by (cases xb)
       (auto simp: isa-vmtf-mark-to-rescore-pre-def in-L_all-atm-of-in-atms-of-iff
          intro!: atms-hash-insert-pre[of - A])
    subgoal by auto
    subgoal by auto
    done
qed


lemma arena-lifting-list:
```

‹*valid-arena arena N vdom* $\Longrightarrow$ $C \in\#$ *dom-m N* $\Longrightarrow$
$N \propto C$ = *map* (λi. *arena-lit arena* (C+i)) [0..<*arena-length arena C*]›
by (*subst list-eq-iff-nth-eq*)
(*auto simp*: *arena-lifting*)

lemma *vmtf-mark-to-rescore-also-reasons-cl-spec*:
‹*vm* $\in$ *bump-heur* $\mathcal{A}$ *M* $\Longrightarrow$ *valid-arena arena N vdom* $\Longrightarrow$ $C \in\#$ *dom-m N* $\Longrightarrow$ *isasat-input-bounded* $\mathcal{A}$ $\Longrightarrow$
($\forall$ $L \in$ *set* ($N \propto C$). $L \in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$) $\Longrightarrow$
($\forall$ $L \in$ *set* ($N \propto C$). $\forall$ C. (*Propagated* ($-L$) $C \in$ *set M* $\longrightarrow$ $C \neq 0$ $\longrightarrow$ ($C \in\#$ *dom-m N* $\wedge$
($\forall$ $C \in$ *set* [C..<C + *arena-length arena C*]. *arena-lit arena C* $\in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$)))) $\Longrightarrow$
vmtf-mark-to-rescore-also-reasons-cl $\mathcal{A}$ *M arena C L vm* $\leq$ *RES* (*bump-heur* $\mathcal{A}$ *M*)›
unfolding *vmtf-mark-to-rescore-also-reasons-cl-def mop-arena-lit2-def*
apply (*subst RES-SPEC-conv*)
apply (*refine-vcg nfoldli-rule*[**where** I = ‹λ- - *vm*. *vm* $\in$ *bump-heur* $\mathcal{A}$ *M*›])
subgoal by (*auto simp*: *arena-is-valid-clause-idx-def*)
subgoal for *x l1 l2* σ **by** (*auto simp*: *arena-lit-pre-def arena-lifting*
arena-is-valid-clause-idx-and-access-def intro!: *exI*[*of* - *C*] *exI*[*of* - *N*]
dest: *in-list-in-setD*)
subgoal by (*auto simp*: *arena-lifting arena-lifting-list image-image uminus-*$\mathcal{A}_{in}$*-iff*)
subgoal for *x l1 l2* σ
unfolding *get-the-propagation-reason-def*
apply (*rule SPEC-rule*)
apply (*rename-tac reason*, *case-tac reason*; *simp only*: *option.simps RES-SPEC-conv*[*symmetric*])
subgoal
using *length-get-bumped-variables-le*[*of* - $\mathcal{A}$]
by (*auto intro*!: *isa-bump-mark-to-rescore*[**where** $\mathcal{A}$=$\mathcal{A}$ **and** M=M, *THEN order-trans*]
simp: *arena-lifting-list in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*[*symmetric*])
apply (*rename-tac D*, *case-tac* ‹D = 0›; *simp*)
subgoal
by (*rule vmtf-mark-to-rescore-clause-spec*, *assumption*, *assumption*)
(*auto simp*: *arena-lifting arena-lifting-list image-image uminus-*$\mathcal{A}_{in}$*-iff*)
done
done

13.3 Backtrack level for Restarts

hide-const (**open**) *find-decomp-wl-imp*

lemma *isa-bump-unset-pre*:
assumes
‹$x \in$ *bump-heur* $\mathcal{A}$ *M*› **and**
‹$L \in\#$ $\mathcal{A}$›
shows ‹*isa-bump-unset-pre L x*›
using *assms vmtf-unset-pre-vmtf*[**where** $\mathcal{A}$=$\mathcal{A}$ **and** M=M **and** L=L]
acids-tl[**where** $\mathcal{A}$=$\mathcal{A}$ **and** M=M **and** L=L]
by (*cases* ‹*get-focused-heuristics x*›)
(*auto simp*: *isa-bump-unset-pre-def bump-heur-def acids-tl-pre-def*
atms-of-$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *acids-def*)

definition *isa-acids-flush-int* :: ‹*trail-pol* $\Rightarrow$ (*nat*, *nat*)*acids* $\Rightarrow$ - $\Rightarrow$ ((*nat*, *nat*)*acids* $\times$ -) *nres*› **where**
‹*isa-acids-flush-int* = (λ*M vm* (*to-remove*, *h*). *do* {
ASSERT(*length to-remove* $\leq$ *unat32-max*);
(-, *vm*, *h*) $\leftarrow$ *WHILE*$_T$$^{\lambda(i,\ vm',\ h).\ i \leq length\ to\text{-}remove}$
(λ(*i*, *vm*, *h*). *i* < *length to-remove*)

```
    (λ(i, vm, h). do {
      ASSERT(i < length to-remove);
      vm ← acids-push-literal (to-remove!i) vm;
  ASSERT(atoms-hash-del-pre (to-remove!i) h);
      RETURN (i+1, vm, atoms-hash-del (to-remove!i) h)})
    (0, vm, h);
   RETURN (vm, (emptied-list to-remove, h))
  })⟩

lemma isa-acids-flush-int:
  ⟨(uncurry2 isa-acids-flush-int, uncurry2 (acids-flush-int A)) ∈ trail-pol (A::nat multiset) ×f Id ×f Id
→f ⟨Id×f Id⟩nres-rel⟩
proof −
  have [refine]: ⟨x2c=x2 ⟹ x2e=x2b ⟹ ((0, x2c::(nat, nat)acids, x2e), 0, x2, x2b) ∈ nat-rel ×r Id
×r Id⟩
    for x2c x2 x2e x2b
    by auto
  have [refine]: ⟨(a,a')∈Id⟹(b,b')∈Id ⟹ acids-push-literal a b ≤⇓Id (acids-push-literal a' b')⟩ for a
a' b b'
    by auto
  show ?thesis
    unfolding isa-acids-flush-int-def acids-flush-int-def uncurry-def
    apply (intro frefI nres-relI)
    apply refine-vcg
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    done
qed


definition isa-acids-incr-score :: ⟨(nat, nat)acids ⇒ (nat, nat)acids⟩ where
  ⟨isa-acids-incr-score = (λ(a, m). (a, if m < unat64-max then m+1 else m))⟩

lemma isa-acids-incr-score: ⟨ac ∈ acids A M ⟹ isa-acids-incr-score ac ∈ acids A M⟩
  by (auto simp: isa-acids-incr-score-def acids-def)

definition isa-bump-heur-flush where
  ⟨isa-bump-heur-flush M x = (case x of Tuple4 stabl focused foc bumped ⇒ do {
 (stable, bumped) ← (if foc then RETURN (stabl, bumped) else isa-acids-flush-int M (isa-acids-incr-score
stabl) bumped);
   (focused, bumped) ← (if ¬foc then RETURN (focused, bumped) else isa-vmtf-flush-int M focused
bumped);
  RETURN (Tuple4 stable focused foc bumped)})⟩



definition isa-bump-flush
```

:: ‹*nat multiset* ⇒ (*nat*,*nat*) *ann-lits* ⇒ *bump-heuristics* ⇒ (*bump-heuristics*) *nres*›
where
‹*isa-bump-flush* $\mathcal{A}_{in}$ = (λ*M vm*. *SPEC* (λ*x*. *x* ∈ *bump-heur* $\mathcal{A}_{in}$ *M*))›

lemma *in-distinct-atoms-rel-in-atmsD*: ‹(*ba*, *y*) ∈ *distinct-atoms-rel* $\mathcal{A}$ ⟹
xa ∈ *set* (*fst ba*) ⟹ *xa* ∈# $\mathcal{A}$› **and**
distinct-atoms-rel-emptiedI: ‹((*ae*, *baa*), {}) ∈ *distinct-atoms-rel* $\mathcal{A}$ ⟹
((*ae*, *baa*), *set ae*) ∈ *distinct-atoms-rel* $\mathcal{A}$›
by (*auto simp*: *distinct-atoms-rel-def distinct-hash-atoms-rel-def*)

lemma *acids-change-to-remove-order'*:
‹(*uncurry2* (*acids-flush-int* $\mathcal{A}_{in}$), *uncurry2* (*acids-flush* $\mathcal{A}_{in}$)) ∈
[λ((*M*, *vm*), *to-r*). *vm* ∈ *acids* $\mathcal{A}_{in}$ *M* ∧ *isasat-input-bounded* $\mathcal{A}_{in}$ ∧ *isasat-input-nempty* $\mathcal{A}_{in}$ ∧ *to-r* ⊆ *set-mset* $\mathcal{A}_{in}$]$_f$
Id ×$_f$ *Id* ×$_f$ *distinct-atoms-rel* $\mathcal{A}_{in}$ → ⟨(*Id* ×$_r$ *distinct-atoms-rel* $\mathcal{A}_{in}$)⟩ *nres-rel*›
by (*intro frefI nres-relI*)
(*use* **in** ‹*auto intro*!: *acids-change-to-remove-order*›)

lemma *isa-bump-heur-flush-isa-bump-flush*:
‹(*uncurry* (*isa-bump-heur-flush*), *uncurry* (*isa-bump-flush* $\mathcal{A}$))
∈ [λ(*M*, *vm*). *vm* ∈ *bump-heur* $\mathcal{A}$ *M* ∧ *isasat-input-bounded* $\mathcal{A}$ ∧
isasat-input-nempty $\mathcal{A}$]$_f$ *trail-pol* $\mathcal{A}$ ×$_f$ *Id* → ⟨*Id*⟩*nres-rel*›
unfolding *isa-bump-heur-flush-def isa-bump-flush-def uncurry-def*
apply (*intro frefI nres-relI*)
apply (*case-tac x*, *case-tac* ‹*snd x*›, *simp only*: *snd-conv case-prod-beta tuple4.case*)
apply *refine-vcg*
subgoal by *fast*
subgoal for *x y a b x1 x2 x3 x4 aa ba*
apply (*cases y*)
apply (*auto split*: *bump-heuristics-splits simp*: *bump-heur-def vmtf-flush-def*
conc-fun-RES distinct-atoms-rel-emptiedI
intro!: *isa-vmtf-flush-int*[**where** $\mathcal{A}$=$\mathcal{A}$, *THEN fref-to-Down-curry2*, *of* - - - , *THEN order-trans*]
vmtf-change-to-remove-order'[*THEN fref-to-Down-curry2*, *of* $\mathcal{A}$ ‹*fst y*›, *THEN order-trans*]
dest: *in-distinct-atoms-rel-in-atmsD*
)
apply (*auto split*: *bump-heuristics-splits simp*: *bump-heur-def vmtf-flush-def*
conc-fun-RES distinct-atoms-rel-emptiedI
intro!: *isa-vmtf-flush-int*[**where** $\mathcal{A}$=$\mathcal{A}$, *THEN fref-to-Down-curry2*, *of* - - - , *THEN order-trans*]
vmtf-change-to-remove-order'[*THEN fref-to-Down-curry2*, *of* $\mathcal{A}$ ‹*fst y*›, *THEN order-trans*]
dest: *in-distinct-atoms-rel-in-atmsD*
)
done
subgoal for *x y a b x1 x2 x3 x4*
apply (*cases y*)
apply (*auto split*: *bump-heuristics-splits simp*: *bump-heur-def vmtf-flush-def*
conc-fun-RES distinct-atoms-rel-emptiedI
intro!: *isa-acids-flush-int*[**where** $\mathcal{A}$=$\mathcal{A}$, *THEN fref-to-Down-curry2*, *of* - - - , *THEN order-trans*]
acids-change-to-remove-order'[*THEN fref-to-Down-curry2*, *of* $\mathcal{A}$ ‹*fst y*›, *THEN order-trans*]
dest!: *isa-acids-incr-score*[*of x1*]
dest: *in-distinct-atoms-rel-in-atmsD*
)
apply (*auto split*: *bump-heuristics-splits simp*: *bump-heur-def acids-flush-def*
conc-fun-RES distinct-atoms-rel-emptiedI
dest: *in-distinct-atoms-rel-in-atmsD*)
done
done

We here find out how many decisions can be reused. Remark that since VMTF does not reuse many levels anyway, the implementation might be mostly useless, but I was not aware of that when I implemented it.

definition *find-decomp-w-ns-pre* **where**
‹*find-decomp-w-ns-pre* $\mathcal{A}$ = (λ((*M*, *highest*), *vm*).
no-dup M $\wedge$
highest < *count-decided M* $\wedge$
isasat-input-bounded $\mathcal{A}$ $\wedge$
literals-are-in-$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *M* $\wedge$
vm $\in$ *bump-heur* $\mathcal{A}$ *M*)›

definition *find-decomp-wl-imp*
:: ‹*nat multiset* $\Rightarrow$ (*nat*, *nat*) *ann-lits* $\Rightarrow$ *nat* $\Rightarrow$ *bump-heuristics* $\Rightarrow$
((*nat*, *nat*) *ann-lits* $\times$ *bump-heuristics*) *nres*›
where
‹*find-decomp-wl-imp* $\mathcal{A}$ = (λ*M*$_0$ *lev vm*. *do* {
let k = *count-decided M*$_0$;
let M$_0$ = *trail-conv-to-no-CS M*$_0$;
let n = *length M*$_0$;
pos $\leftarrow$ *get-pos-of-level-in-trail M*$_0$ *lev*;
ASSERT((*n* − *pos*) $\leq$ *unat32-max*);
ASSERT(*n* $\geq$ *pos*);
let target = *n* − *pos*;
(-, *M*, *vm*′) $\leftarrow$
WHILE$_T$$^{\lambda(j,\ M,\ vm').\ j \leq target \wedge \quad M = drop\ j\ M_0 \wedge target \leq length\ M_0 \wedge \quad vm' \in bump\text{-}heur\ \mathcal{A}\ M \wedge lite}$
(λ(*j*, *M*, *vm*). *j* < *target*)
(λ(*j*, *M*, *vm*). *do* {
ASSERT (*count-decided M* > *lev*);
ASSERT(*M* ≠ []);
ASSERT(*Suc j* ≤ *unat32-max*);
let L = *atm-of* (*lit-of-hd-trail M*);
ASSERT(*L* ∈# $\mathcal{A}$);
vm ←*isa-bump-unset L vm*;
RETURN (*j* + *1*, *tl M*, *vm*)
})
(*0*, *M*$_0$, *vm*);
ASSERT(*lev* = *count-decided M*);
let M = *trail-conv-back lev M*;
RETURN (*M*, *vm*′)
})›

definition *isa-find-decomp-wl-imp*
:: ‹*trail-pol* $\Rightarrow$ *nat* $\Rightarrow$ *bump-heuristics* $\Rightarrow$ (*trail-pol* $\times$ *bump-heuristics*) *nres*›
where
‹*isa-find-decomp-wl-imp* = (λ*M*$_0$ *lev vm*. *do* {
let k = *count-decided-pol M*$_0$;
let M$_0$ = *trail-pol-conv-to-no-CS M*$_0$;
ASSERT(*isa-length-trail-pre M*$_0$);
let n = *isa-length-trail M*$_0$;
pos $\leftarrow$ *get-pos-of-level-in-trail-imp M*$_0$ *lev*;
ASSERT((*n* − *pos*) $\leq$ *unat32-max*);
ASSERT(*n* $\geq$ *pos*);
let target = *n* − *pos*;
(-, *M*, *vm*′) $\leftarrow$

```
    WHILE_T^λ(j, M, vm'). j ≤ target
     (λ(j, M, vm). j < target)
     (λ(j, M, vm). do {
       ASSERT(Suc j ≤ unat32-max);
       ASSERT(case M of (M, -) ⇒ M ≠ []);
       ASSERT(tl-trailt-tr-no-CS-pre M);
       let L = atm-of (lit-of-last-trail-pol M);
       ASSERT(isa-bump-unset-pre L vm);
       vm ← isa-bump-unset L vm;
       RETURN (j + 1, tl-trailt-tr-no-CS M, vm)
     })
     (0, M_0, vm);
   M ← trail-conv-back-imp lev M;
   RETURN (M, vm')
  })⟩

abbreviation find-decomp-w-ns-prop where
  ⟨find-decomp-w-ns-prop A ≡
    (λ(M::(nat, nat) ann-lits) highest -.
      (λ(M1, vm). ∃K M2. (Decided K # M1, M2) ∈ set (get-all-ann-decomposition M) ∧
        get-level M K = Suc highest ∧ vm ∈ bump-heur A M1))⟩

definition find-decomp-w-ns where
  ⟨find-decomp-w-ns A =
    (λ(M::(nat, nat) ann-lits) highest vm.
      SPEC(find-decomp-w-ns-prop A M highest vm))⟩

lemma isa-find-decomp-wl-imp-find-decomp-wl-imp:
  ⟨(uncurry2 isa-find-decomp-wl-imp, uncurry2 (find-decomp-wl-imp A)) ∈
    [λ((M, lev), vm). lev < count-decided M]_f trail-pol A ×_f nat-rel ×_f Id →
    ⟨trail-pol A ×_r (Id)⟩nres-rel⟩
proof −
  have [intro]: ⟨(M', M) ∈ trail-pol A ⟹ (M', M) ∈ trail-pol-no-CS A⟩ for M' M
    by (auto simp: trail-pol-def trail-pol-no-CS-def control-stack-length-count-dec[symmetric])

  have loop-init[refine0]: ⟨⋀x y x1 x1a x2 x2a x1b x1c x2b x2c pos posa.
    case y of (x, xa) ⇒ (case x of (M, lev) ⇒ λvm. lev < count-decided M) xa ⟹
    (x, y) ∈ trail-pol A ×_f nat-rel ×_f Id ⟹
    x1 = (x1a, x2) ⟹
    y = (x1, x2a) ⟹
    x1b = (x1c, x2b) ⟹
    x = (x1b, x2c) ⟹
    isa-length-trail-pre (trail-pol-conv-to-no-CS x1c) ⟹
    (pos, posa) ∈ nat-rel ⟹
    length (trail-conv-to-no-CS x1a) − posa ≤ unat32-max ⟹
    posa ≤ length (trail-conv-to-no-CS x1a) ⟹
    isa-length-trail (trail-pol-conv-to-no-CS x1c) − pos ≤ unat32-max ⟹
    pos ≤ isa-length-trail (trail-pol-conv-to-no-CS x1c) ⟹
    case (0, trail-conv-to-no-CS x1a, x2a) of
    (j, M, vm') ⇒
      j ≤ length (trail-conv-to-no-CS x1a) − posa ∧
      M = drop j (trail-conv-to-no-CS x1a) ∧
      length (trail-conv-to-no-CS x1a) − posa
      ≤ length (trail-conv-to-no-CS x1a) ∧
      vm' ∈ bump-heur A M ∧ literals-are-in-L_in A (lit-of `# mset M) ⟹
```

((0, *trail-pol-conv-to-no-CS x1c*, *x2c*), 0, *trail-conv-to-no-CS x1a*, *x2a*)
∈ *nat-rel* ×$_r$ *trail-pol-no-CS* $\mathcal{A}$ ×$_r$ *Id*›
supply *trail-pol-conv-to-no-CS-def*[*simp*] *trail-conv-to-no-CS-def*[*simp*]
by *auto*
have *trail-pol-empty*: ‹(([], *x2g*), *M*) ∈ *trail-pol-no-CS* $\mathcal{A}$ ⟹ *M* = []› **for** *M x2g*
by (*auto simp*: *trail-pol-no-CS-def ann-lits-split-reasons-def*)

have *trail-pol-no-CS-last-hd*:
‹((*x1h*, *t*), *M*) ∈ *trail-pol-no-CS* $\mathcal{A}$ ⟹ *M* ≠ [] ⟹ (*last x1h*) = *lit-of* (*hd M*)›
for *x1h t M*
by (*auto simp*: *trail-pol-no-CS-def ann-lits-split-reasons-def last-map last-rev*)

have *trail-conv-back*: ‹*trail-conv-back-imp x2b x1g*
≤ *SPEC*
(λ*c*. (*c*, *trail-conv-back x2 x1e*)
∈ *trail-pol* $\mathcal{A}$)›
if
‹*case y of* (*x*, *xa*) ⇒ (*case x of* (*M*, *lev*) ⇒ λ*vm*. *lev* < *count-decided M*) *xa*› **and**
‹(*x*, *y*) ∈ *trail-pol* $\mathcal{A}$ ×$_f$ *nat-rel* ×$_f$ *Id*› **and**
‹*x1* = (*x1a*, *x2*)› **and**
‹*y* = (*x1*, *x2a*)› **and**
‹*x1b* = (*x1c*, *x2b*)› **and**
‹*x* = (*x1b*, *x2c*)› **and**
‹*isa-length-trail-pre* (*trail-pol-conv-to-no-CS x1c*)› **and**
‹(*pos*, *posa*) ∈ *nat-rel*› **and**
‹*length* (*trail-conv-to-no-CS x1a*) − *posa* ≤ *unat32-max*› **and**
‹*isa-length-trail* (*trail-pol-conv-to-no-CS x1c*) − *pos* ≤ *unat32-max*› **and**
‹(*xa*, *x'*) ∈ *nat-rel* ×$_r$ *trail-pol-no-CS* $\mathcal{A}$ ×$_r$ *Id*› **and**
‹*x2d* = (*x1e*, *x2e*)› **and**
‹*x'* = (*x1d*, *x2d*)› **and**
‹*x2f* = (*x1g*, *x2g*)› **and**
‹*xa* = (*x1f*, *x2f*)› **and**
‹*x2* = *count-decided x1e*›
for *x y x1 x1a x2 x2a x1b x1c x2b x2c pos posa xa x' x1d x2d x1e x2e x1f x2f*
x1g x2g
apply (*rule trail-conv-back*[*THEN fref-to-Down-curry*, *THEN order-trans*])
using *that* **by** (*auto simp*: *conc-fun-RETURN*)

have *bump*: ‹(*a*,*a'*)∈*Id* ⟹ (*b*,*b'*)∈*Id* ⟹ *isa-bump-unset a b* ≤⇓*Id* (*isa-bump-unset a' b'*)› **for** *a a' b b'*
by *auto*
show *?thesis*
supply [*refine del*] = *refine*(*13*)
supply *trail-pol-conv-to-no-CS-def*[*simp*] *trail-conv-to-no-CS-def*[*simp*]
unfolding *isa-find-decomp-wl-imp-def find-decomp-wl-imp-def uncurry-def*
apply (*intro frefI nres-relI*)
apply (*refine-vcg bump*
id-trail-conv-to-no-CS[*THEN fref-to-Down*, *unfolded comp-def*]
get-pos-of-level-in-trail-imp-get-pos-of-level-in-trail[*of* $\mathcal{A}$, *THEN fref-to-Down-curry*])
subgoal
by (*rule isa-length-trail-pre*) *auto*
subgoal
by (*auto simp*: *get-pos-of-level-in-trail-pre-def*)
subgoal
by *auto*

```
    subgoal
      by (subst isa-length-trail-length-u-no-CS[THEN fref-to-Down-unRET-Id]) auto
    subgoal
      by (subst isa-length-trail-length-u-no-CS[THEN fref-to-Down-unRET-Id]) auto
      apply (assumption+)[10]
    subgoal
      by (subst isa-length-trail-length-u-no-CS[THEN fref-to-Down-unRET-Id]) auto
    subgoal
      by auto
    subgoal
      by (subst isa-length-trail-length-u-no-CS[THEN fref-to-Down-unRET-Id]) auto
    subgoal
      by (subst isa-length-trail-length-u-no-CS[THEN fref-to-Down-unRET-Id]) auto
    subgoal
      by auto
    subgoal
      by (auto dest!: trail-pol-empty)
    subgoal for x y x1 x1a x2 x2a x1b x1c x2b x2c pos posa
      by (rule tl-trailt-tr-no-CS-pre) auto
    subgoal for x y x1 x1a x2 x2a x1b x1c x2b x2c pos posa xa x′ x1d x2d x1e x2e x1f x2f
        x1g x1h x2g x2h
        by (cases x1g, cases x2h)
          (auto intro!: isa-bump-unset-pre[of - 𝒜 ⟨drop x1d x1a⟩]
          simp: lit-of-last-trail-pol-def trail-pol-no-CS-last-hd lit-of-hd-trail-def)
    subgoal
      by (auto simp: lit-of-last-trail-pol-def trail-pol-no-CS-last-hd lit-of-hd-trail-def
        intro!: tl-trail-tr-no-CS[THEN fref-to-Down-unRET]
          isa-vmtf-unset-vmtf-unset[THEN fref-to-Down-unRET-uncurry])
    subgoal
      by (auto simp: lit-of-last-trail-pol-def trail-pol-no-CS-last-hd lit-of-hd-trail-def
        intro!: tl-trail-tr-no-CS[THEN fref-to-Down-unRET]
          isa-vmtf-unset-vmtf-unset[THEN fref-to-Down-unRET-uncurry])
    subgoal
      by (auto simp: lit-of-last-trail-pol-def trail-pol-no-CS-last-hd lit-of-hd-trail-def
        intro!: tl-trail-tr-no-CS[THEN fref-to-Down-unRET]
          isa-vmtf-unset-vmtf-unset[THEN fref-to-Down-unRET-uncurry])
    apply (rule trail-conv-back; assumption)
    subgoal
      by auto
  done
qed


definition (in −) find-decomp-wl-st :: ⟨nat literal ⇒ nat twl-st-wl ⇒ nat twl-st-wl nres⟩ where
  ⟨find-decomp-wl-st = (λL (M, N, D, oth).  do{
     M′ ← find-decomp-wl′ M (the D) L;
    RETURN (M′, N, D, oth)
  })⟩

definition find-decomp-wl-st-int :: ⟨nat ⇒ isasat ⇒ isasat nres⟩ where
  ⟨find-decomp-wl-st-int = (λhighest S. do{
     let M = get-trail-wl-heur S;
     let vm = get-vmtf-heur S;
     (M′, vm) ← isa-find-decomp-wl-imp M highest vm;
     let S = set-trail-wl-heur M′ S;
     let S = set-vmtf-wl-heur vm S;
```

```
    RETURN S
  })›

lemma
  assumes
    vm: ‹vm ∈ bump-heur 𝒜 M₀› and
    lits: ‹literals-are-in-ℒ_in-trail 𝒜 M₀› and
    target: ‹highest < count-decided M₀› and
    n-d: ‹no-dup M₀› and
    bounded: ‹isasat-input-bounded 𝒜› and
    count:‹count-decided M₀ > 0›
  shows
    find-decomp-wl-imp-le-find-decomp-wl':
      ‹find-decomp-wl-imp 𝒜 M₀ highest vm ≤ find-decomp-w-ns 𝒜 M₀ highest vm›
     (is ?decomp)
proof −
  have length-M0:  ‹length M₀ ≤ unat32-max div 2 + 1›
    using length-trail-unat32-max-div2[of 𝒜 M₀, OF bounded]
      n-d literals-are-in-ℒ_in-trail-in-lits-of-l[of 𝒜, OF lits]
    by (auto simp: lits-of-def)
  have 1: ‹((count-decided x1g, x1g), count-decided x1, x1) ∈ Id›
    if ‹x1g = x1› for x1g x1 :: ‹(nat, nat) ann-lits›
    using that by auto
  have [simp]: ‹∃ M'a. M' @ x2g = M'a @ tl x2g› for M' x2g :: ‹(nat, nat) ann-lits›
    by (rule exI[of - ‹M' @ (if x2g = [] then [] else [hd x2g])›]) auto
  have butlast-nil-iff: ‹butlast xs = [] ⟷ xs = [] ∨ (∃ a. xs = [a])› for xs :: ‹(nat, nat) ann-lits›
    by (cases xs) auto
  have butlast1: ‹tl x2g = drop (Suc (length x1) − length x2g) x1› (is ‹?G1›)
    if ‹x2g = drop (length x1 − length x2g) x1› for x2g x1 :: ‹'a list›
  proof −
    have [simp]: ‹Suc (length x1 − length x2g) = Suc (length x1) − length x2g›
      by (metis Suc-diff-le diff-le-mono2 diff-zero length-drop that zero-le)
    show ?G1
      by (subst that) (auto simp: butlast-conv-take tl-drop-def)[]
  qed
  have butlast2: ‹tl x2g = drop (length x1 − (length x2g − Suc 0)) x1› (is ‹?G2›)
    if ‹x2g = drop (length x1 − length x2g) x1› and x2g: ‹x2g ≠ []› for x2g x1 :: ‹'a list›
  proof −
    have [simp]: ‹Suc (length x1 − length x2g) = Suc (length x1) − length x2g›
      by (metis Suc-diff-le diff-le-mono2 diff-zero length-drop that(1) zero-le)
    have [simp]: ‹Suc (length x1) − length x2g = length x1 − (length x2g − Suc 0)›
      using x2g by auto
    show ?G2
      by (subst that) (auto simp: butlast-conv-take tl-drop-def)[]
  qed
  note butlast = butlast1 butlast2

  have count-decided-not-Nil[simp]:  ‹0 < count-decided M ⟹ M ≠ []› for M :: ‹(nat, nat) ann-lits›
    by auto
  have get-lev-last: ‹get-level (M' @ M) (lit-of (last M')) = Suc (count-decided M)›
    if ‹M₀ = M' @ M› and ‹M' ≠ []› and ‹is-decided (last M')› for M' M
    apply (cases M' rule: rev-cases)
    using that apply (solves simp)
    using n-d that by auto

  have atm-of-N:
```

‹*literals-are-in-$\mathcal{L}_{in}$ $\mathcal{A}$ (lit-of `# mset aa) $\Longrightarrow$ aa $\neq$ [] $\Longrightarrow$ atm-of (lit-of (hd aa)) $\in$ atms-of ($\mathcal{L}_{all}$ $\mathcal{A}$)*›
for *aa*
by (*cases aa*) (*auto simp*: *literals-are-in-$\mathcal{L}_{in}$-add-mset in-$\mathcal{L}_{all}$-atm-of-in-atms-of-iff*)
have *Lin-drop-tl*: ‹*literals-are-in-$\mathcal{L}_{in}$ $\mathcal{A}$ (lit-of `# mset (drop b M_0)) $\Longrightarrow$*
literals-are-in-$\mathcal{L}_{in}$ $\mathcal{A}$ (lit-of `# mset (tl (drop b M_0)))› **for** *b*
apply (*rule literals-are-in-$\mathcal{L}_{in}$-mono*)
apply *assumption*
by (*cases ‹drop b M_0›*) *auto*

have *highest*: ‹*highest = count-decided M*› **and**
ex-decomp: ‹$\exists$ *K M2*.
(*Decided K # M*, *M2*)
$\in$ *set* (*get-all-ann-decomposition* M_0) $\wedge$
get-level M_0 *K = Suc highest* $\wedge$ *vm* $\in$ *bump-heur* $\mathcal{A}$ *M*›
if
pos: ‹*pos < length* M_0 $\wedge$ *is-decided* (*rev* M_0 ! *pos*) $\wedge$ *get-level* M_0 (*lit-of* (*rev* M_0 ! *pos*)) =
highest + 1› **and**
‹*length* M_0 − *pos* $\leq$ *unat32-max*› **and**
inv: ‹*case s of* (*j*, *M*, *vm'*) $\Rightarrow$
j $\leq$ *length* M_0 − *pos* $\wedge$
M = drop j M_0 $\wedge$
length M_0 − *pos* $\leq$ *length* M_0 $\wedge$
vm' $\in$ *bump-heur* $\mathcal{A}$ *M* $\wedge$
literals-are-in-$\mathcal{L}_{in}$ $\mathcal{A}$ (*lit-of `# mset M*)› **and**
cond: ‹$\neg$ (*case s of*
(*j*, *M*, *vm*) $\Rightarrow$ *j < length* M_0 − *pos*)› **and**
s: ‹*s* = (*j*, *s'*)› ‹*s'* = (*M*, *vm*)›
for *pos s j s' M vm*
proof −
have
‹*j = length* M_0 − *pos*› **and**
M: ‹*M = drop* (*length* M_0 − *pos*) M_0› **and**
vm: ‹*vm* $\in$ *bump-heur* $\mathcal{A}$ (*drop* (*length* M_0 − *pos*) M_0)› **and**
‹*literals-are-in-$\mathcal{L}_{in}$* $\mathcal{A}$ (*lit-of `# mset* (*drop* (*length* M_0 − *pos*) M_0))›
using *cond inv* **unfolding** *s*
by *auto*
define *M2* **and** *L* **where** ‹*M2 = take* (*length* M_0 − *Suc pos*) M_0› **and** ‹*L = rev* M_0 ! *pos*›
have *le-Suc-pos*: ‹*length* M_0 − *pos = Suc* (*length* M_0 − *Suc pos*)›
using *pos* **by** *auto*
have *1*: ‹*take* (*length* M_0 − *pos*) M_0 = *take* (*length* M_0 − *Suc pos*) M_0 @ [*rev* M_0 ! *pos*]›
unfolding *le-Suc-pos*
apply (*subst take-Suc-conv-app-nth*)
using *pos* **by** (*auto simp*: *rev-nth*)
have M_0: ‹M_0 = *M2* @ *L* # *M*›
apply (*subst append-take-drop-id*[*symmetric*, *of* - ‹*length* M_0 − *pos*›])
unfolding *M L-def M2-def 1*
by *auto*
have *L'*: ‹*Decided* (*lit-of L*) = *L*›
using *pos* **unfolding** *L-def*[*symmetric*] **by** (*cases L*) *auto*
then have M_0': ‹M_0 = *M2* @ *Decided* (*lit-of L*) # *M*›
unfolding M_0 **by** *auto*

have ‹*highest = count-decided M*› **and** ‹*get-level* M_0 (*lit-of L*) = *Suc highest*› **and** ‹*is-decided L*›
using *n-d pos* **unfolding** *L-def*[*symmetric*] **unfolding** M_0
by (*auto simp*: *get-level-append-if split*: *if-splits*)

```
    then show
     ‹∃ K M2.
       (Decided K # M, M2)
       ∈ set (get-all-ann-decomposition M₀) ∧
       get-level M₀ K = Suc highest ∧ vm ∈ bump-heur 𝒜 M›
     using get-all-ann-decomposition-ex[of ‹lit-of L› M M2] vm unfolding M₀'[symmetric] M[symmetric]
      by blast
    show ‹highest = count-decided M›
      using ‹highest = count-decided M› .
  qed
  have count-dec-larger: ‹highest < count-decided M›
    if
      pos: ‹pos < length M₀ ∧ is-decided (rev M₀ ! pos) ∧ get-level M₀ (lit-of (rev M₀ ! pos)) =
        highest + 1› and
      ‹length M₀ − pos ≤ unat32-max› and
      inv: ‹case s of (j, M, vm') ⇒
        j ≤ length M₀ − pos ∧
        M = drop j M₀ ∧
        length M₀ − pos ≤ length M₀ ∧
        vm' ∈ bump-heur 𝒜 M ∧
        literals-are-in-ℒ_in 𝒜 (lit-of `# mset M)› and
      cond: ‹(case s of
        (j, M, vm) ⇒ j < length M₀ − pos)› and
      s: ‹s = (j, s')› ‹s' = (M, vm)›
    for pos s j s' M vm
  proof −
    define M2 and L where ‹M2 = take (length M₀ − Suc pos) M₀› and ‹L = rev M₀ ! pos›
    have le-Suc-pos: ‹length M₀ − pos = Suc (length M₀ − Suc pos)›
      using pos by auto
    have 1: ‹take (length M₀ − pos) M₀ = take (length M₀ − Suc pos) M₀ @ [rev M₀ ! pos]›
      unfolding le-Suc-pos
      apply (subst take-Suc-conv-app-nth)
      using pos by (auto simp: rev-nth)
    have ‹L ∈ set M›
      using that inv L-def apply auto
       by (metis bot-nat-0.not-eq-extremum diff-Suc-less in-set-dropI le-Suc-pos minus-eq nat.simps(3)
nat-Suc-less-le-imp rev-nth)
    moreover have ‹count-decided M ≥ get-level M (lit-of L)›
      using count-decided-ge-get-level by blast
    moreover have ‹get-level M₀ (lit-of L) = Suc highest› and ‹is-decided L›
      using n-d pos unfolding L-def[symmetric]
      by (auto simp: get-level-append-if split: if-splits)
    moreover have ‹get-level M₀ (lit-of L) = get-level M (lit-of L)›
      using n-d
     apply (subst append-take-drop-id[symmetric, of M₀ j], subst (asm)append-take-drop-id[symmetric,
of M₀ j])
      using that ‹L ∈ set M› apply (auto simp del: append-take-drop-id simp: get-level-append-if
        dest: defined-lit-no-dupD undefined-notin)
      using defined-lit-no-dupD(1) undefined-notin by blast
    ultimately show ?thesis
      by auto
  qed
find-theorems isa-bump-unset bump-heur
  show ?decomp
    unfolding find-decomp-wl-imp-def Let-def find-decomp-w-ns-def trail-conv-to-no-CS-def
      get-pos-of-level-in-trail-def trail-conv-back-def
```

apply (*refine-vcg 1 WHILEIT-rule*[**where** *R*=‹*measure* (λ(-, *M*, -). *length M*)›]
isa-bump-unset-vmtf-tl[*unfolded lit-of-hd-trail-def*[*symmetric*], *of* - $\mathcal{A}$, *THEN order-trans*])
subgoal using *length-M0* **unfolding** *unat32-max-def* **by** *simp*
subgoal by *auto*
subgoal by *auto*
subgoal using *target* **by** (*auto simp*: *count-decided-ge-get-maximum-level*)
subgoal by *auto*
subgoal by *auto*
subgoal using *vm* **by** *auto*
subgoal using *lits* **unfolding** *literals-are-in-*$\mathcal{L}_{in}$*-trail-lit-of-mset* **by** *auto*
subgoal by (*rule count-dec-larger*)
subgoal for *target s j b M vm* **by** *simp*
subgoal using *length-M0* **unfolding** *unat32-max-def* **by** *simp*
subgoal for *x s a ab aa bb*
by (*cases* ‹*drop a* M_0›)
(*auto simp*: *lit-of-hd-trail-def literals-are-in-*$\mathcal{L}_{in}$*-add-mset*)
subgoal by *auto*
subgoal by (*auto simp*: *drop-Suc drop-tl atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$)
subgoal by *auto*
subgoal for *s a b aa ba vm x2 x1a x2a*
using *target*
by (*cases vm*)
(*auto intro*!: *isa-bump-unset-vmtf-tl atm-of-N drop-tl simp*: *lit-of-hd-trail-def*)
subgoal for *s a b aa ba x1 x2 x1a x2a*
using *lits* **by** (*auto intro*: *Lin-drop-tl simp*: *drop-Suc tl-drop*)
subgoal by *auto*
subgoal by *auto*
subgoal for *s a b aa ba x1 x2 x1a x2a*
using *lits* **by** (*auto intro*: *Lin-drop-tl*)
subgoal by *auto*
subgoal by (*rule highest*)
subgoal by (*rule ex-decomp*) (*assumption*+, *auto*)
done
qed

lemma *find-decomp-wl-imp-find-decomp-wl′*:
‹(*uncurry2* (*find-decomp-wl-imp* $\mathcal{A}$), *uncurry2* (*find-decomp-w-ns* $\mathcal{A}$)) ∈
[*find-decomp-w-ns-pre* $\mathcal{A}$]$_f$ *Id* ×$_f$ *Id* ×$_f$ *Id* → ⟨*Id* ×$_f$ *Id*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *find-decomp-w-ns-pre-def simp del*: *twl-st-of-wl.simps*
intro!: *find-decomp-wl-imp-le-find-decomp-wl′*)

lemma *find-decomp-wl-imp-code-conbine-cond*:
‹(λ((*b*, *a*), *c*). *find-decomp-w-ns-pre* $\mathcal{A}$ ((*b*, *a*), *c*) ∧ *a* < *count-decided b*) = (λ((*b*, *a*), *c*).
find-decomp-w-ns-pre $\mathcal{A}$ ((*b*, *a*), *c*))›
by (*auto intro*!: *ext simp*: *find-decomp-w-ns-pre-def*)

end
theory *IsaSAT-VMTF-LLVM*
imports *Watched-Literals.WB-Sort IsaSAT-VMTF*
IsaSAT-VMTF-Setup-LLVM
Examples.Sorting-Introsort
IsaSAT-Sorting-LLVM
IsaSAT-Literals-LLVM
IsaSAT-Trail-LLVM

IsaSAT-Clauses-LLVM
IsaSAT-Lookup-Conflict-LLVM
begin
hide-const (**open**) *NEMonad.RETURN NEMonad.ASSERT*

definition *valid-atoms* :: ‹*nat-vmtf-node list* ⇒ *nat set*› **where**
‹*valid-atoms xs* ≡ {*i*. *i* < *length xs*}›

definition *VMTF-score-less* **where**
‹ *VMTF-score-less xs i j* ⟷ *stamp* (*xs* ! *i*) < *stamp* (*xs* ! *j*)›

definition *mop-VMTF-score-less* **where**
‹*mop-VMTF-score-less xs i j* = *do* {
ASSERT(*i* < *length xs*);
ASSERT(*j* < *length xs*);
RETURN (*stamp* (*xs* ! *i*) < *stamp* (*xs* ! *j*))
}›

sepref-register *VMTF-score-less*

sepref-def (**in** −) *mop-VMTF-score-less-impl*
is ‹*uncurry2* (*mop-VMTF-score-less*)›
:: ‹(*array-assn vmtf-node-assn*)^k *∗$_a$ atom-assn*^{k} *∗$_a$ atom-assn*^{k} →$_a$ *bool1-assn*›
supply [[*goals-limit* = *1*]]
unfolding *mop-VMTF-score-less-def*
apply (*rewrite at* ‹*stamp* (- ! ¤)› *value-of-atm-def*[*symmetric*])
apply (*rewrite at* ‹*stamp* (- ! ¤)› **in** ‹- < ¤› *value-of-atm-def*[*symmetric*])
unfolding *index-of-atm-def*[*symmetric*]
by *sepref*

interpretation *VMTF*: *weak-ordering-on-lt* **where**
C = ‹*valid-atoms vs*› **and**
less = ‹ *VMTF-score-less vs*›
by *unfold-locales*
(*auto simp*: *VMTF-score-less-def split*: *if-splits*)

interpretation *VMTF*: *parameterized-weak-ordering valid-atoms VMTF-score-less*
mop-VMTF-score-less
by *unfold-locales*
(*auto simp*: *mop-VMTF-score-less-def*
valid-atoms-def VMTF-score-less-def)

global-interpretation *VMTF*: *parameterized-sort-impl-context*
‹*woarray-assn atom-assn*› ‹*eoarray-assn atom-assn*› *atom-assn*
Mreturn Mreturn
eo-extract-impl
array-upd
valid-atoms VMTF-score-less mop-VMTF-score-less mop-VMTF-score-less-impl

‹*array-assn vmtf-node-assn*›
defines
VMTF-is-guarded-insert-impl = *VMTF.is-guarded-param-insert-impl*
and *VMTF-is-unguarded-insert-impl* = *VMTF.is-unguarded-param-insert-impl*
and *VMTF-unguarded-insertion-sort-impl* = *VMTF.unguarded-insertion-sort-param-impl*
and *VMTF-guarded-insertion-sort-impl* = *VMTF.guarded-insertion-sort-param-impl*
and *VMTF-final-insertion-sort-impl* = *VMTF.final-insertion-sort-param-impl*

and *VMTF-pcmpo-idxs-impl* = *VMTF.pcmpo-idxs-impl*
and *VMTF-pcmpo-v-idx-impl* = *VMTF.pcmpo-v-idx-impl*
and *VMTF-pcmpo-idx-v-impl* = *VMTF.pcmpo-idx-v-impl*
and *VMTF-pcmp-idxs-impl* = *VMTF.pcmp-idxs-impl*

and *VMTF-mop-geth-impl* = *VMTF.mop-geth-impl*
and *VMTF-mop-seth-impl* = *VMTF.mop-seth-impl*
and *VMTF-sift-down-impl* = *VMTF.sift-down-impl*
and *VMTF-heapify-btu-impl* = *VMTF.heapify-btu-impl*
and *VMTF-heapsort-impl* = *VMTF.heapsort-param-impl*
and *VMTF-qsp-next-l-impl* = *VMTF.qsp-next-l-impl*
and *VMTF-qsp-next-h-impl* = *VMTF.qsp-next-h-impl*
and *VMTF-qs-partition-impl* = *VMTF.qs-partition-impl*

and *VMTF-partition-pivot-impl* = *VMTF.partition-pivot-impl*
and *VMTF-introsort-aux-impl* = *VMTF.introsort-aux-param-impl*
and *VMTF-introsort-impl* = *VMTF.introsort-param-impl*
and *VMTF-move-median-to-first-impl* = *VMTF.move-median-to-first-param-impl*
apply *unfold-locales*
apply (*rule eo-hnr-dep*)+
unfolding *GEN-ALGO-def refines-param-relp-def*
supply[[*unify-trace-failure*]]
by (*rule mop-VMTF-score-less-impl.refine*)

global-interpretation
VMTF-it: *pure-eo-adapter atom-assn* ‹*arl64-assn atom-assn*› *arl-nth arl-upd*
defines *VMTF-it-eo-extract-impl* = *VMTF-it.eo-extract-impl*
apply (*rule al-pure-eo*)
by (*simp add*: *safe-constraint-rules*)

global-interpretation *VMTF-it*: *parameterized-sort-impl-context*
where
wo-assn = ‹*arl64-assn atom-assn*› **and**
eo-assn = *VMTF-it.eo-assn* **and**
elem-assn = *atom-assn* **and**
to-eo-impl = *Mreturn* **and**
to-wo-impl = *Mreturn* **and**
extract-impl = *VMTF-it-eo-extract-impl* **and**
set-impl = *arl-upd* **and**
cdom = *valid-atoms* **and**
pless = *VMTF-score-less* **and**
pcmp = *mop-VMTF-score-less* **and**
pcmp-impl = *mop-VMTF-score-less-impl* **and**

cparam-assn = ‹*array-assn vmtf-node-assn*›
defines
VMTF-it-is-guarded-insert-impl = *VMTF-it.is-guarded-param-insert-impl*
and *VMTF-it-is-unguarded-insert-impl* = *VMTF-it.is-unguarded-param-insert-impl*
and *VMTF-it-unguarded-insertion-sort-impl* = *VMTF-it.unguarded-insertion-sort-param-impl*
and *VMTF-it-guarded-insertion-sort-impl* = *VMTF-it.guarded-insertion-sort-param-impl*
and *VMTF-it-final-insertion-sort-impl* = *VMTF-it.final-insertion-sort-param-impl*

and *VMTF-it-pcmpo-idxs-impl* = *VMTF-it.pcmpo-idxs-impl*
and *VMTF-it-pcmpo-v-idx-impl* = *VMTF-it.pcmpo-v-idx-impl*
and *VMTF-it-pcmpo-idx-v-impl* = *VMTF-it.pcmpo-idx-v-impl*
and *VMTF-it-pcmp-idxs-impl* = *VMTF-it.pcmp-idxs-impl*

and *VMTF-it-mop-geth-impl* = *VMTF-it.mop-geth-impl*
and *VMTF-it-mop-seth-impl* = *VMTF-it.mop-seth-impl*
and *VMTF-it-sift-down-impl* = *VMTF-it.sift-down-impl*
and *VMTF-it-heapify-btu-impl* = *VMTF-it.heapify-btu-impl*
and *VMTF-it-heapsort-impl* = *VMTF-it.heapsort-param-impl*
and *VMTF-it-qsp-next-l-impl* = *VMTF-it.qsp-next-l-impl*
and *VMTF-it-qsp-next-h-impl* = *VMTF-it.qsp-next-h-impl*
and *VMTF-it-qs-partition-impl* = *VMTF-it.qs-partition-impl*

and *VMTF-it-partition-pivot-impl* = *VMTF-it.partition-pivot-impl*
and *VMTF-it-introsort-aux-impl* = *VMTF-it.introsort-aux-param-impl*
and *VMTF-it-introsort-impl* = *VMTF-it.introsort-param-impl*
and *VMTF-it-move-median-to-first-impl* = *VMTF-it.move-median-to-first-param-impl*

apply *unfold-locales*
unfolding *GEN-ALGO-def refines-param-relp-def*
apply (*rule mop-VMTF-score-less-impl.refine*)
done

lemmas [*llvm-inline*] = *VMTF-it.eo-extract-impl-def*[*THEN meta-fun-cong*, *THEN meta-fun-cong*]

export-llvm
‹*VMTF-heapsort-impl* :: - ⇒ - ⇒ -›
‹*VMTF-introsort-impl* :: - ⇒ - ⇒ -›

definition *VMTF-sort-scores-raw* :: ‹-› **where**
‹*VMTF-sort-scores-raw* = *pslice-sort-spec valid-atoms VMTF-score-less*›

definition *VMTF-sort-scores* :: ‹-› **where**
‹*VMTF-sort-scores xs ys* = *VMTF-sort-scores-raw xs ys 0* (*length ys*)›

lemmas *VMTF-introsort*[*sepref-fr-rules*] =
VMTF-it.introsort-param-impl-correct[*unfolded VMTF-sort-scores-raw-def*[*symmetric*] *PR-CONST-def*]

sepref-register *VMTF-sort-scores-raw vmtf-reorder-list-raw*

lemma *VMTF-sort-scores-vmtf-reorder-list-raw*:
‹(*VMTF-sort-scores*, *vmtf-reorder-list-raw*) ∈ *Id* → *Id* → ⟨*Id*⟩*nres-rel*›
unfolding *VMTF-sort-scores-def VMTF-sort-scores-raw-def pslice-sort-spec-def*
vmtf-reorder-list-raw-def
apply (*refine-rcg*)

subgoal by (*auto simp*: *valid-atoms-def*)
subgoal for *vm vm' arr arr'*
by (*auto intro*!: *slice-sort-spec-refine-sort*[*THEN order-trans*, *of - arr' arr'*]
simp: *valid-atoms-def slice-rel-def br-def reorder-list-def conc-fun-RES sort-spec-def*
eq-commute[*of* ‹*length* -› ‹*length arr'*›])
done

sepref-def *VMTF-sort-scores-raw-impl*
is ‹*uncurry VMTF-sort-scores*›
:: ‹(*IICF-Array.array-assn vmtf-node-assn*)^k $*_a$ *VMTF-it.arr-assn*^{d} $\to_a$ *VMTF-it.arr-assn*›
unfolding *VMTF-sort-scores-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemmas[*sepref-fr-rules*] =
VMTF-sort-scores-raw-impl.refine[*FCOMP VMTF-sort-scores-vmtf-reorder-list-raw*]

sepref-def *VMTF-sort-scores-impl*
is ‹*uncurry vmtf-reorder-list*›
:: ‹(*vmtf-assn*)^k $*_a$ *VMTF-it.arr-assn*^{d} $\to_a$ *VMTF-it.arr-assn*›
unfolding *vmtf-reorder-list-def vmtf-assn-def*
by *sepref*

sepref-def *atoms-hash-del-code*
is ‹*uncurry* (*RETURN oo atoms-hash-del*)›
:: ‹[*uncurry atoms-hash-del-pre*]$_a$ *atom-assn*^{k} $*_a$ (*atoms-hash-assn*)^d $\to$ *atoms-hash-assn*›
unfolding *atoms-hash-del-def atoms-hash-del-pre-def*
apply *annot-all-atm-idxs*
by *sepref*

sepref-def *atoms-hash-insert-code*
is ‹*uncurry* (*RETURN oo atoms-hash-insert*)›
:: ‹[*uncurry atms-hash-insert-pre*]$_a$
atom-assn^k $*_a$ (*distinct-atoms-assn*)^d $\to$ *distinct-atoms-assn*›
unfolding *atoms-hash-insert-def atms-hash-insert-pre-def*
supply [[*goals-limit=1*]]
apply *annot-all-atm-idxs*
by *sepref*

sepref-register *find-decomp-wl-imp*
sepref-register *rescore-clause vmtf-flush*

sepref-register *get-the-propagation-reason-pol*

sepref-def *update-next-search-impl*
is ‹*uncurry* (*RETURN oo update-next-search*)›
:: ‹(*atom.option-assn*)^k $*_a$ *vmtf-assn*^{d} $\to_a$ *vmtf-assn*›
supply [[*goals-limit=1*]]
unfolding *update-next-search-def vmtf-assn-def*
by *sepref*

lemma *case-option-split*:
‹(*case a of None* $\Rightarrow$ *x* | *Some y* $\Rightarrow$ *f y*) =

```
   (if is-None a then x else let y = the a in f y)›
  by (auto split: option.splits)

sepref-def ns-vmtf-dequeue-code
  is ‹uncurry (RETURN oo ns-vmtf-dequeue)›
  :: ‹[vmtf-dequeue-pre]_a
      atom-assn^k *_a (array-assn vmtf-node-assn)^d → array-assn vmtf-node-assn›
  supply [[goals-limit = 1]]
  supply option.splits[split] if-splits[split]
  unfolding ns-vmtf-dequeue-def vmtf-dequeue-pre-alt-def case-option-split atom.fold-option
  apply annot-all-atm-idxs
  by sepref


sepref-register get-next get-prev stamp
lemma eq-Some-iff: ‹x = Some b ⟷ (¬is-None x ∧ the x = b)›
  by (cases x) auto

lemma hfref-refine-with-pre:
  assumes ‹⋀x. P x ⟹ g' x ≤ g x›
  assumes ‹(f,g') ∈ [P]_a A → R›
  shows ‹(f,g) ∈ [P]_a A → R›
  using assms(2)[THEN hfrefD] assms(1)
  by (auto intro!: hfrefI intro: hn-refine-ref)

lemma isa-vmtf-en-dequeue-preI:
  assumes ‹isa-vmtf-en-dequeue-pre ((M,L),(ns, m, fst-As, lst-As, next-search))›
  shows ‹fst-As < length ns› ‹L < length ns› ‹Suc m < max-unat 64›
    and ‹get-next (ns!L) = Some i ⟶ i < length ns›
    and ‹fst-As ≠ lst-As ⟶ get-prev (ns ! lst-As) ≠ None›
    and ‹get-next (ns ! fst-As) ≠ None ⟶ get-prev (ns ! lst-As) ≠ None›
  using assms
  unfolding isa-vmtf-en-dequeue-pre-def vmtf-dequeue-pre-def
  apply (auto simp: max-unat-def unat64-max-def snat64-max-def)
  done

lemma isa-vmtf-en-dequeue-alt-def2:
   ‹isa-vmtf-en-dequeue-pre x ⟹ uncurry2 (λM L vm.
    case vm of (ns, m, fst-As, lst-As, next-search) ⇒ doN {
      ASSERT(L<length ns);
      nsL ← mop-list-get ns (index-of-atm L);
      let fst-As = (if fst-As = L then get-next nsL else (Some fst-As));

      let next-search = (if next-search = (Some L) then get-next nsL
                         else next-search);
      let lst-As = (if lst-As = L then get-prev nsL else (Some lst-As));
      ASSERT (vmtf-dequeue-pre (L,ns));
      let ns = ns-vmtf-dequeue L ns;
      ASSERT (defined-atm-pol-pre M L);
      let de = (defined-atm-pol M L);
      ASSERT (Suc m < max-unat 64);
      case fst-As of
        None ⇒ RETURN
          (ns[L := VMTF-Node m fst-As None], m + 1, L, L,
           if de then None else Some L)
      | Some fst-As ⇒ doN {
```

```
        ASSERT (L < length ns ∧ fst-As < length ns ∧ lst-As ≠ None);
        let fst-As' =
              VMTF-Node (stamp (ns ! fst-As)) (Some L)
              (get-next (ns ! fst-As));
        RETURN (
          ns[L := VMTF-Node (m + 1) None (Some fst-As),
          fst-As := fst-As'],
          m + 1, L, the lst-As,
          if de then next-search else Some L)
      }
    }) x
  ≤ uncurry2 (isa-vmtf-en-dequeue) x
    ›
  unfolding isa-vmtf-en-dequeue-def vmtf-dequeue-def isa-vmtf-enqueue-def
    annot-unat-snat-upcast[symmetric] ASSN-ANNOT-def
  apply (cases x; simp add: Let-def)
  apply (simp
    only: pw-le-iff refine-pw-simps
    split: prod.splits
    )
  supply isa-vmtf-en-dequeue-preD[simp]
  apply (auto
    split!: if-splits option.splits
    simp: refine-pw-simps isa-vmtf-en-dequeue-preI dest: isa-vmtf-en-dequeue-preI
    simp del: not-None-eq
    )
  done


sepref-register 1 0



lemma vmtf-en-dequeue-fast-codeI:
  assumes ‹isa-vmtf-en-dequeue-pre ((M, L),(ns,m,fst-As, lst-As, next-search))›
  shows ‹Suc m < max-unat 64›
  using assms
  unfolding isa-vmtf-en-dequeue-pre-def max-unat-def unat64-max-def
  by auto

sepref-def vmtf-en-dequeue-fast-code
   is ‹uncurry2 isa-vmtf-en-dequeue›
   :: ‹[isa-vmtf-en-dequeue-pre]_a
        trail-pol-fast-assn^k *_a atom-assn^k *_a vmtf-assn^d → vmtf-assn›
  apply (rule hfref-refine-with-pre[OF isa-vmtf-en-dequeue-alt-def2], assumption)

  supply [[goals-limit = 1]]
  unfolding isa-vmtf-en-dequeue-alt-def2 case-option-split eq-Some-iff  vmtf-assn-def
  apply (rewrite in ‹if ¤ then get-next - else -› short-circuit-conv)
  apply annot-all-atm-idxs
  apply (annot-unat-const ‹TYPE(64)›)
  unfolding atom.fold-option
  unfolding fold-tuple-optimizations
  by sepref
```

sepref-register *vmtf-rescale*
sepref-def *vmtf-rescale-code*
  is ‹*vmtf-rescale*›
  :: ‹*vmtf-assn*^{d} →$_a$ *vmtf-assn*›
  supply [[*goals-limit* = *1*]]
  supply *vmtf-en-dequeue-pre-def*[*simp*]
  unfolding *vmtf-rescale-alt-def update-stamp.simps vmtf-assn-def*
  unfolding *atom.fold-option*
  apply (*annot-unat-const* ‹*TYPE*(*64*)›)
  apply *annot-all-atm-idxs*
  by *sepref*

sepref-register *partition-between-ref*

sepref-register *isa-vmtf-enqueue*

lemma *emptied-list-alt-def*: ‹*emptied-list xs* = *take 0 xs*›
  by (*auto simp*: *emptied-list-def*)

sepref-def *current-stamp-impl*
  is ‹*RETURN o current-stamp*›
  :: ‹*vmtf-assn*^{k} →$_a$ *uint64-nat-assn*›
  unfolding *current-stamp-alt-def vmtf-assn-def*
  by *sepref*

sepref-register *isa-vmtf-en-dequeue*

sepref-def *isa-vmtf-flush-fast-code*
  is ‹*uncurry2 isa-vmtf-flush-int*›
  :: ‹*trail-pol-fast-assn*^{k} *$*_a$ (*vmtf-assn*)^d *$*_a$ *distinct-atoms-assn*^{d} →$_a$ *vmtf-assn* ×$_a$ *distinct-atoms-assn*›
  supply [[*goals-limit* = *1*]]
  unfolding *vmtf-flush-def PR-CONST-def isa-vmtf-flush-int-def*
    current-stamp-def[*symmetric*] *emptied-list-alt-def*
  apply (*rewrite at* ‹*If* (- − - ≤ ⌑) - -› *annot-snat-unat-conv*)
  apply (*rewrite at* ‹*WHILEIT* - (λ(-, -, -).- < ⌑)› *annot-snat-unat-conv*)
  apply (*rewrite at* ‹*isa-vmtf-en-dequeue* - (- ! ⌑)› *annot-unat-snat-conv*)
  apply (*rewrite at* ‹*atoms-hash-del* (- ! ⌑)› *annot-unat-snat-conv*)
  apply (*rewrite at* ‹*take* ⌑ -› *snat-const-fold*[**where** *'a=64*])
  apply (*annot-unat-const* ‹*TYPE*(*64*)›)
  by *sepref*

sepref-register *isa-vmtf-mark-to-rescore*

sepref-register *isa-vmtf-unset*
sepref-def *isa-vmtf-unset-code*
  is ‹*uncurry* (*RETURN oo isa-vmtf-unset*)›
  :: ‹[*uncurry vmtf-unset-pre*]$_a$
    atom-assn^k *$*_a$ *vmtf-assn*^{d} → *vmtf-assn*›
  supply [[*goals-limit=1*]] *option.splits*[*split*] *vmtf-def*[*simp*] *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*[*simp*]

neq-NilE[*elim*!] *literals-are-in-*$\mathcal{L}_{in}$*-add-mset*[*simp*]
unfolding *isa-vmtf-unset-def vmtf-unset-pre-def atom.fold-option vmtf-assn-def*
apply (*rewrite* **in** ‹*If* (- ∨ -)› *short-circuit-conv*)
apply *annot-all-atm-idxs*
by *sepref*

lemma *isa-vmtf-mark-to-rescore-and-unsetI*: ‹
atms-hash-insert-pre ak (*ad*, *ba*) ⟹
isa-vmtf-mark-to-rescore-pre ak ((*a*, *aa*, *ab*, *ac*, *Some ak'*), *ad*, *ba*)›
by (*auto simp*: *isa-vmtf-mark-to-rescore-pre-def*)

lemma (**in** −) *arena-is-valid-clause-idx-le-unat64-max*:
‹*arena-is-valid-clause-idx be bd* ⟹
length be ≤ *snat64-max* ⟹
bd + *arena-length be bd* < *max-snat 64*›
‹*arena-is-valid-clause-idx be bd* ⟹ *length be* ≤ *snat64-max* ⟹
bd < *max-snat 64*›
using *arena-lifting*(*10*)[*of be - - bd*] **unfolding** *max-snat-def snat64-max-def*
by (*fastforce simp*: *arena-lifting arena-is-valid-clause-idx-def*)+

lemma *isa-vmtf-bump-to-rescore-also-reasons-clD*:
‹*arena-is-valid-clause-idx arena C* ⟹ *C* + *arena-length arena C* ≤ *length arena*›
apply (*auto simp*: *arena-is-valid-clause-idx-def arena-lifting*)
using *arena-lifting*(*10*) *arena-lifting*(*4*) **by** *auto*

schematic-goal *mk-free-vmtf-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE vmtf-assn ?fr*›
unfolding *vmtf-assn-def*
by *synthesize-free*

experiment begin

export-llvm
ns-vmtf-dequeue-code
atoms-hash-del-code
atoms-hash-insert-code
update-next-search-impl
ns-vmtf-dequeue-code
vmtf-en-dequeue-fast-code
vmtf-rescale-code
current-stamp-impl
isa-vmtf-flush-fast-code
isa-vmtf-unset-code
end

end
theory *IsaSAT-Bump-Heuristics-LLVM*
imports *IsaSAT-Bump-Heuristics*
IsaSAT-VMTF-LLVM
Tuple4-LLVM
IsaSAT-Bump-Heuristics-State-LLVM
IsaSAT-ACIDS-LLVM
begin

sepref-register *isa-acids-flush-int isa-acids-find-next-undef*

```
  acids-push-literal isa-acids-incr-score

sepref-def isa-acids-incr-score-code
  is ‹RETURN o isa-acids-incr-score›
  :: ‹acids-assn2^d →_a acids-assn2›
  unfolding isa-acids-incr-score-def acids-assn2-def
  apply (annot-unat-const ‹TYPE(64)›)
  by sepref

lemma isa-acids-flush-int-alt-def:
‹isa-acids-flush-int = (λM vm (to-remove, h). do {
    ASSERT(length to-remove ≤ unat32-max);
    let n = length to-remove;
    (-, vm, h) ← WHILE_T^λ(i, vm', h). i ≤ n
      (λ(i, vm, h). i < n)
      (λ(i, vm, h). do {
        ASSERT(i < length to-remove);
        let L = to-remove!i;
        vm ← acids-push-literal L vm;
  ASSERT(atoms-hash-del-pre L h);
        RETURN (i+1, vm, atoms-hash-del L h)})
      (0, vm, h);
    RETURN (vm, (emptied-list to-remove, h))
  })›
  unfolding isa-acids-flush-int-def Let-def
  by auto

sepref-def acids-flush-int
  is ‹uncurry2 isa-acids-flush-int›
  :: ‹trail-pol-fast-assn^k *_a acids-assn2^d *_a distinct-atoms-assn^d →_a acids-assn2 ×_a distinct-atoms-assn
›
  unfolding isa-acids-flush-int-alt-def emptied-list-alt-def
  apply (rewrite at ‹WHILEIT - (λ(-, -, -).- < ⌑)› annot-snat-unat-conv)
  apply (rewrite at ‹- ! ⌑› annot-unat-snat-conv)
  apply (rewrite at ‹take ⌑ -› snat-const-fold[where 'a=64])
  apply (annot-unat-const ‹TYPE(64)›)
  by sepref

definition acids0-mset-empty where
  ‹acids0-mset-empty = (λ(-, b, -). b= {#})›

definition hp-acids-empty where
  ‹hp-acids-empty = (λ(-, -, -, -, -, h). h = None)›

lemma hp-acids-empty:
  ‹(RETURN o hp-acids-empty, RETURN o acids0-mset-empty) ∈
   ((((⟨nat-rel⟩option-rel, ⟨nat-rel⟩option-rel⟩pairing-heaps-rel)) O
   acids-encoded-hmrel) →_f ⟨bool-rel⟩nres-rel›
proof −
  have [intro!]: ‹Me ∈# {#} ⟹ False›for Me
    by auto
   have 1: ‹(({#}, (λ-. None, λ-. None, λ-. None, λ-. None, λ-. None), None), empty-acids0) ∈
acids-encoded-hmrel›
    by (auto simp: acids-encoded-hmrel-def bottom-acids0-def pairing-heaps-rel-def map-fun-rel-def
     ACIDS.hmrel-def encoded-hp-prop-list-conc-def encoded-hp-prop-def empty-outside-def empty-acids0-def
      intro!: relcompI)
```

have *H*: ‹*mset-nodes ya* ≠ {#}› **for** *ya*
by (*cases ya*) *auto*
show *?thesis*
unfolding *uncurry0-def*
by (*intro frefI nres-relI*)
(*auto simp add*: *acids-encoded-hmrel-def acids0-mset-empty-def encoded-hp-prop-def ACIDS.hmrel-def encoded-hp-prop-list-conc-def hp-acids-empty-def pairing-heaps-rel-def H*
split: *option.splits*)
qed

definition *acids-mset-empty* :: ‹-› **where**
‹*acids-mset-empty x* = (*acids-mset x* = {#})›

lemma *acids-mset-empty-alt-def*:
‹*acids-mset-empty* = (λ(*a*, *b*). *acids0-mset-empty a*)›
by (*auto intro*!: *ext simp*: *acids-mset-empty-def acids0-mset-empty-def*
acids-mset-def)

sepref-def *hp-acids-empty-code*
is ‹*RETURN o hp-acids-empty*›
:: ‹*hp-assn*^{k} →$_a$ *bool1-assn*›
unfolding *hp-assn-def hp-acids-empty-def atom.fold-option*
by *sepref*

lemmas [*fcomp-norm-unfold*] = *acids-assn-def*[*symmetric*]

lemmas [*sepref-fr-rules*] = *hp-acids-empty-code.refine*[*FCOMP hp-acids-empty*,
unfolded hr-comp-assoc[*symmetric*] *acids-assn-def*[*symmetric*] *acids-assn2-def*[*symmetric*]]

sepref-def *acids-mset-empty-code*
is ‹*RETURN o acids-mset-empty*›
:: ‹*acids-assn2*^{k} →$_a$ *bool1-assn*›
unfolding *acids-mset-empty-alt-def acids-assn2-def*
by *sepref*

sepref-def *acids-find-next-undef-impl*
is ‹*uncurry isa-acids-find-next-undef*›
:: ‹*acids-assn2*^{d} *$_a$ *trail-pol-fast-assn*^{k} →$_a$ *atom.option-assn* ×$_a$ *acids-assn2*›
unfolding *isa-acids-find-next-undef-def*
atom.fold-option acids-mset-empty-def[*symmetric*]
by *sepref*

lemma *isa-bump-unset-alt-def*:
‹*isa-bump-unset L vm* = (*case vm of Tuple4* (*hstable*) (*focused*) *foc a* ⇒ *do* {
hstable ← (*if* ¬*foc then acids-tl L hstable else RETURN hstable*);
let focused = (*if foc then isa-vmtf-unset L focused else focused*);
RETURN (*Tuple4 hstable focused foc a*)
})›
unfolding *isa-bump-unset-def isa-vmtf-unset-def vmtf-unset-def*[*symmetric*]
by (*auto intro*!: *ext split*: *bump-heuristics-splits*)

sepref-register *vmtf-unset case-tuple4*
sepref-def *isa-bump-unset-impl*
  is ‹*uncurry* (*isa-bump-unset*)›
  :: ‹[*uncurry isa-bump-unset-pre*]$_a$ *atom-assn*^{k} $*_a$ *heuristic-bump-assn*^{d} $\to$ *heuristic-bump-assn*›
  unfolding *isa-bump-unset-alt-def isa-bump-unset-pre-def*
  by *sepref*

sepref-def *isa-find-decomp-wl-imp-impl*
  is ‹*uncurry2 isa-find-decomp-wl-imp*›
  :: ‹*trail-pol-fast-assn*^{d} $*_a$ *uint32-nat-assn*^{k} $*_a$ *heuristic-bump-assn*^{d} $\to_a$
  trail-pol-fast-assn $\times_a$ *heuristic-bump-assn*›
  unfolding *isa-find-decomp-wl-imp-def get-maximum-level-remove-def*[*symmetric*] *PR-CONST-def*
    trail-pol-conv-to-no-CS-def
  supply *trail-conv-to-no-CS-def*[*simp*] *lit-of-hd-trail-def*[*simp*]
  supply [[*goals-limit=1*]] *literals-are-in-*$\mathcal{L}_{in}$*-add-mset*[*simp*]
  supply *vmtf-unset-pre-def*[*simp*]
  apply (*rewrite at* ‹*let* - = - − ⌑ *in* -› *annot-unat-snat-upcast*[**where** *'l=64*])
  apply (*annot-snat-const* ‹*TYPE*(*64*)›)
  by *sepref*

sepref-register *isa-bump-mark-to-rescore isa-find-decomp-wl-imp*
sepref-def *isa-bump-mark-to-rescore-code*
  is ‹*uncurry* (*isa-bump-mark-to-rescore*)›
  :: ‹*atom-assn*^{k} $*_a$ *heuristic-bump-assn*^{d} $\to_a$ *heuristic-bump-assn*›
  supply [[*goals-limit=1*]] *option.splits*[*split*] *vmtf-def*[*simp*] *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff*[*simp*]
    neq-NilE[*elim*!] *literals-are-in-*$\mathcal{L}_{in}$*-add-mset*[*simp*]
  unfolding *isa-vmtf-mark-to-rescore-pre-def isa-vmtf-mark-to-rescore-def*
    isa-bump-mark-to-rescore-def
  by *sepref*

sepref-def *isa-bump-mark-to-rescore-clause-fast-code*
  is ‹*uncurry2* (*isa-bump-mark-to-rescore-clause*)›
  :: ‹[*λ*((*N*, -), -). *length N* $\leq$ *snat64-max*]$_a$
      arena-fast-assn^k $*_a$ *sint64-nat-assn*^{k} $*_a$ *heuristic-bump-assn*^{d} $\to$ *heuristic-bump-assn*›
  supply [[*goals-limit=1*]] *arena-is-valid-clause-idx-le-unat64-max*[*intro*]
  unfolding *isa-bump-mark-to-rescore-clause-def PR-CONST-def*
  unfolding *while-eq-nfoldli*[*symmetric*]
  apply (*subst while-upt-while-direct*, *simp*)
  unfolding *nres-monad3*
  apply (*annot-snat-const* ‹*TYPE*(*64*)›)
  by *sepref*

sepref-def *isa-bump-rescore-fast-code*
  is ‹*uncurry2 isa-bump-rescore*›
  :: ‹*clause-ll-assn*^{k} $*_a$ *trail-pol-fast-assn*^{k} $*_a$ *heuristic-bump-assn*^{d} $\to_a$
      heuristic-bump-assn›
  unfolding *isa-bump-rescore-def*[*abs-def*] *PR-CONST-def isa-bump-rescore-body-def*
  supply [[*goals-limit = 1*]] *fold-is-None*[*simp*]
  apply (*annot-snat-const* ‹*TYPE*(*64*)›)
  by *sepref*

sepref-def *vmtf-mark-to-rescore-also-reasons-fast-code*
  is ‹*uncurry4* (*isa-vmtf-mark-to-rescore-also-reasons*)›

:: ‹[λ((((-, N), -), -), -). length N ≤ snat64-max]$_a$
trail-pol-fast-assn^k *$_a$ arena-fast-assn^k *$_a$ out-learned-assn^k *$_a$ unat-lit-assn^k *$_a$ heuristic-bump-assn^d
→
heuristic-bump-assn›
supply image-image[simp] uminus-$\mathcal{A}_{in}$-iff[iff] in-diffD[dest] option.splits[split]
in-$\mathcal{L}_{all}$-atm-of-$\mathcal{A}_{in}$[simp]
supply [[goals-limit=1]]
unfolding isa-vmtf-mark-to-rescore-also-reasons-def PR-CONST-def
unfolding while-eq-nfoldli[symmetric]
apply (subst while-upt-while-direct, simp)
apply (annot-snat-const ‹TYPE(64)›)
unfolding nres-monad3 case-option-split
by sepref

sepref-register isa-vmtf-bump-to-rescore-also-reasons-cl isa-vmtf-mark-to-rescore-also-reasons
isa-bump-heur-flush

sepref-def isa-vmtf-bump-to-rescore-also-reasons-cl-impl
is ‹uncurry4 (isa-vmtf-bump-to-rescore-also-reasons-cl)›
:: ‹[λ((((-, N), -), -), -). length N ≤ snat64-max]$_a$
trail-pol-fast-assn^k *$_a$ arena-fast-assn^k *$_a$ sint64-nat-assn^k *$_a$ unat-lit-assn^k *$_a$ heuristic-bump-assn^d
→
heuristic-bump-assn›
supply image-image[simp] uminus-$\mathcal{A}_{in}$-iff[iff] in-diffD[dest] option.splits[split]
in-$\mathcal{L}_{all}$-atm-of-$\mathcal{A}_{in}$[simp]
supply [[goals-limit=1]]
supply [dest] = isa-vmtf-bump-to-rescore-also-reasons-clD
unfolding isa-vmtf-bump-to-rescore-also-reasons-cl-def PR-CONST-def
unfolding while-eq-nfoldli[symmetric]
apply (subst while-upt-while-direct, simp)
apply (annot-snat-const ‹TYPE(64)›)
unfolding nres-monad3 case-option-split
by sepref

sepref-def isa-bump-heur-flush-impl
is ‹uncurry isa-bump-heur-flush›
:: ‹trail-pol-fast-assn^k *$_a$ heuristic-bump-assn^d →$_a$ heuristic-bump-assn›
unfolding isa-bump-heur-flush-def
by sepref

sepref-def isa-vmtf-heur-fst-code
is ‹isa-vmtf-heur-fst›
:: ‹heuristic-bump-assn^k →$_a$ atom-assn›
unfolding isa-vmtf-heur-fst-def
by sepref

definition isa-vmtf-heur-array-nth **where**
‹isa-vmtf-heur-array-nth x = vmtf-heur-array-nth (bump-get-heuristics x)›

lemma isa-vmtf-heur-array-nth-alt-def:
‹isa-vmtf-heur-array-nth x i = (case x of Bump-Heuristics hstable focused foc - ⇒
(vmtf-heur-array-nth focused i))›
by (cases x) (auto simp: bump-get-heuristics-def isa-vmtf-heur-array-nth-def)

sepref-register is-focused-heuristics vmtf-heur-array-nth

sepref-def *isa-vmtf-heur-array-nth-code*
is ‹*uncurry isa-vmtf-heur-array-nth*›
:: ‹$[\lambda(vm,\ i).\ i < length\ (fst\ (bump\text{-}get\text{-}heuristics\ vm))]_a$ *heuristic-bump-assn*^{k} $*_a$ *atom-assn*^{k} $\to$ *vmtf-node-assn*›
supply *if-splits*[*split*]
unfolding *isa-vmtf-heur-array-nth-alt-def bump-get-heuristics-def*
by *sepref*

definition *vmtf-array-fst* :: ‹*vmtf* $\Rightarrow$ *nat*› **where**
‹*vmtf-array-fst* = $(\lambda(a,\ b,\ c,\ d,\ e).\ c)$›

lemma *bumped-vmtf-array-fst-alt-def*: ‹*bumped-vmtf-array-fst x* = (*case x of Bump-Heuristics a b c d* $\Rightarrow$
(*vmtf-array-fst b*))›
by (*cases x*) (*auto simp*: *vmtf-array-fst-def current-vmtf-array-nxt-score-def*
bump-get-heuristics-def bumped-vmtf-array-fst-def)

sepref-def *vmtf-array-fst-code*
is ‹*RETURN o vmtf-array-fst*›
:: ‹*vmtf-assn*^{k} $\to_a$ *atom-assn*›
unfolding *vmtf-assn-def vmtf-array-fst-def*
by *sepref*

sepref-def *bumped-vmtf-array-fst-code*
is ‹*RETURN o bumped-vmtf-array-fst*›
:: ‹*heuristic-bump-assn*^{k} $\to_a$ *atom-assn*›
unfolding *bumped-vmtf-array-fst-alt-def*
by *sepref*

sepref-register *access-focused-vmtf-array*

definition *access-vmtf-array* :: ‹*vmtf* $\Rightarrow$ *nat* $\Rightarrow$ - *nres*› **where**
‹*access-vmtf-array* = $(\lambda(a,\ b,\ c,\ d,\ f)\ i.$ *do* {
ASSERT $(i < length\ a)$;
RETURN $(a\ !\ i)\}$)›

lemma *access-focused-vmtf-array-alt-def*:
‹*access-focused-vmtf-array x i* = (*case x of Bump-Heuristics a b c d* $\Rightarrow$ *do* {
access-vmtf-array b i
})›
by (*cases x*) (*auto simp*: *access-focused-vmtf-array-def access-vmtf-array-def*
bump-get-heuristics-def)

sepref-def *access-vmtf-array-code*
is ‹*uncurry access-vmtf-array*›
:: ‹*vmtf-assn*^{k} $*_a$ *atom-assn*^{k} $\to_a$ *vmtf-node-assn*›
unfolding *access-vmtf-array-def vmtf-assn-def*
apply (*rewrite at* ‹*RETURN* ⌑› *annot-index-of-atm*)
by *sepref*

sepref-register *access-vmtf-array*
sepref-def *access-focused-vmtf-array-code*

is ‹*uncurry access-focused-vmtf-array*›
:: ‹*heuristic-bump-assn*^{k} $*_a$ *atom-assn*^{k} $\rightarrow_a$ *vmtf-node-assn*›
unfolding *access-focused-vmtf-array-alt-def*
by *sepref*

experiment begin

export-llvm
isa-vmtf-bump-to-rescore-also-reasons-cl-impl
vmtf-mark-to-rescore-also-reasons-fast-code
isa-bump-rescore-fast-code
isa-bump-mark-to-rescore-clause-fast-code
isa-bump-heur-flush-impl
isa-vmtf-heur-array-nth-code

end

end
theory *IsaSAT-Setup2-LLVM*
imports *IsaSAT-Setup*
IsaSAT-Bump-Heuristics-LLVM
IsaSAT-Setup0-LLVM
begin

definition *opts-restart-st-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*opts-restart-st-fast-code* = *read-opts-wl-heur-code opts-rel-restart-code*›

global-interpretation *opts-restart*: *read-opts-param-adder0* **where**
f' = ‹*RETURN o opts-restart*› **and**
f = *opts-rel-restart-code* **and**
x-assn = *bool1-assn* **and**
P = ‹λ-. *True*›
rewrites ‹*read-opts-wl-heur* (*RETURN o opts-restart*) = *RETURN o opts-restart-st*› **and**
‹*read-opts-wl-heur-code opts-rel-restart-code* = *opts-restart-st-fast-code*›
apply *unfold-locales*
apply (*rule opts-refine*; *assumption*)
subgoal by (*auto simp*: *opts-restart-st-def read-all-st-def intro*!: *ext*
split: *isasat-int-splits*)
subgoal by (*auto simp*: *opts-restart-st-fast-code-def*)
done

definition *opts-reduction-st-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*opts-reduction-st-fast-code* = *read-opts-wl-heur-code opts-rel-reduce-code*›

global-interpretation *opts-reduce*: *read-opts-param-adder0* **where**
f' = ‹*RETURN o opts-reduce*› **and**
f = *opts-rel-reduce-code* **and**
x-assn = *bool1-assn* **and**
P = ‹λ-. *True*›
rewrites ‹*read-opts-wl-heur* (*RETURN o opts-reduce*) = *RETURN o opts-reduction-st*› **and**
‹*read-opts-wl-heur-code opts-rel-reduce-code* = *opts-reduction-st-fast-code*›
apply *unfold-locales*
apply (*rule opts-refine*; *assumption*)
subgoal by (*auto simp*: *opts-reduction-st-fast-code-def read-all-st-def opts-reduction-st-def intro*!: *ext*
split: *isasat-int-splits*)
subgoal by (*auto simp*: *opts-reduction-st-fast-code-def*)

done

definition *opts-unbounded-mode-st-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*opts-unbounded-mode-st-fast-code* = *read-opts-wl-heur-code opts-rel-unbounded-mode-code*›

global-interpretation *opts-unbounded-mode*: *read-opts-param-adder0* **where**
f' = ‹*RETURN o opts-unbounded-mode*› **and**
f = *opts-rel-unbounded-mode-code* **and**
x-assn = *bool1-assn* **and**
P = ‹λ-. *True*›
rewrites ‹*read-opts-wl-heur* (*RETURN o opts-unbounded-mode*) = *RETURN o opts-unbounded-mode-st*›
and
‹*read-opts-wl-heur-code opts-rel-unbounded-mode-code* = *opts-unbounded-mode-st-fast-code*›
apply *unfold-locales*
apply (*rule opts-refine*; *assumption*)
subgoal by (*auto simp*: *read-all-st-def opts-unbounded-mode-st-def intro*!: *ext*
split: *isasat-int-splits*)
subgoal by (*auto simp*: *opts-unbounded-mode-st-fast-code-def*)
done

definition *opts-minimum-between-restart-st-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*opts-minimum-between-restart-st-fast-code* = *read-opts-wl-heur-code opts-rel-miminum-between-restart-code*›

global-interpretation *opts-minimum-between-restart*: *read-opts-param-adder0* **where**
f' = ‹*RETURN o opts-minimum-between-restart*› **and**
f = *opts-rel-miminum-between-restart-code* **and**
x-assn = *word-assn* **and**
P = ‹λ-. *True*›
rewrites ‹*read-opts-wl-heur* (*RETURN o opts-minimum-between-restart*) = *RETURN o opts-minimum-between-restart-
and
‹*read-opts-wl-heur-code opts-rel-miminum-between-restart-code* = *opts-minimum-between-restart-st-fast-code*›
apply *unfold-locales*
apply (*rule opts-refine*; *assumption*)
subgoal by (*auto simp*: *read-all-st-def opts-minimum-between-restart-st-def intro*!: *ext*
split: *isasat-int-splits*)
subgoal by (*auto simp*: *opts-minimum-between-restart-st-fast-code-def*)
done

definition *opts-restart-coeff1-st-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*opts-restart-coeff1-st-fast-code* = *read-opts-wl-heur-code opts-rel-restart-coeff1-code*›

global-interpretation *opts-restart-coeff1*: *read-opts-param-adder0* **where**
f' = ‹*RETURN o opts-restart-coeff1*› **and**
f = *opts-rel-restart-coeff1-code* **and**
x-assn = *word-assn* **and**
P = ‹λ-. *True*›
rewrites ‹*read-opts-wl-heur* (*RETURN o opts-restart-coeff1*) = *RETURN o opts-restart-coeff1-st*›
and
‹*read-opts-wl-heur-code opts-rel-restart-coeff1-code* = *opts-restart-coeff1-st-fast-code*›
apply *unfold-locales*
apply (*rule opts-refine*; *assumption*)
subgoal by (*auto simp*: *read-all-st-def opts-minimum-between-restart-st-def opts-restart-coeff1-st-def*
intro!: *ext*
split: *isasat-int-splits*)
subgoal by (*auto simp*: *opts-restart-coeff1-st-fast-code-def*)
done

definition *opts-restart-coeff2-st-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*opts-restart-coeff2-st-fast-code* = *read-opts-wl-heur-code opts-rel-restart-coeff2-code*›

global-interpretation *opts-restart-coeff2*: *read-opts-param-adder0* **where**
f' = ‹*RETURN o opts-restart-coeff2*› **and**
f = *opts-rel-restart-coeff2-code* **and**
x-assn = ‹*snat-assn'* (*TYPE*(*64*))› **and**
P = ‹λ-. *True*›
rewrites ‹*read-opts-wl-heur* (*RETURN o opts-restart-coeff2*) = *RETURN o opts-restart-coeff2-st*› **and**
‹*read-opts-wl-heur-code opts-rel-restart-coeff2-code* = *opts-restart-coeff2-st-fast-code*›
apply *unfold-locales*
apply (*rule opts-refine*; *assumption*)
subgoal by (*auto simp*: *read-all-st-def opts-minimum-between-restart-st-def opts-restart-coeff2-st-def intro*!: *ext*
split: *isasat-int-splits*)
subgoal by (*auto simp*: *opts-restart-coeff2-st-fast-code-def*)
done

definition *units-since-last-GC-st-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*units-since-last-GC-st-code* = *read-stats-wl-heur-code units-since-last-GC-stats-impl*›

global-interpretation *units-since-last-GC*: *read-stats-param-adder0* **where**
f' = ‹*RETURN o units-since-last-GC*› **and**
f = *units-since-last-GC-stats-impl* **and**
x-assn = *word-assn* **and**
P = ‹λ-. *True*›
rewrites ‹*read-stats-wl-heur* (*RETURN o units-since-last-GC*) = *RETURN o units-since-last-GC-st*› **and**
‹*read-stats-wl-heur-code units-since-last-GC-stats-impl* = *units-since-last-GC-st-code*›
apply *unfold-locales*
apply (*rule units-since-last-GC-stats-impl.refine*; *assumption*)
subgoal by (*auto simp*: *read-all-st-def units-since-last-GC-st-def intro*!: *ext*
split: *isasat-int-splits*)
subgoal by (*auto simp*: *units-since-last-GC-st-code-def*)
done

definition *units-since-beginning-st-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*units-since-beginning-st-code* = *read-stats-wl-heur-code units-since-beginning-stats-impl*›

global-interpretation *units-since-beginning*: *read-stats-param-adder0* **where**
f' = ‹*RETURN o units-since-beginning*› **and**
f = *units-since-beginning-stats-impl* **and**
x-assn = *word-assn* **and**
P = ‹λ-. *True*›
rewrites ‹*read-stats-wl-heur* (*RETURN o units-since-beginning*) = *RETURN o units-since-beginning-st*› **and**
‹*read-stats-wl-heur-code units-since-beginning-stats-impl* = *units-since-beginning-st-code*›
apply *unfold-locales*
apply (*rule units-since-beginning-stats-impl.refine*; *assumption*)
subgoal by (*auto simp*: *read-all-st-def units-since-beginning-st-def intro*!: *ext*
split: *isasat-int-splits*)
subgoal by (*auto simp*: *units-since-beginning-st-code-def*)
done

definition *get-GC-units-opt-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*get-GC-units-opt-code* = *read-opts-wl-heur-code opts-rel-GC-units-lim-code*›

global-interpretation *opts-GC-units-lim*: *read-opts-param-adder0* **where**
f' = ‹*RETURN o opts-GC-units-lim*› **and**
f = *opts-rel-GC-units-lim-code* **and**
x-assn = *word-assn* **and**
P = ‹λ-. *True*›
rewrites ‹*read-opts-wl-heur* (*RETURN o opts-GC-units-lim*) = *RETURN o get-GC-units-opt*› **and**
‹*read-opts-wl-heur-code opts-rel-GC-units-lim-code* = *get-GC-units-opt-code*›
apply *unfold-locales*
apply (*rule opts-refine*)
subgoal by (*auto simp*: *read-all-st-def get-GC-units-opt-def intro*!: *ext*
split: *isasat-int-splits*)
subgoal by (*auto simp*: *get-GC-units-opt-code-def*)
done

definition *get-target-opts-impl* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*get-target-opts-impl* = *read-opts-wl-heur-code opts-rel-target-code*›

global-interpretation *get-target-opts*: *read-opts-param-adder0* **where**
f' = ‹*RETURN o opts-target*› **and**
f = *opts-rel-target-code* **and**
x-assn = ‹*word-assn'* *TYPE*(*3*)› **and**
P = ‹λ-. *True*›
rewrites ‹*read-opts-wl-heur* (*RETURN o opts-target*) = *RETURN o get-target-opts*› **and**
‹*read-opts-wl-heur-code opts-rel-target-code* = *get-target-opts-impl*›
apply *unfold-locales*
apply (*rule opts-refine*)
subgoal by (*auto simp*: *get-target-opts-def read-all-st-def intro*!: *ext*
split: *isasat-int-splits*)
subgoal by (*auto simp*: *get-target-opts-impl-def*)
done

definition *isasat-length-trail-st-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*isasat-length-trail-st-code* = *read-trail-wl-heur-code isa-length-trail-fast-code*›

global-interpretation *trail-length*: *read-trail-param-adder0* **where**
f' = ‹*RETURN o isa-length-trail*› **and**
f = *isa-length-trail-fast-code* **and**
x-assn = *sint64-nat-assn* **and**
P = ‹λ-. *True*›
rewrites ‹*read-trail-wl-heur* (*RETURN o isa-length-trail*) = *RETURN o isasat-length-trail-st*› **and**
‹*read-trail-wl-heur-code isa-length-trail-fast-code* = *isasat-length-trail-st-code*›
apply *unfold-locales*
apply (*rule isa-length-trail-fast-code.refine*)
subgoal by (*auto simp*: *isa-length-trail-def read-all-st-def isasat-length-trail-st-def*
intro!: *ext split*: *isasat-int-splits*)
subgoal by (*auto simp*: *isasat-length-trail-st-code-def*)
done

lemma *get-pos-of-level-in-trail-imp-alt-def*:
‹*get-pos-of-level-in-trail-imp* = (λ*S C*. *do* {*k* ← *get-pos-of-level-in-trail-imp S C*; *RETURN k*})›
by *auto*

sepref-def *get-pos-of-level-in-trail-imp-code*

is ‹*uncurry get-pos-of-level-in-trail-imp*›
:: ‹*trail-pol-fast-assn*^{k} $*_a$ *uint32-nat-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*›
supply [[*goals-limit=1*]]
apply (*subst get-pos-of-level-in-trail-imp-alt-def*)
apply (*rewrite* **in** ‹-› *eta-expand*[**where** *f* = *RETURN*])
apply (*rewrite* **in** ‹*RETURN* ⌑› *annot-unat-snat-upcast*[**where** *'l=64*])
by *sepref*

definition *get-pos-of-level-in-trail-imp-st-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*get-pos-of-level-in-trail-imp-st-code* = (λ*N C*. *read-trail-wl-heur-code* (λ*c*. *get-pos-of-level-in-trail-imp-code c C*) *N*)›

global-interpretation *pos-of-level-in-trail*: *read-trail-param-adder* **where**
R = ‹*unat-rel'* *TYPE*(*32*)› **and**
f' = ‹λ*M c*. *get-pos-of-level-in-trail-imp c M*› **and**
f = ‹λ*M c*. *get-pos-of-level-in-trail-imp-code c M*› **and**
x-assn = *sint64-nat-assn* **and**
P = ‹λ- -. *True*›
rewrites ‹(λ*N C'*. *read-trail-wl-heur* (λ*c*. *get-pos-of-level-in-trail-imp c C'*) *N*) = *get-pos-of-level-in-trail-imp-st*›
and
‹(λ*N C*. *read-trail-wl-heur-code* (λ*c*. *get-pos-of-level-in-trail-imp-code c C*) *N*) = *get-pos-of-level-in-trail-imp-st-code*›
apply *unfold-locales*
apply (*subst lambda-comp-true*)+
apply (*rule get-pos-of-level-in-trail-imp-code.refine*)
subgoal by (*auto simp*: *get-pos-of-level-in-trail-imp-st-def read-all-st-def*
intro!: *ext split*: *isasat-int-splits*)
subgoal by (*auto simp*: *get-pos-of-level-in-trail-imp-st-code-def*)
done

definition *get-global-conflict-count-impl* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*get-global-conflict-count-impl* = *read-stats-wl-heur-code stats-conflicts-impl*›

global-interpretation *stats-conflict*: *read-stats-param-adder0* **where**
f' = ‹*RETURN o stats-conflicts*› **and**
f = *stats-conflicts-impl* **and**
x-assn = *word-assn* **and**
P = ‹λ-. *True*›
rewrites ‹*read-stats-wl-heur* (*RETURN o stats-conflicts*) = *RETURN o get-global-conflict-count*› **and**
‹*read-stats-wl-heur-code stats-conflicts-impl* = *get-global-conflict-count-impl*›
apply *unfold-locales*
apply (*rule stats-conflicts-impl.refine*; *assumption*)
subgoal by (*auto simp*: *read-all-st-def stats-conflicts-def get-global-conflict-count-def intro*!: *ext*
split: *isasat-int-splits*)
subgoal by (*auto simp*: *get-global-conflict-count-impl-def*)
done

lemmas [*unfolded lambda-comp-true*, *sepref-fr-rules*] =
opts-restart.refine[*unfolded*]
opts-reduce.refine[*unfolded*]
opts-unbounded-mode.refine
opts-minimum-between-restart.refine
opts-restart-coeff1.refine
opts-restart-coeff2.refine
units-since-last-GC.refine
units-since-beginning.refine
opts-GC-units-lim.refine

trail-length.refine
pos-of-level-in-trail.refine
stats-conflict.refine
get-target-opts.refine

sepref-register *opts-reduction-st opts-restart-st opts-restart-coeff2-st opts-restart-coeff1-st opts-minimum-between-restart-st opts-unbounded-mode-st get-GC-units-opt units-since-last-GC-st isasat-length-trail-st get-pos-of-level-in-trail-imp-st units-since-beginning*

lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
opts-restart-st-fast-code-def[*unfolded read-all-st-code-def*]
opts-reduction-st-fast-code-def[*unfolded read-all-st-code-def*]
opts-unbounded-mode-st-fast-code-def[*unfolded read-all-st-code-def*]
opts-minimum-between-restart-st-fast-code-def[*unfolded read-all-st-code-def*]
opts-restart-coeff1-st-fast-code-def[*unfolded read-all-st-code-def*]
opts-restart-coeff2-st-fast-code-def[*unfolded read-all-st-code-def*]
units-since-last-GC-st-code-def[*unfolded read-all-st-code-def*]
units-since-beginning-st-code-def[*unfolded read-all-st-code-def*]
get-GC-units-opt-code-def[*unfolded read-all-st-code-def*]
isasat-length-trail-st-code-def[*unfolded read-all-st-code-def*]
get-pos-of-level-in-trail-imp-st-code-def[*unfolded read-all-st-code-def*]
get-global-conflict-count-impl-def[*unfolded read-all-st-code-def*]
get-target-opts-impl-def[*unfolded read-all-st-code-def*]

sepref-register *reset-units-since-last-GC*

lemma *reset-units-since-last-GC-st-alt-def*:
‹*reset-units-since-last-GC-st S* =
(*let* (*stats*, *S*) = *extract-stats-wl-heur S in*
let stats = *reset-units-since-last-GC stats in*
let S = *update-stats-wl-heur stats S in S*
)›
by (*auto simp*: *reset-units-since-last-GC-st-def state-extractors split*: *isasat-int-splits*)

sepref-def *reset-units-since-last-GC-st-code*
is ‹*RETURN o reset-units-since-last-GC-st*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *reset-units-since-last-GC-st-alt-def*
by *sepref*

lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] = *units-since-last-GC-st-code-def*[*unfolded read-all-st-code-def*]
lemmas [*llvm-code del*] = *units-since-last-GC-st-code-def*

lemma *mop-isasat-length-trail-st-alt-def*:
‹*mop-isasat-length-trail-st S* = *do* {
ASSERT(*isa-length-trail-pre* (*get-trail-wl-heur S*));
RETURN (*isasat-length-trail-st S*)
}›
unfolding *isasat-length-trail-st-def mop-isasat-length-trail-st-def*
by *auto*

sepref-def *mop-isasat-length-trail-st-code*

is ‹*mop-isasat-length-trail-st*›
:: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*›
supply [[*goals-limit=1*]]
unfolding *mop-isasat-length-trail-st-alt-def*
by *sepref*

definition *arena-status-st* **where**
‹*arena-status-st* = (λ*S*. *arena-status* (*get-clauses-wl-heur S*))›

definition *arena-status-st-impl* **where**
‹*arena-status-st-impl* = (λ*S C'*. *read-arena-wl-heur-code* (λ*N*. *arena-status-impl N C'*) *S*)›

global-interpretation *arena-is-valid*: *read-arena-param-adder* **where**
R = ‹*snat-rel'* *TYPE*(*64*)› **and**
f = ‹λ*C N*. *arena-status-impl N C*› **and**
f' = ‹(λ*C' N*. (*RETURN oo arena-status*) *N C'*)› **and**
x-assn = *status-impl-assn* **and**
P = ‹(λ*C S*. *arena-is-valid-clause-vdom S C*)›
rewrites ‹(λ*S C'*. *read-arena-wl-heur* (λ*N*. (*RETURN oo arena-status*) *N C'*) *S*) = *RETURN oo arena-status-st*› **and**
‹(λ*S C'*. *read-arena-wl-heur-code* (λ*N*. *arena-status-impl N C'*) *S*) = *arena-status-st-impl*› **and**
‹*arena-is-valid.mop* = *mop-arena-status-st*› **and**
‹(λ*S*. *arena-is-valid-clause-vdom* (*get-clauses-wl-heur S*)) = *curry clause-not-marked-to-delete-heur-pre*›
apply *unfold-locales*
apply (*rule arena-status-impl.refine*)
subgoal by (*auto simp*: *mop-arena-status-st-def read-all-st-def arena-status-st-def*
intro!: *ext split*: *isasat-int-splits*)
subgoal by (*auto simp*: *arena-status-st-impl-def*)
subgoal by (*auto simp*: *read-arena-param-adder-ops.mop-def mop-arena-status-st-def mop-arena-status-def read-all-st-def arena-status-st-def*
intro!: *ext split*: *isasat-int-splits*)
subgoal by (*auto simp*: *clause-not-marked-to-delete-heur-pre-def*)
done

lemmas [*sepref-fr-rules*] = *arena-is-valid.mop-refine arena-is-valid.refine*[*unfolded uncurry-curry-id*]

sepref-def *mop-arena-status-st-impl*
is ‹*uncurry mop-arena-status-st*›
:: ‹*isasat-bounded-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *status-impl-assn*›
supply [[*goals-limit=1*]]
by *sepref*

definition *arena-length-st-impl* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*arena-length-st-impl* = (λ*S C'*. *read-arena-wl-heur-code* (λ*N*. *arena-length-impl N C'*) *S*)›

global-interpretation *arena-length-clause*: *read-arena-param-adder* **where**
R = ‹*snat-rel'* *TYPE*(*64*)› **and**
f = ‹λ*C N*. *arena-length-impl N C*› **and**
f' = ‹(λ*C' N*. (*RETURN oo arena-length*) *N C'*)› **and**
x-assn = *sint64-nat-assn* **and**
P = ‹(λ*C S*. *arena-is-valid-clause-idx S C*)›
rewrites ‹(λ*S C'*. *read-arena-wl-heur* (λ*N*. (*RETURN oo arena-status*) *N C'*) *S*) = *RETURN oo arena-status-st*› **and**
‹(λ*S C'*. *read-arena-wl-heur-code* (λ*N*. *arena-length-impl N C'*) *S*) = *arena-length-st-impl*› **and**
‹*arena-length-clause.mop* = *mop-arena-length-st*›

apply *unfold-locales*
apply (*rule arena-length-impl.refine*)
subgoal by (*auto simp*: *mop-arena-status-st-def read-all-st-def arena-status-st-def*
intro!: *ext split*: *isasat-int-splits*)
subgoal by (*auto simp*: *arena-length-st-impl-def*)
subgoal
by (*auto simp*: *read-arena-param-adder-ops.mop-def mop-arena-length-st-def mop-arena-length-def*
read-all-st-def
split: *isasat-int-splits intro*!: *ext*)
done

lemmas [*sepref-fr-rules*] = *arena-length-clause.mop-refine*
lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] = *arena-length-st-impl-def*[*unfolded read-all-st-code-def*]
arena-status-st-impl-def[*unfolded read-all-st-code-def*]

sepref-definition *arena-full-length-impl*
is ‹*RETURN o length*›
:: ‹*arena-fast-assn*^{k} →$_a$ *sint64-nat-assn*›
by *sepref*

definition *full-arena-length-st-impl* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*full-arena-length-st-impl* = *read-arena-wl-heur-code* (*arena-full-length-impl*)›

global-interpretation *arena-full-length*: *read-arena-param-adder0* **where**
f = ‹*arena-full-length-impl*› **and**
f' = ‹(*RETURN o length*)› **and**
x-assn = *sint64-nat-assn* **and**
P = ‹(λ-. *True*)›
rewrites ‹*read-arena-wl-heur* (*RETURN o length*) = *RETURN o full-arena-length-st*› **and**
‹*read-arena-wl-heur-code* (*arena-full-length-impl*) = *full-arena-length-st-impl*›
apply *unfold-locales*
apply (*rule arena-full-length-impl.refine*)
subgoal by (*auto simp*: *mop-arena-status-st-def read-all-st-def full-arena-length-st-def*
intro!: *ext split*: *isasat-int-splits*)
subgoal by (*auto simp*: *full-arena-length-st-impl-def*)
done

lemma *incr-wasted-st-alt-def*:
‹*incr-wasted-st* = (λ*waste S*. *do*{
let (*heur*, *S*) = *extract-heur-wl-heur S in*
let heur = *incr-wasted waste heur in*
let S = *update-heur-wl-heur heur S in S*
})›
by (*auto simp*: *incr-wasted-st-def state-extractors intro*!: *ext split*: *isasat-int-splits*)

sepref-def *incr-wasted-st-impl*
is ‹*uncurry* (*RETURN oo incr-wasted-st*)›
:: ‹*word64-assn*^{k} ×$_a$ *isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
supply[[*goals-limit=1*]]
unfolding *incr-wasted-st-alt-def*
by *sepref*

lemma *id-clvls-assn*: ‹(*Mreturn*, *RETURN*) ∈ (*uint32-nat-assn*)^k →$_a$ *uint32-nat-assn*›
by *sepref-to-hoare vcg*

definition *get-count-max-lvls-heur-impl* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**

‹*get-count-max-lvls-heur-impl* = *read-ccount-wl-heur-code* (*Mreturn*)›

global-interpretation *get-count-max-lvls*: *read-ccount-param-adder0* **where**
f = ‹*Mreturn*› **and**
f' = ‹(*RETURN*)› **and**
x-assn = *uint32-nat-assn* **and**
P = ‹(λ-. *True*)›
rewrites ‹*read-ccount-wl-heur* (*RETURN*) = *RETURN o get-count-max-lvls-heur*› **and**
‹*read-ccount-wl-heur-code* (*Mreturn*) = *get-count-max-lvls-heur-impl*›
apply *unfold-locales*
apply (*rule id-clvls-assn*)
subgoal by (*auto simp*: *read-all-st-def*
intro!: *ext split*: *isasat-int-splits*)
subgoal by (*auto simp*: *get-count-max-lvls-heur-impl-def*)
done

lemmas [*sepref-fr-rules*] = *arena-full-length.refine get-count-max-lvls.refine*
lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] = *full-arena-length-st-impl-def*[*unfolded read-all-st-code-def*]
arena-full-length-impl-def
get-count-max-lvls-heur-impl-def[*unfolded read-all-st-code-def*]

lemma *clss-size-resetUS0-st-alt-def*:
‹*clss-size-resetUS0-st S* =
(*let* (*stats*, *S*) = *extract-lcount-wl-heur S in*
let stats = *clss-size-resetUS0 stats in*
let S = *update-lcount-wl-heur stats S in S*
)›
by (*auto simp*: *clss-size-resetUS0-st-def state-extractors*
split: *isasat-int-splits*)

sepref-def *clss-size-resetUS0-st*
is ‹*RETURN o clss-size-resetUS0-st*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
unfolding *clss-size-resetUS0-st-alt-def*
by *sepref*

lemma *length-ll*[*def-pat-rules*]: ‹*length-ll*$*xs*$*i* ≡ *op-list-list-llen*$*xs*$*i*›
by (*auto simp*: *op-list-list-llen-def length-ll-def*)

sepref-def *length-watchlist-impl*
is ‹*uncurry* (*RETURN oo length-watchlist*)›
:: ‹[*uncurry* (λ*S L*. *nat-of-lit L* < *length S*)]$_a$ *watchlist-fast-assn*^{k} *$_a$ *unat-lit-assn*^{k} → *sint64-nat-assn*›
unfolding *length-watchlist-def*
by *sepref*

definition *length-ll-fs-heur-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*length-ll-fs-heur-fast-code* = (λ*N C*. *read-watchlist-wl-heur-code* (λ*N*. *length-watchlist-impl N C*) *N*)›
global-interpretation *watched-by-app*: *read-watchlist-param-adder* **where**
R = ‹*unat-lit-rel*› **and**
f = ‹λ*C N*. *length-watchlist-impl N C*› **and**
f' = ‹λ*C N*. (*RETURN oo length-watchlist*) *N C*› **and**
x-assn = *sint64-nat-assn* **and**
P = ‹(λ*L S*. *nat-of-lit L* < *length* (*S*))›
rewrites
‹(λ*N C'*. *read-watchlist-wl-heur* (λ*N*. (*RETURN* ∘∘ *length-watchlist*) *N C'*) *N*) = *RETURN oo*

length-ll-fs-heur› **and**
‹(λ*N C*. *read-watchlist-wl-heur-code* (λ*N*. *length-watchlist-impl N C*) *N*) = *length-ll-fs-heur-fast-code*›
and
‹*watched-by-app.XX.mop* = *mop-length-watched-by-int*›
apply *unfold-locales*
apply (*rule length-watchlist-impl.refine*)
subgoal
by (*auto intro*!: *ext simp*: *length-ll-fs-heur-def read-all-st-def length-watchlist-def*
length-ll-def
split: *isasat-int-splits*)
subgoal by (*auto simp*: *length-ll-fs-heur-fast-code-def*)
subgoal
by (*auto simp*: *mop-length-watched-by-int-def read-all-param-adder-ops.mop-def read-all-st-def*
length-watchlist-def length-ll-def split: *isasat-int-splits*
intro!: *ext*)
done

lemmas [*sepref-fr-rules*] = *watched-by-app.refine watched-by-app.XX.mop-refine*
lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
length-ll-fs-heur-fast-code-def[*unfolded read-all-st-code-def*]

definition *get-vmtf-heur-fst-impl* **where**
‹*get-vmtf-heur-fst-impl* = *read-vmtf-wl-heur-code* (*isa-vmtf-heur-fst-code*)›

global-interpretation *vmtf-fst*: *read-vmtf-param-adder0* **where**
f' = ‹*isa-vmtf-heur-fst*› **and**
f = ‹*isa-vmtf-heur-fst-code*› **and**
x-assn = *atom-assn* **and**
P = ‹(λ-. *True*)›
rewrites
‹*read-vmtf-wl-heur* (*isa-vmtf-heur-fst*) = *RETURN o get-vmtf-heur-fst*› **and**
‹*read-vmtf-wl-heur-code* (*isa-vmtf-heur-fst-code*) = *get-vmtf-heur-fst-impl*›
apply *unfold-locales*
apply (*rule isa-vmtf-heur-fst-code.refine*)
subgoal
by (*auto intro*!: *ext simp*: *get-vmtf-heur-fst-def read-all-st-def vmtf-heur-fst-def*
isa-vmtf-heur-fst-def
split: *isasat-int-splits bump-heuristics-splits*)
subgoal by (*auto simp*: *get-vmtf-heur-fst-impl-def*)
done

definition *get-bump-heur-array-nth-impl* **where**
‹*get-bump-heur-array-nth-impl N C'* = *read-vmtf-wl-heur-code* (λ*M*. *isa-vmtf-heur-array-nth-code M C'*) *N*›

lemma *get-vmtf-heur-array-alt-def*: ‹*get-vmtf-heur-array S* = *fst* (*bump-get-heuristics* (*get-vmtf-heur S*))›
by (*auto simp*: *get-vmtf-heur-array-def bump-get-heuristics-def*)

global-interpretation *vmtf-array-nth*: *read-vmtf-param-adder* **where**
f' = ‹λ*a b*. *isa-vmtf-heur-array-nth b a*› **and**
f = ‹λ*a b*. *isa-vmtf-heur-array-nth-code b a*› **and**
x-assn = *vmtf-node-assn* **and**
P = ‹(λ*n S*. *n* < *length* (*fst* (*bump-get-heuristics S*)))› **and**
R = *atom-rel*
rewrites

‹(λN C′. read-vmtf-wl-heur (λM. isa-vmtf-heur-array-nth M C′) N) = RETURN oo get-bump-heur-array-nth›
and
‹(λN C′. read-vmtf-wl-heur-code (λM. isa-vmtf-heur-array-nth-code M C′) N) = get-bump-heur-array-nth-impl›
apply *unfold-locales*
apply (*subst* (*3*)*uncurry-def*)
apply (*rule isa-vmtf-heur-array-nth-code.refine*)
subgoal
by (*auto intro*!: *ext simp*: *get-bump-heur-array-nth-def read-all-st-def vmtf-heur-array-nth-def*
get-vmtf-heur-array-def isa-vmtf-heur-array-nth-def bump-get-heuristics-def
split: *isasat-int-splits bump-heuristics-splits prod.splits*)
subgoal by (*auto simp*: *get-bump-heur-array-nth-impl-def*[*abs-def*])
done

lemmas [*sepref-fr-rules*] = *vmtf-fst.refine*
vmtf-array-nth.refine[*unfolded get-vmtf-heur-array-def*[*symmetric*, *unfolded comp-def*] *get-vmtf-heur-array-alt-def*[*symm*

lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
get-vmtf-heur-fst-impl-def[*unfolded read-all-st-code-def*]
get-bump-heur-array-nth-impl-def[*unfolded read-all-st-code-def*]

lemma *mop-mark-added-heur-st-alt-def*:
‹*mop-mark-added-heur-st L S* = *do* {
let (*heur*, *S*) = *extract-heur-wl-heur S*;
heur ← *mop-mark-added-heur L True heur*;
RETURN (*update-heur-wl-heur heur S*)
}›
unfolding *mop-mark-added-heur-st-def*
by (*auto simp*: *incr-restart-stat-def state-extractors split*: *isasat-int-splits*
intro!: *ext*)

sepref-def *mop-mark-added-heur-iml*
is ‹*uncurry2 mop-mark-added-heur*›
:: ‹*atom-assn*^{k} $*_a$ *bool1-assn*^{k} $*_a$ *heuristic-assn*^{d} $\to_a$ *heuristic-assn*›
supply [*sepref-fr-rules*] = *mark-added-heur-impl-refine*
unfolding *mop-mark-added-heur-def*
by *sepref*

sepref-register *mop-mark-added-heur mop-mark-added-heur-st mark-added-clause-heur2 maybe-mark-added-clause-heur*

sepref-def *mop-mark-added-heur-st-impl*
is ‹*uncurry mop-mark-added-heur-st*›
:: ‹*atom-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\to_a$ *isasat-bounded-assn*›
unfolding *mop-mark-added-heur-st-alt-def*
by *sepref*

experiment
begin

export-llvm *opts-reduction-st-fast-code opts-restart-st-fast-code opts-unbounded-mode-st-fast-code*
opts-minimum-between-restart-st-fast-code mop-arena-status-st-impl full-arena-length-st-impl
get-global-conflict-count-impl get-count-max-lvls-heur-impl clss-size-resetUS0-st
end

end

theory *IsaSAT-Setup3-LLVM*
imports *IsaSAT-Setup*
IsaSAT-Setup0-LLVM
begin

definition *wasted-bytes-st-impl* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*wasted-bytes-st-impl*= *read-heur-wl-heur-code wasted-of-stats-impl*›

global-interpretation *wasted-of*: *read-heur-param-adder0* **where**
f' = ‹*RETURN o wasted-of*› **and**
f = *wasted-of-stats-impl* **and**
x-assn = ‹*word64-assn*› **and**
P = ‹(λ-. *True*)›
rewrites
‹*read-heur-wl-heur* (*RETURN o wasted-of*) = *RETURN o wasted-bytes-st*› **and**
‹*read-heur-wl-heur-code wasted-of-stats-impl* = *wasted-bytes-st-impl*›
apply *unfold-locales*
apply (*rule heur-refine*)
subgoal by (*auto simp*: *wasted-bytes-st-def read-all-st-def intro*!: *ext split*: *isasat-int-splits*)
subgoal by (*auto simp*: *wasted-bytes-st-impl-def*)
done

definition *get-restart-phase-imp* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*get-restart-phase-imp* = *read-heur-wl-heur-code current-restart-phase-impl*›

global-interpretation *current-restart-phase*: *read-heur-param-adder0* **where**
f' = ‹*RETURN o current-restart-phase*› **and**
f = *current-restart-phase-impl* **and**
x-assn = ‹*word64-assn*› **and**
P = ‹(λ-. *True*)›
rewrites
‹*read-heur-wl-heur* (*RETURN o current-restart-phase*) = *RETURN o get-restart-phase*› **and**
‹*read-heur-wl-heur-code current-restart-phase-impl* = *get-restart-phase-imp*›
apply *unfold-locales*
apply (*rule heur-refine*)
subgoal by (*auto simp*: *get-restart-phase-def read-all-st-def intro*!: *ext split*: *isasat-int-splits*)
subgoal by (*auto simp*: *get-restart-phase-imp-def*)
done

definition *next-pure-lits-schedule-st-impl* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*next-pure-lits-schedule-st-impl* = *read-heur-wl-heur-code next-pure-lits-schedule-info-stats-impl*›

global-interpretation *next-pure-lits-schedule*: *read-heur-param-adder0* **where**
f' = ‹*RETURN o next-pure-lits-schedule*› **and**
f = *next-pure-lits-schedule-info-stats-impl* **and**
x-assn = ‹*word64-assn*› **and**
P = ‹λ-. *True*›
rewrites
‹*read-heur-wl-heur* (*RETURN o next-pure-lits-schedule*) = *RETURN o next-pure-lits-schedule-st*›
and
‹*read-heur-wl-heur-code next-pure-lits-schedule-info-stats-impl* = *next-pure-lits-schedule-st-impl*›
apply *unfold-locales*
apply (*rule heur-refine*)
subgoal by (*auto simp*: *next-pure-lits-schedule-st-def read-all-st-def intro*!: *ext split*: *isasat-int-splits*)
subgoal by (*auto simp*: *next-pure-lits-schedule-st-impl-def*)
done

definition *next-reduce-schedule-st-impl* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*next-reduce-schedule-st-impl* = *read-heur-wl-heur-code next-reduce-schedule-info-stats-impl*›

global-interpretation *next-reduce-schedule*: *read-heur-param-adder0* **where**
f' = ‹*RETURN o next-reduce-schedule*› **and**
f = *next-reduce-schedule-info-stats-impl* **and**
x-assn = ‹*word64-assn*› **and**
P = ‹λ-. *True*›
rewrites
‹*read-heur-wl-heur* (*RETURN o next-reduce-schedule*) = *RETURN o next-reduce-schedule-st*› **and**
‹*read-heur-wl-heur-code next-reduce-schedule-info-stats-impl* = *next-reduce-schedule-st-impl*›
apply *unfold-locales*
apply (*rule heur-refine*)
subgoal by (*auto simp*: *next-reduce-schedule-st-def read-all-st-def intro*!: *ext split*: *isasat-int-splits*)
subgoal by (*auto simp*: *next-reduce-schedule-st-impl-def*)
done

lemmas [*sepref-fr-rules*] =
wasted-of.refine
current-restart-phase.refine
next-pure-lits-schedule.refine
next-reduce-schedule.refine

lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
wasted-bytes-st-impl-def[*unfolded read-all-st-code-def*]
get-restart-phase-imp-def[*unfolded read-all-st-code-def*]
next-pure-lits-schedule-st-impl-def[*unfolded read-all-st-code-def*]
next-reduce-schedule-st-impl-def[*unfolded read-all-st-code-def*]

sepref-register *set-zero-wasted mop-save-phase-heur add-lbd*

sepref-register *isa-trail-nth isasat-trail-nth-st*

sepref-def *isa-trail-nth-impl*
is ‹*uncurry isa-trail-nth*›
:: ‹*trail-pol-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\to_a$ *unat-lit-assn*›
unfolding *trail-pol-fast-assn-def isa-trail-nth-def*
by *sepref*

definition *isasat-trail-nth-st-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*isasat-trail-nth-st-code* = (λ*N C*. *read-trail-wl-heur-code* (λ*M*. *isa-trail-nth-impl M C*) *N*)›

global-interpretation *trail-nth*: *read-trail-param-adder* **where**
R = ‹*snat-rel*′ *TYPE*(*64*)› **and**
f' = ‹λ*C M*. *isa-trail-nth M C*› **and**
f = ‹λ*C M*. *isa-trail-nth-impl M C*› **and**
x-assn = *unat-lit-assn* **and**
P = ‹λ- -. *True*›
rewrites
‹(λ*N C*′. *read-trail-wl-heur* (λ*M*. *isa-trail-nth M C*′) *N*) = *isasat-trail-nth-st*› **and**
‹(λ*N C*. *read-trail-wl-heur-code* (λ*M*. *isa-trail-nth-impl M C*) *N*) = *isasat-trail-nth-st-code*›
apply *unfold-locales*
apply (*subst lambda-comp-true*)
apply (*rule isa-trail-nth-impl.refine*)

subgoal by (*auto simp*: *isasat-trail-nth-st-def read-all-st-def isasat-length-trail-st-def*
intro!: *ext split*: *isasat-int-splits*)
subgoal by (*auto simp*: *isasat-trail-nth-st-code-def*)
done

lemma *trail-nth-precond-simp*: ‹(λ*M*. *fst M* ≠ []) = (λ(*M*,-). *M* ≠ [])›
by *auto*
definition *lit-of-hd-trail-st-heur-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*lit-of-hd-trail-st-heur-fast-code* = *read-trail-wl-heur-code lit-of-last-trail-fast-code*›

global-interpretation *last-trail*: *read-trail-param-adder0* **where**
f' = ‹*RETURN o lit-of-last-trail-pol*› **and**
f = ‹*lit-of-last-trail-fast-code*› **and**
x-assn = *unat-lit-assn* **and**
P = ‹λ*M*. *fst M* ≠ []›
rewrites
‹*last-trail.mop* = *lit-of-hd-trail-st-heur*› **and**
‹*read-trail-wl-heur-code lit-of-last-trail-fast-code* = *lit-of-hd-trail-st-heur-fast-code*›
apply *unfold-locales*
apply (*subst trail-nth-precond-simp*)
apply (*rule lit-of-last-trail-fast-code.refine*)
subgoal by (*auto simp*: *lit-of-hd-trail-st-heur-def lit-of-last-trail-pol-def read-all-st-def read-trail-param-adder0-ops.mop-*
intro!: *ext split*: *isasat-int-splits*)
subgoal by (*auto simp*: *lit-of-hd-trail-st-heur-fast-code-def*)
done

definition *get-the-propagation-reason-pol-st-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*get-the-propagation-reason-pol-st-code* = (λ*N C*. *read-trail-wl-heur-code* (λ*M*. *get-the-propagation-reason-fast-code*
M C) *N*)›

global-interpretation *propagation-reason*: *read-trail-param-adder* **where**
R = *unat-lit-rel* **and**
f' = ‹λ*C M*. *get-the-propagation-reason-pol M C*› **and**
f = ‹λ*C M*. *get-the-propagation-reason-fast-code M C*› **and**
x-assn = ‹*snat-option-assn'* *TYPE*(*64*)› **and**
P = ‹λ*M* -. *True*›
rewrites
‹(λ*M C*. *read-trail-wl-heur* (λ*M*. *get-the-propagation-reason-pol M C*) *M*) = *get-the-propagation-reason-pol-st*›
and
‹(λ*N C*. *read-trail-wl-heur-code* (λ*M*. *get-the-propagation-reason-fast-code M C*) *N*) = *get-the-propagation-reason-pol-st*
apply *unfold-locales*
apply (*subst lambda-comp-true*)
apply (*rule get-the-propagation-reason-fast-code.refine*)
subgoal by (*auto simp*: *get-the-propagation-reason-pol-st-def lit-of-last-trail-pol-def read-all-st-def read-trail-param-adder*
intro!: *ext split*: *isasat-int-splits*)
subgoal by (*auto simp*: *get-the-propagation-reason-pol-st-code-def*)
done

definition *is-fully-propagated-heur-st-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*is-fully-propagated-heur-st-code* = *read-heur-wl-heur-code is-fully-propagated-heur-stats-impl*›

global-interpretation *is-fully-proped*: *read-heur-param-adder0* **where**
f' = ‹*RETURN o is-fully-propagated-heur*› **and**
f = ‹*is-fully-propagated-heur-stats-impl*› **and**
x-assn = *bool1-assn* **and**
P = ‹λ-. *True*›

rewrites
‹*read-heur-wl-heur* (*RETURN o is-fully-propagated-heur*) = *RETURN o is-fully-propagated-heur-st*›
and
‹*read-heur-wl-heur-code is-fully-propagated-heur-stats-impl* = *is-fully-propagated-heur-st-code*›
apply *unfold-locales*
apply (*rule heur-refine*)
subgoal by (*auto simp*: *is-fully-propagated-heur-def is-fully-propagated-heur-st-def read-all-st-def*
intro!: *ext split*: *isasat-int-splits*)
subgoal by (*auto simp*: *is-fully-propagated-heur-st-code-def*)
done

definition *heuristic-reluctant-triggered2-st-impl* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*heuristic-reluctant-triggered2-st-impl* = *read-heur-wl-heur-code heuristic-reluctant-triggered2-stats-impl*›

global-interpretation *heuristic-reluctant-triggered2*: *read-heur-param-adder0* **where**
f' = ‹*RETURN o heuristic-reluctant-triggered2*› **and**
f = *heuristic-reluctant-triggered2-stats-impl* **and**
x-assn = ‹*bool1-assn*› **and**
P = ‹(λ-. *True*)›
rewrites
‹*read-heur-wl-heur* (*RETURN o heuristic-reluctant-triggered2*) = *RETURN o heuristic-reluctant-triggered2-st*›
and
‹*read-heur-wl-heur-code heuristic-reluctant-triggered2-stats-impl* = *heuristic-reluctant-triggered2-st-impl*›
apply *unfold-locales*
apply (*rule heur-refine*)
subgoal by (*auto simp*: *read-all-st-def heuristic-reluctant-triggered2-st-def heuristic-reluctant-triggered2-def*
intro!: *ext split*: *isasat-int-splits*)
subgoal by (*auto simp*: *heuristic-reluctant-triggered2-st-impl-def*)
done

lemma *heuristic-reluctant-untrigger-st-alt-def*:
‹*heuristic-reluctant-untrigger-st S* =
(*let* (*heur*, *S*) = *extract-heur-wl-heur S*;
heur = *heuristic-reluctant-untrigger heur*;
S = *update-heur-wl-heur heur S in*
S)›
by (*auto simp*: *heuristic-reluctant-untrigger-st-def state-extractors split*: *isasat-int-splits*
intro!: *ext*)

sepref-def *heuristic-reluctant-untrigger-st-impl*
is ‹*RETURN o heuristic-reluctant-untrigger-st*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
unfolding *heuristic-reluctant-untrigger-st-alt-def*
by *sepref*

lemmas [*sepref-fr-rules*] =
trail-nth.refine[*unfolded lambda-comp-true*]
last-trail.mop-refine
is-fully-proped.refine
propagation-reason.refine
heuristic-reluctant-triggered2.refine

lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
isasat-trail-nth-st-code-def[*unfolded read-all-st-code-def*]
lit-of-hd-trail-st-heur-fast-code-def[*unfolded read-all-st-code-def*]
is-fully-propagated-heur-st-code-def[*unfolded read-all-st-code-def*]

get-the-propagation-reason-pol-st-code-def[*unfolded read-all-st-code-def*]
heuristic-reluctant-triggered2-st-impl-def[*unfolded read-all-st-code-def*]

sepref-register *incr-restart-stat clss-size-lcountUE clss-size-lcountUS learned-clss-count clss-size-allcount*

lemma *incr-restart-stat-alt-def*:
‹*incr-restart-stat* = (λ*S*. *do*{
let (*heur*, *S*) = *extract-heur-wl-heur S*;
let heur = *unset-fully-propagated-heur* (*heuristic-reluctant-untrigger* (*restart-info-restart-done-heur heur*));
let S = *update-heur-wl-heur heur S*;
let (*stats*, *S*) = *extract-stats-wl-heur S*;
let stats = *incr-restart* (*stats*);
let S = *update-stats-wl-heur stats S*;
RETURN S
})›
by (*auto simp*: *incr-restart-stat-def state-extractors split*: *isasat-int-splits*
intro!: *ext*)

sepref-def *incr-restart-stat-fast-code*
is ‹*incr-restart-stat*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *incr-restart-stat-alt-def*
by *sepref*

sepref-register *incr-reduction-stat clss-size-decr-lcount*
clss-size-incr-lcountUE clss-size-incr-lcountUS

lemma *incr-reduction-stat-alt-def*:
‹*incr-reduction-stat* = (λ*S*. *do*{
let (*stats*, *S*) = *extract-stats-wl-heur S*;
let stats = *incr-reduction stats*;
let S = *update-stats-wl-heur stats S*;
RETURN S
})›
by (*auto simp*: *incr-reduction-stat-def state-extractors split*: *isasat-int-splits*
intro!: *ext*)

sepref-def *incr-reduction-stat-fast-code*
is ‹*incr-reduction-stat*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *incr-reduction-stat-alt-def*
by *sepref*

sepref-register *mark-unused-st-heur*
lemma *mark-unused-st-heur-alt-def*:
‹*RETURN oo mark-unused-st-heur* = (λ*C* S_0. *do* {
let (*N*, *S*) = *extract-arena-wl-heur* S_0;
ASSERT (*N* = *get-clauses-wl-heur* S_0);
let N′ = *mark-unused N C*;
let S = *update-arena-wl-heur N*′ *S*;
RETURN S})›

by (*auto simp*: *mark-unused-st-heur-def state-extractors Let-def intro*!: *ext split*: *isasat-int-splits*)

sepref-def *mark-unused-st-fast-code*
is ‹*uncurry* (*RETURN oo mark-unused-st-heur*)›
:: ‹[λ(*C*, *S*). *arena-act-pre* (*get-clauses-wl-heur S*) *C*]$_a$
sint64-nat-assn^k $*_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
unfolding *mark-unused-st-heur-alt-def*
arena-act-pre-mark-used[*intro*!]
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-def *mop-mark-unused-st-heur-impl*
is ‹*uncurry mop-mark-unused-st-heur*›
:: ‹ *sint64-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
unfolding *mop-mark-unused-st-heur-def fold-tuple-optimizations*
by *sepref*

sepref-register *get-the-propagation-reason-pol-st*
lemma *empty-US-heur-alt-def*:
‹*empty-US-heur S* =
(*let* (*lcount*, *S*) = *extract-lcount-wl-heur S in*
let lcount = *clss-size-resetUS0 lcount in*
let S = *update-lcount-wl-heur lcount S in S*
)›
by (*auto simp*: *empty-US-heur-def state-extractors Let-def intro*!: *ext split*: *isasat-int-splits*)

sepref-def *empty-US-heur-code*
is ‹*RETURN o empty-US-heur*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
unfolding *empty-US-heur-alt-def*
by *sepref*

lemma *mark-garbage-heur2-alt-def*:
‹*mark-garbage-heur2 C* = (λS_0. *do*{
ASSERT (*mark-garbage-pre* (*get-clauses-wl-heur* S_0, *C*));
let (*N*, *S*) = *extract-arena-wl-heur* S_0;
ASSERT (*N* = *get-clauses-wl-heur* S_0);
let st = *arena-status N C* = *IRRED*;
let N' = *extra-information-mark-to-delete* (*N*) *C*;
let (*lcount*, *S*) = *extract-lcount-wl-heur S*;
ASSERT (*lcount* = *get-learned-count* S_0);
ASSERT(¬*st* ⟶ *clss-size-lcount lcount* ≥ *1*);
let lcount = (*if st then lcount else clss-size-decr-lcount lcount*);
RETURN (*update-lcount-wl-heur lcount* (*update-arena-wl-heur N' S*))})›
by (*auto simp*: *mark-garbage-heur2-def state-extractors Let-def intro*!: *ext split*: *isasat-int-splits*)

lemma *mark-garbage-preD*:
‹*mark-garbage-pre* (*N*, *C*) ⟹ *arena-is-valid-clause-vdom N C*›
by (*auto simp*: *mark-garbage-pre-def arena-is-valid-clause-idx-def arena-is-valid-clause-vdom-def*)

sepref-register *mark-garbage-heur2 mark-garbage-heur4*

sepref-def *mark-garbage-heur2-code*
is ‹*uncurry mark-garbage-heur2*›
:: ‹[λ(*C*, *S*). *True*]$_a$ *sint64-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›

supply [[*goals-limit=1*]]
supply [*intro*] = *mark-garbage-preD*
unfolding *mark-garbage-heur2-alt-def*
by *sepref*

lemma *mark-garbage-heur4-alt-def*:
‹*mark-garbage-heur4 C* S_0 = *do*{
let (*N'*, *S*) = *extract-arena-wl-heur* S_0;
ASSERT (*N'* = *get-clauses-wl-heur* S_0);
let st = *arena-status N' C* = *IRRED*;
let N' = *extra-information-mark-to-delete* (*N'*) *C*;
let (*lcount*, *S*) = *extract-lcount-wl-heur S*;
ASSERT (*lcount* = *get-learned-count* S_0);
ASSERT(¬*st* ⟶ *clss-size-lcount lcount* ≥ *1*);
let lcount = (*if st then lcount else clss-size-incr-lcountUEk* (*clss-size-decr-lcount lcount*));
let (*stats*, *S*) = *extract-stats-wl-heur S*;
ASSERT (*stats* = *get-stats-heur* S_0);
let stats = (*if st then decr-irred-clss stats else stats*);
let S = *update-arena-wl-heur N' S*;
let S = *update-lcount-wl-heur lcount S*;
let S = *update-stats-wl-heur stats S*;
RETURN S
}›
by (*cases* S_0)
(*auto simp*: *mark-garbage-heur4-def state-extractors Let-def intro*!: *ext split*: *isasat-int-splits*)

sepref-def *mark-garbage-heur4-code*
is ‹*uncurry mark-garbage-heur4*›
:: ‹[λ(*C*, *S*). *mark-garbage-pre* (*get-clauses-wl-heur S*, *C*) ∧ *arena-is-valid-clause-vdom* (*get-clauses-wl-heur S*) *C* ∧
learned-clss-count S ≤ *unat64-max*]$_a$
sint64-nat-assn^k *$*_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
supply [[*goals-limit=1*]] *isasat-fast-countD*[*dest*] *learned-clss-count-def*[*simp*]
unfolding *mark-garbage-heur4-alt-def*
by *sepref*

sepref-definition *access-avdom-aivdom-at-impl*
is ‹*uncurry* (λ*N C*. *RETURN* (*get-avdom-aivdom N* ! *C*))›
:: ‹[*uncurry* (λ*N C*. *C* < *length* (*get-avdom-aivdom N*))]$_a$ *aivdom-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} → *sint64-nat-assn*›
unfolding *avdom-aivdom-at-def*[*symmetric*]
by *sepref*

definition *access-avdom-at-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*access-avdom-at-fast-code* = (λ*N C*. *read-vdom-wl-heur-code* (λ*N*. *avdom-aivdom-at-impl N C*) *N*)›

global-interpretation *avdom-aivdom-at*: *read-vdom-param-adder* **where**
R = ‹*snat-rel'* *TYPE*(*64*)› **and**
x-assn = *sint64-nat-assn* **and**
f' = ‹λ*C N*. (*RETURN oo avdom-aivdom-at*) *N C*› **and**
f = ‹λ*C N*. *avdom-aivdom-at-impl N C*› **and**
P = ‹λ*C N*. *C* < *length* (*get-avdom-aivdom N*)›
rewrites
‹(λ*N C'*. *read-vdom-wl-heur* (λ*N*. (*RETURN* ∘∘ *avdom-aivdom-at*) *N C'*) *N*) = *RETURN oo access-avdom-at*› **and**

‹(λN C. read-vdom-wl-heur-code (λN. avdom-aivdom-at-impl N C) N) = access-avdom-at-fast-code›
and
‹(λS C. C < length (get-avdom-aivdom (get-aivdom S))) = access-avdom-at-pre›
apply *unfold-locales*
apply (*subst* (*3*) *uncurry-def*)
apply (*rule avdom-aivdom-at-impl-refine*)
subgoal by (*auto simp*: *access-avdom-at-def read-all-st-def avdom-aivdom-at-def split*: *isasat-int-splits intro*!: *ext*)
subgoal by (*auto simp*: *access-avdom-at-fast-code-def*)
subgoal by (*auto simp* :*access-avdom-at-pre-def split*: *isasat-int-splits intro*!: *ext*)
done

definition *access-ivdom-at-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*access-ivdom-at-fast-code* = (λN C. *read-vdom-wl-heur-code* (λN. *ivdom-aivdom-at-impl N C*) N)›

global-interpretation *ivdom-aivdom-at*: *read-vdom-param-adder* **where**
R = ‹*snat-rel*′ *TYPE*(*64*)› **and**
x-assn = *sint64-nat-assn* **and**
f′ = ‹λC N. (*RETURN oo ivdom-aivdom-at*) N C› **and**
f = ‹λC N. *ivdom-aivdom-at-impl N C*› **and**
P = ‹λC N. C < *length* (*get-ivdom-aivdom N*)›
rewrites
‹(λN C′. *read-vdom-wl-heur* (λN. (*RETURN* ∘∘ *ivdom-aivdom-at*) N C′) N) = *RETURN oo access-ivdom-at*› **and**
‹(λN C. *read-vdom-wl-heur-code* (λN. *ivdom-aivdom-at-impl N C*) N) = *access-ivdom-at-fast-code*›
and
‹(λS C. C < *length* (*get-ivdom-aivdom* (*get-aivdom S*))) = *access-ivdom-at-pre*›
apply *unfold-locales*
apply (*subst* (*3*) *uncurry-def*)
apply (*rule ivdom-aivdom-at-impl-refine*)
subgoal by (*auto simp*: *access-ivdom-at-def read-all-st-def ivdom-aivdom-at-def split*: *isasat-int-splits intro*!: *ext*)
subgoal by (*auto simp*: *access-ivdom-at-fast-code-def*)
subgoal by (*auto simp* :*access-ivdom-at-pre-def split*: *isasat-int-splits intro*!: *ext*)
done

definition *access-tvdom-at-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*access-tvdom-at-fast-code* = (λN C. *read-vdom-wl-heur-code* (λN. *tvdom-aivdom-at-impl N C*) N)›

global-interpretation *tvdom-aivdom-at*: *read-vdom-param-adder* **where**
R = ‹*snat-rel*′ *TYPE*(*64*)› **and**
x-assn = *sint64-nat-assn* **and**
f′ = ‹λC N. (*RETURN oo tvdom-aivdom-at*) N C› **and**
f = ‹λC N. *tvdom-aivdom-at-impl N C*› **and**
P = ‹λC N. C < *length* (*get-tvdom-aivdom N*)›
rewrites
‹(λN C′. *read-vdom-wl-heur* (λN. (*RETURN* ∘∘ *tvdom-aivdom-at*) N C′) N) = *RETURN oo access-tvdom-at*› **and**
‹(λN C. *read-vdom-wl-heur-code* (λN. *tvdom-aivdom-at-impl N C*) N) = *access-tvdom-at-fast-code*›
and
‹(λS C. C < *length* (*get-tvdom-aivdom* (*get-aivdom S*))) = *access-tvdom-at-pre*›
apply *unfold-locales*
apply (*subst* (*3*) *uncurry-def*)
apply (*rule tvdom-aivdom-at-impl-refine*)
subgoal by (*auto simp*: *access-tvdom-at-def read-all-st-def tvdom-aivdom-at-def split*: *isasat-int-splits intro*!: *ext*)

subgoal by (*auto simp*: *access-tvdom-at-fast-code-def*)
subgoal by (*auto simp* :*access-tvdom-at-pre-def split*!: *isasat-int-splits intro*!: *ext*)
done

lemmas [*sepref-fr-rules*] =
avdom-aivdom-at.refine
ivdom-aivdom-at.refine
tvdom-aivdom-at.refine
lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
access-avdom-at-fast-code-def[*unfolded read-all-st-code-def*]
access-ivdom-at-fast-code-def[*unfolded read-all-st-code-def*]
access-tvdom-at-fast-code-def[*unfolded read-all-st-code-def*]

sepref-register *mop-access-lit-in-clauses-heur mop-watched-by-app-heur*
get-target-opts get-opts

sepref-register *print-literal-of-trail*
print-trail print-trail-st print-trail-st2

sepref-def *print-literal-of-trail-code*
is *print-literal-of-trail*
:: ‹*unat-lit-assn*^{k} $\rightarrow_a$ *unit-assn*›
unfolding *print-literal-of-trail-def*
by *sepref*

sepref-def *print-encoded-lit-end-code*
is *print-literal-of-trail*
:: ‹*uint32-nat-assn*^{k} $\rightarrow_a$ *unit-assn*›
unfolding *print-literal-of-trail-def*
by *sepref*

sepref-def *print-trail-code*
is ‹*print-trail*›
:: ‹*trail-pol-fast-assn*^{k} $\rightarrow_a$ *unit-assn*›
unfolding *print-trail-def trail-pol-fast-assn-def*
apply (*rewrite at* ‹*print-literal-of-trail* (⌑)› *unat-const-fold*[**where** *'a=32*])
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemmas *print-trail*[*sepref-fr-rules*] =
print-trail-code.refine[*FCOMP print-trail-print-trail2-rel*]

definition *print-trail-st-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*print-trail-st-code* = *read-trail-wl-heur-code print-trail-code*›

global-interpretation *print-trail*: *read-trail-param-adder0* **where**
f' = *print-trail* **and**
f = *print-trail-code* **and**
x-assn = *unit-assn* **and**
P = ‹λ -. *True*›
rewrites
‹*read-trail-wl-heur print-trail* = *print-trail-st*› **and**
‹*read-trail-wl-heur-code print-trail-code* = *print-trail-st-code*›

apply *unfold-locales*
apply (*rule print-trail-code.refine*)
subgoal by (*auto simp*: *print-trail-st-def read-all-st-def print-trail-def*
intro!: *ext split*: *isasat-int-splits*)
subgoal by (*auto simp*: *print-trail-st-code-def*)
done

lemmas [*sepref-fr-rules*] =
print-trail.refine[*FCOMP print-trail-st-print-trail-st2-rel*]

lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
print-trail-st-code-def[*unfolded read-all-st-code-def*]
sepref-register *is-fully-propagated-heur-st*

lemma [*def-pat-rules*]: ‹*nth-rll* ≡ *op-list-list-idx*›
by (*auto simp*: *nth-rll-def intro*!: *ext eq-reflection*)

definition *access-watchlist* :: ‹(*nat* × *nat literal* × *bool*) *list list* ⇒ -› **where**
‹*access-watchlist N C C′* = *nth-rll N* (*nat-of-lit C*) *C′*›

sepref-def *access-watchlist-impl*
is ‹*uncurry2* (*RETURN ooo access-watchlist*)›
:: ‹[*uncurry2* (λ*S L K*. *nat-of-lit L* < *length S* ∧
K < *length* (*S* ! *nat-of-lit L*))]$_a$
watchlist-fast-assn^k *$*_a$ *unat-lit-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} → *watcher-fast-assn*›
unfolding *access-watchlist-def*
by *sepref*

lemma *watched-by-app-helper*:
‹*uncurry* (λ*NC D*. *uncurry* (λ*N C*. *access-watchlist-impl N C D*) *NC*) = *uncurry2 access-watchlist-impl*›
‹*uncurry* (λ*NC D′*. *uncurry* (λ*N C′*. (*RETURN* ∘∘∘ *access-watchlist*) *N C′ D′*) *NC*) = *uncurry2*
(*RETURN ooo access-watchlist*)›
‹*uncurry* (λ*a b*. *uncurry* (λ*a c*. *nat-of-lit c* < *length a* ∧ *b* < *length* (*a* ! *nat-of-lit c*)) *a*)=
uncurry2 (λ*S L K*. *nat-of-lit L* < *length S* ∧ *K* < *length* (*S* ! *nat-of-lit L*))›
by (*auto*)

definition *watched-by-app-heur-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*watched-by-app-heur-fast-code* = (λ*N C D*. *read-watchlist-wl-heur-code* (λ*N*. *access-watchlist-impl N*
C D) *N*)›

global-interpretation *watched-by-app*: *read-watchlist-param-adder-twoargs* **where**
R = ‹*unat-lit-rel*› **and**
R′ = ‹*snat-rel′ TYPE*(*64*)› **and**
f = ‹λ*C C′ N*. *access-watchlist-impl N C C′*› **and**
f′ = ‹λ*C C′ N*. (*RETURN ooo access-watchlist*) *N C C′*› **and**
x-assn = *watcher-fast-assn* **and**
P = ‹(λ*L K S*. *nat-of-lit L* < *length S* ∧
K < *length* (*S* ! *nat-of-lit L*))›
rewrites
‹(λ*N C′ D′*. *read-watchlist-wl-heur* (λ*N*. (*RETURN* ∘∘∘ *access-watchlist*) *N C′ D′*) *N*) = *RETURN*
ooo watched-by-app-heur› **and**
‹(λ*N C D*. *read-watchlist-wl-heur-code* (λ*N*. *access-watchlist-impl N C D*) *N*) = *watched-by-app-heur-fast-code*›
and
‹*uncurry2* (λ*S C D*. *nat-of-lit C* < *length* (*get-watched-wl-heur S*) ∧ *D* < *length* (*get-watched-wl-heur*

S ! nat-of-lit C)) = watched-by-app-heur-pre›

apply *unfold-locales*
unfolding *watched-by-app-helper*
apply (*rule access-watchlist-impl.refine*)
subgoal
by (*auto intro*!: *ext split*: *tuple17.split*
simp: *read-all-st-def watched-by-app-heur-def access-watchlist-def*
nth-rll-def)
subgoal by (*auto simp*: *watched-by-app-heur-fast-code-def*)
subgoal by (*auto simp*: *watched-by-app-heur-pre-def*)
done

lemma *mop-watched-by-app-heur-alt-def*: ‹*mop-watched-by-app-heur* = (λ*N C' D'*. *watched-by-app.XX.XX.mop N* (*C'*, *D'*))›
by (*auto simp*: *mop-watched-by-app-heur-def read-all-param-adder-ops.mop-def summarize-ASSERT-conv read-all-st-def access-watchlist-def conj-commute nth-rll-def intro*!: *ext intro*: *bind-cong split*: *isasat-int-splits*)
definition *mop-watched-by-app-heur-fast-impl* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*mop-watched-by-app-heur-fast-impl* = (λ*N C D*. *read-watchlist-wl-heur-code* (*case* (*C*, *D*) *of* (*C*, *D*) ⇒ λ*N*. *access-watchlist-impl N C D*) *N*) ›
lemma *split-snd-pure-arg'*:
assumes ‹(*uncurry* (λ*N C*. *f C N*), *uncurry* (λ*N C'*. *f' C' N*))
∈ [λ-. *True*]$_a$ K^k *$*$$_a$ (*pure* (*R* ×$_f$ *R'*))^k → *x-assn*›
shows ‹(*uncurry2* (λ*N C D*. *f* (*C*,*D*) *N*), *uncurry2* (λ*N C' D'*. *f'* (*C'*,*D'*) *N*))
∈ [λ-. *True*]$_a$ K^k *$*$$_a$ (*pure*(*R*))^k *$*$$_a$ (*pure R'*)^k → *x-assn*›
using *assms* **unfolding** *hfref-def*
by (*auto simp flip*: *prod-assn-pure-conv*)

lemmas [*sepref-fr-rules*] =
watched-by-app.refine
watched-by-app.mop-refine[*THEN split-snd-pure-arg'*, *unfolded mop-watched-by-app-heur-alt-def*[*symmetric*] *mop-watched-by-app-heur-fast-impl-def*[*symmetric*]]

lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
watched-by-app-heur-fast-code-def[*unfolded read-all-st-code-def*]
mop-watched-by-app-heur-fast-impl-def[*unfolded read-all-st-code-def prod.case*]

definition *mop-is-marked-added-heur-stats-st-impl* **where**
‹*mop-is-marked-added-heur-stats-st-impl* =
(λ*N A*. *read-heur-wl-heur-code* (λ*S*. *mop-is-marked-added-heur-stats-impl S A*) *N*)›

global-interpretation *is-marked-added*: *read-heur-param-adder* **where**
R = *atom-rel* **and**
f' = ‹ λ*S A*. *RETURN* (*is-marked-added-heur A S*)› **and**
f = ‹ λ*S A*. *mop-is-marked-added-heur-stats-impl A S*› **and**
x-assn = ‹*bool1-assn*› **and**
P = ‹(λ*S A*. *is-marked-added-heur-pre A S*)›
rewrites
‹(λ*N A*. *read-heur-wl-heur-code* (λ*S*. *mop-is-marked-added-heur-stats-impl S A*) *N*) = *mop-is-marked-added-heur-stats-st*
apply *unfold-locales*
unfolding *lambda-comp-true*
apply (*unfold uncurry-def*, *rule is-marked-added-heur-refine*[*unfolded comp-def uncurry-def*])
apply (*subst mop-is-marked-added-heur-stats-st-impl-def*, *rule refl*)
done

lemma *mop-is-marked-added-heur-st-alt-def*:

‹*is-marked-added.XX.mop* = *mop-is-marked-added-heur-st*›
unfolding *is-marked-added.XX.mop-def mop-is-marked-added-heur-st-def*
mop-is-marked-added-heur-def
apply (*intro ext*, *case-tac S*)
apply (*auto simp*: *read-all-st-def intro*!: *ext*)
done

lemmas [*sepref-fr-rules*] = *is-marked-added.XX.mop-refine*[*unfolded mop-is-marked-added-heur-st-alt-def*]
lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
mop-is-marked-added-heur-stats-st-impl-def[*unfolded read-all-st-code-def*]

definition *length-watchlist-raw* **where**
‹*length-watchlist-raw S* = *length* (*get-watched-wl-heur S*)›

sepref-def *length-watchlist-full-impl*
is ‹*RETURN o length*›
:: ‹*watchlist-fast-assn*^{k} →$_a$ *sint64-nat-assn*›
unfolding *op-list-list-len-def*[*symmetric*]
by *sepref*

definition *length-watchlist-raw-code* **where**
‹*length-watchlist-raw-code* = *read-watchlist-wl-heur-code* (*length-watchlist-full-impl*)›

global-interpretation *watchlist-length-raw*: *read-watchlist-param-adder0* **where**
f' = ‹*RETURN o length*› **and**
f = ‹*length-watchlist-full-impl*› **and**
x-assn = *sint64-nat-assn* **and**
P = ‹(λ-. *True*)›
rewrites
‹*read-watchlist-wl-heur* (*RETURN* ∘ *length*) = *RETURN o length-watchlist-raw*› **and**
‹*read-watchlist-wl-heur-code* (*length-watchlist-full-impl*) = *length-watchlist-raw-code*›
apply *unfold-locales*
apply (*rule length-watchlist-full-impl.refine*)
subgoal
by (*auto intro*!: *ext simp*: *length-watchlist-raw-def read-all-st-def length-watchlist-def*
length-ll-def
split: *isasat-int-splits*)
subgoal by (*auto simp*: *length-watchlist-raw-code-def*)
done

lemmas [*sepref-fr-rules*] = *watchlist-length-raw.refine*
lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
length-watchlist-raw-code-def[*unfolded read-all-st-code-def*]

definition *get-restart-count-st* **where**
‹*get-restart-count-st S* = *get-restart-count* (*get-stats-heur S*)›

definition *get-restart-count-st-impl* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*get-restart-count-st-impl* = *read-stats-wl-heur-code get-restart-count-impl*›

global-interpretation *restart-count*: *read-stats-param-adder0* **where**
f' = ‹*RETURN o get-restart-count*› **and**
f = *get-restart-count-impl* **and**
x-assn = *word-assn* **and**
P = ‹λ-. *True*›
rewrites ‹*read-stats-wl-heur* (*RETURN o get-restart-count*) = *RETURN o get-restart-count-st*› **and**

‹read-stats-wl-heur-code get-restart-count-impl = get-restart-count-st-impl›
apply *unfold-locales*
apply (*rule get-restart-count-impl.refine*; *assumption*)
subgoal by (*auto simp*: *read-all-st-def stats-conflicts-def get-restart-count-st-def intro*!: *ext*
split: *isasat-int-splits*)
subgoal by (*auto simp*: *get-restart-count-st-impl-def*)
done

definition *get-reductions-count-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*get-reductions-count-fast-code* = *read-stats-wl-heur-code get-reduction-count-impl*›

global-interpretation *reduction-count*: *read-stats-param-adder0* **where**
f' = ‹*RETURN o get-reduction-count*› **and**
f = *get-reduction-count-impl* **and**
x-assn = *word-assn* **and**
P = ‹λ-. *True*›
rewrites ‹*read-stats-wl-heur* (*RETURN o get-reduction-count*) = *RETURN o get-reductions-count*›
and
‹*read-stats-wl-heur-code get-reduction-count-impl* = *get-reductions-count-fast-code*›
apply *unfold-locales*
apply (*rule get-reduction-count-impl.refine*)
subgoal by (*auto simp*: *read-all-st-def stats-conflicts-def intro*!: *ext*
split: *isasat-int-splits*)
subgoal by (*auto simp*: *get-reductions-count-fast-code-def*)
done

definition *get-irredundant-count-st-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*get-irredundant-count-st-code* = *read-stats-wl-heur-code get-irredundant-count-impl*›

global-interpretation *irredandant-count*: *read-stats-param-adder0* **where**
f' = ‹*RETURN o get-irredundant-count*› **and**
f = *get-irredundant-count-impl* **and**
x-assn = *word-assn* **and**
P = ‹λ-. *True*›
rewrites ‹*read-stats-wl-heur* (*RETURN o get-irredundant-count*) = *RETURN o get-irredundant-count-st*›
and
‹*read-stats-wl-heur-code get-irredundant-count-impl* = *get-irredundant-count-st-code*›
apply *unfold-locales*
apply (*rule get-irredundant-count-impl.refine*)
subgoal by (*auto simp*: *read-all-st-def get-irredundant-count-st-def stats-conflicts-def intro*!: *ext*
split: *isasat-int-splits*)
subgoal by (*auto simp*: *get-irredundant-count-st-code-def*)
done

definition *get-slow-ema-heur-full* **where**
‹*get-slow-ema-heur-full S* = *ema-get-value* (*slow-ema-of S*)›
definition *get-fast-ema-heur-full* **where**
‹*get-fast-ema-heur-full S* = *ema-get-value* (*fast-ema-of S*)›

sepref-def *get-slow-ema-heur-full-impl*
is ‹*RETURN o get-slow-ema-heur-full*›
:: ‹*heuristic-assn*^{k} →$_a$ *word64-assn*›
unfolding *get-slow-ema-heur-full-def*
by *sepref*

sepref-def *get-fast-ema-heur-full-impl*
  is ‹*RETURN o get-fast-ema-heur-full*›
  :: ‹*heuristic-assn*^{k} →$_a$ *word64-assn*›
  unfolding *get-fast-ema-heur-full-def*
  by *sepref*

definition *get-slow-ema-heur-st* **where**
  ‹*get-slow-ema-heur-st S* = *ema-get-value* (*get-slow-ema-heur S*)›
definition *get-fast-ema-heur-st* **where**
  ‹*get-fast-ema-heur-st S* = *ema-get-value* (*get-fast-ema-heur S*)›

definition *get-slow-ema-heur-st-impl* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
  ‹*get-slow-ema-heur-st-impl* = *read-heur-wl-heur-code get-slow-ema-heur-full-impl*›

definition *get-fast-ema-heur-st-impl* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
  ‹*get-fast-ema-heur-st-impl* = *read-heur-wl-heur-code get-fast-ema-heur-full-impl*›

global-interpretation *slow-ema*: *read-heur-param-adder0* **where**
  f' = ‹*RETURN o get-slow-ema-heur-full*› **and**
  f = *get-slow-ema-heur-full-impl* **and**
  x-assn = *word-assn* **and**
  P = ‹λ-. *True*›
 rewrites ‹*read-heur-wl-heur* (*RETURN o get-slow-ema-heur-full*) = *RETURN o get-slow-ema-heur-st*›
and
    ‹*read-heur-wl-heur-code get-slow-ema-heur-full-impl* = *get-slow-ema-heur-st-impl*›
  apply *unfold-locales*
  apply (*rule get-slow-ema-heur-full-impl.refine*)
  subgoal by (*auto simp*: *read-all-st-def stats-conflicts-def get-slow-ema-heur-st-def*
      get-slow-ema-heur-full-def intro!: *ext*
    split: *isasat-int-splits*)
  subgoal by (*auto simp*: *get-slow-ema-heur-st-impl-def*)
  done

global-interpretation *fast-ema*: *read-heur-param-adder0* **where**
  f' = ‹*RETURN o get-fast-ema-heur-full*› **and**
  f = *get-fast-ema-heur-full-impl* **and**
  x-assn = *word-assn* **and**
  P = ‹λ-. *True*›
  rewrites ‹*read-heur-wl-heur* (*RETURN o get-fast-ema-heur-full*) = *RETURN o get-fast-ema-heur-st*›
and
    ‹*read-heur-wl-heur-code get-fast-ema-heur-full-impl* = *get-fast-ema-heur-st-impl*›
  apply *unfold-locales*
  apply (*rule get-fast-ema-heur-full-impl.refine*)
  subgoal by (*auto simp*: *read-all-st-def stats-conflicts-def get-fast-ema-heur-st-def*
      get-fast-ema-heur-full-def intro!: *ext*
    split: *isasat-int-splits*)
  subgoal by (*auto simp*: *get-fast-ema-heur-st-impl-def*)
  done

lemmas [*sepref-fr-rules*] = *restart-count.refine reduction-count.refine fast-ema.refine slow-ema.refine*
lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
  get-restart-count-st-impl-def[*unfolded read-all-st-code-def*]
  get-reductions-count-fast-code-def[*unfolded read-all-st-code-def*]
  get-fast-ema-heur-st-impl-def[*unfolded read-all-st-code-def*]
  get-slow-ema-heur-st-impl-def[*unfolded read-all-st-code-def*]

end
theory *IsaSAT-Setup4-LLVM*
imports
IsaSAT-Setup
IsaSAT-Setup0-LLVM
begin

definition *length-occs* **where**
‹*length-occs S = length* (*get-occs S*)›

sepref-def *length-occs-raw*
is ‹*RETURN o length*›
:: ‹*occs-assn*^{k} →$_a$ *sint64-nat-assn*›
unfolding *op-list-list-len-def*[*symmetric*]
by *sepref*
term *op-list-list-llen*

definition *length-occs-impl* **where**
‹*length-occs-impl = read-occs-wl-heur-code length-occs-raw*›

sepref-register *length-occs*

global-interpretation *length-occs*: *read-occs-param-adder0* **where**
f = ‹*length-occs-raw*› **and**
f' = ‹*RETURN o length*› **and**
x-assn = sint64-nat-assn **and**
P = ‹(λ*S*. *True*)›
rewrites ‹*read-occs-wl-heur* (*RETURN o length*) = *RETURN o length-occs*› **and**
‹*read-occs-wl-heur-code length-occs-raw = length-occs-impl*›
apply *unfold-locales*
apply (*rule length-occs-raw.refine*)
subgoal
by (*auto simp*: *read-all-st-def length-occs-def intro*!: *ext*
split: *isasat-int-splits*)
subgoal
by (*auto simp*: *length-occs-impl-def*)
done

term *mop-cocc-list-length*

definition *length-occs-at* **where**
‹*length-occs-at S i = mop-cocc-list-length* (*get-occs S*) *i*›

sepref-def *mop-cocc-list-length-impl*
is ‹*uncurry* (*mop-cocc-list-length*)›
:: ‹[*uncurry cocc-list-length-pre*]$_a$ *occs-assn*^{k} *$_a$ *unat-lit-assn*^{k} → *sint64-nat-assn*›
unfolding *cocc-list-length-def*
cocc-list-length-pre-def fold-op-list-list-llen mop-cocc-list-length-def
by *sepref*

definition *length-occs-at-impl* **where**
‹*length-occs-at-impl* = (λ*N C*. *read-occs-wl-heur-code* (λ*M*. *mop-cocc-list-length-impl M C*) *N*)›

sepref-register *length-occs-at*

global-interpretation *length-occs-at*: *read-occs-param-adder* **where**
f = ‹λ*L S*. *mop-cocc-list-length-impl S L*› **and**
f′ = ‹λ*L S*. *mop-cocc-list-length S L*› **and**
x-assn = *sint64-nat-assn* **and**
P = ‹λ*L S*. *cocc-list-length-pre S L*› **and**
R = ‹*unat-lit-rel*›
rewrites ‹(λ*N C*. *read-occs-wl-heur-code* (λ*M*. *mop-cocc-list-length-impl M C*) *N*) = *length-occs-at-impl*›
and
‹(λ*S C′*. *read-occs-wl-heur* (λ*L*. *mop-cocc-list-length L C′*) *S*) = *length-occs-at*›
apply *unfold-locales*
apply (*rule mop-cocc-list-length-impl.refine*)
subgoal
by (*auto simp*: *length-occs-at-impl-def*)
subgoal
by (*auto simp*: *read-all-st-def length-occs-at-def intro*!: *ext*
split: *isasat-int-splits*)
done

lemma *length-occs-at-alt-def*:
‹*length-occs-at* = *length-occs-at.XX.mop*›
by (*auto simp*: *length-occs-at.XX.mop-def length-occs-at-def read-all-param-adder-ops.mop-def*
read-all-st-def summarize-ASSERT-conv
mop-cocc-list-length-def split: *tuple17.splits intro*!: *ext*)

lemmas [*sepref-fr-rules*] = *length-occs.refine*[*unfolded lambda-comp-true*]
length-occs-at.refine
length-occs-at.XX.mop-refine[*unfolded length-occs-at-alt-def*[*symmetric*]]
lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
length-occs-impl-def[*unfolded read-all-st-code-def*]
length-occs-at-impl-def[*unfolded read-all-st-code-def*]

definition *get-occs-list-at* :: ‹*isasat* ⇒ *nat literal* ⇒ *nat* ⇒ *nat nres*› **where**
‹*get-occs-list-at S L i* = *mop-cocc-list-at* (*get-occs S*) *L i*›

sepref-def *mop-cocc-list-at-impl*
is ‹*uncurry2* (*mop-cocc-list-at*)›
:: ‹[*uncurry2 cocc-list-at-pre*]$_a$ *occs-assn*^{k} ∗$_a$ *unat-lit-assn*^{k} ∗$_a$ *sint64-nat-assn*^{k} → *sint64-nat-assn*›
unfolding *mop-cocc-list-at-def cocc-list-at-def cocc-list-at-pre-def fold-op-list-list-idx*
by *sepref*

definition *get-occs-list-at-impl* :: ‹- ⇒ - ⇒ - ⇒ -› **where**
‹*get-occs-list-at-impl* = (λ*N C D*. *read-occs-wl-heur-code* (λ*M*. *mop-cocc-list-at-impl M C D*) *N*)›

global-interpretation *occs-at-at*: *read-occs-param-adder2* **where**
f = ‹λ*C D S*. *mop-cocc-list-at-impl S C D*› **and**
f′ = ‹λ*C D S*. *mop-cocc-list-at S C D*› **and**
x-assn = *sint64-nat-assn* **and**
P = ‹λ*C D S*. *cocc-list-at-pre S C D*› **and**
R = ‹*unat-lit-rel*› **and**
R′ = ‹*snat-rel′ TYPE*(*64*)›
rewrites
‹(λ*N C D*. *read-occs-wl-heur* (λ*M*. *mop-cocc-list-at M C D*) *N*) = *get-occs-list-at*› **and**
‹(λ*N C D*. *read-occs-wl-heur-code* (λ*M*. *mop-cocc-list-at-impl M C D*) *N*) = *get-occs-list-at-impl*›
apply *unfold-locales*
apply (*rule mop-cocc-list-at-impl.refine*)
subgoal

by (*auto simp*: *read-all-st-def get-occs-list-at-def intro*!: *ext*
split: *isasat-int-splits*)
subgoal
by (*auto simp*: *get-occs-list-at-impl-def*)
done

lemma *get-occs-list-at-alt-def*: ‹*get-occs-list-at* = (λ*N C D*. *occs-at-at.XX.XX.mop N* (*C*, *D*))›
by (*auto simp*: *occs-at-at.XX.XX.mop-def get-occs-list-at-def mop-cocc-list-at-def read-all-param-adder-ops.mop-def*
read-all-st-def summarize-ASSERT-conv
mop-cocc-list-length-def split: *tuple17.splits intro*!: *ext*)

lemmas [*sepref-fr-rules*] = *occs-at-at.refine*
occs-at-at.mop-refine[*unfolded get-occs-list-at-alt-def*[*symmetric*]]

lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
length-occs-impl-def[*unfolded read-all-st-code-def*]
length-occs-at-impl-def[*unfolded read-all-st-code-def*]

lemma *mop-arena-promote-st-alt-def*:
‹*mop-arena-promote-st S C* = *do* {
let (*N'*, *S*) = *extract-arena-wl-heur S*;
let (*lcount*, *S*) = *extract-lcount-wl-heur S*;
ASSERT(*clss-size-lcount lcount* ≥ *1*);
let lcount = *clss-size-decr-lcount lcount*;
N' ← *mop-arena-set-status N' C IRRED*;
RETURN (*update-arena-wl-heur N'* (*update-lcount-wl-heur lcount S*))
}›
by (*auto simp*: *mop-arena-promote-st-def state-extractors split*: *isasat-int-splits*)

sepref-def *mop-arena-promote-st-impl*
is ‹*uncurry mop-arena-promote-st*›
:: ‹*isasat-bounded-assn*^{d} *$*_a$ *sint64-nat-assn*^{k} →$_a$ *isasat-bounded-assn*›
unfolding *mop-arena-promote-st-alt-def*
by *sepref*

sepref-def *get-lsize-limit-stats-impl*
is ‹*RETURN o get-lsize-limit-stats*›
:: ‹*isasat-stats-assn*^{k} →$_a$ *lbd-size-limit-assn*›
unfolding *stats-code-unfold*
by *sepref*

definition *get-lsize-limit-stats-st-impl* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*get-lsize-limit-stats-st-impl* = *read-stats-wl-heur-code get-lsize-limit-stats-impl*›

global-interpretation *lsize-limit*: *read-stats-param-adder0* **where**
f' = ‹*RETURN o get-lsize-limit-stats*› **and**
f = *get-lsize-limit-stats-impl* **and**
x-assn = ‹*uint32-nat-assn* ×$_a$ *sint64-nat-assn*› **and**
P = ‹λ-. *True*›
rewrites ‹*read-stats-wl-heur* (*RETURN o get-lsize-limit-stats*) = *RETURN o get-lsize-limit-stats-st*›
and
‹*read-stats-wl-heur-code get-lsize-limit-stats-impl* = *get-lsize-limit-stats-st-impl*›
apply *unfold-locales*
apply (*rule get-lsize-limit-stats-impl.refine*[*unfolded lbd-size-limit-assn-def*]; *assumption*)
subgoal by (*auto simp*: *read-all-st-def get-lsize-limit-stats-def get-lsize-limit-stats-def get-lsize-limit-stats-st-def*
intro!: *ext*

split: *isasat-int-splits*)
subgoal by (*auto simp*: *get-lsize-limit-stats-st-impl-def*)
done

lemmas [*sepref-fr-rules*] = *lsize-limit.refine*

lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
get-lsize-limit-stats-st-impl-def[*unfolded read-all-st-code-def*]

lemma *set-stats-size-limit-st-alt-def*:
‹*RETURN ooo set-stats-size-limit-st* = (λ*lbd sze T*.
let (*stats*, *T*) = *extract-stats-wl-heur T*;
stats = *set-stats-size-limit lbd sze stats*
in RETURN (*update-stats-wl-heur stats T*)
)›
by (*auto simp*: *set-stats-size-limit-st-def state-extractors split*: *isasat-int-splits intro*!: *ext*)

sepref-register ‹*LSize-Stats* :: *nat* ⇒ *nat* ⇒ -›
IsaSAT-Stats-LLVM.update-f set-stats-size-limit-st stats-forward-rounds-st
incr-purelit-rounds-st

lemma *set-stats-size-limit-alt-def*:
‹*RETURN ooo set-stats-size-limit* = (λ*lbd size*′ *stats*. *RETURN* (*set-lsize-limit-stats* (*LSize-Stats lbd size*′) *stats*))›
unfolding *set-stats-size-limit-def LSize-Stats-def*
by (*auto intro*!: *ext*)

sepref-def *set-stats-size-limit-impl*
is ‹*uncurry2* (*RETURN ooo set-stats-size-limit*)›
:: ‹*uint32-nat-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} *$*_a$ *isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
unfolding *set-stats-size-limit-alt-def stats-code-unfold*
by *sepref*

sepref-def *set-stats-size-limit-st-impl*
is ‹*uncurry2* (*RETURN ooo set-stats-size-limit-st*)›
:: ‹*uint32-nat-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} *$*_a$ *isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
unfolding *set-stats-size-limit-st-alt-def*
by *sepref*

lemma *stats-forward-rounds-st-alt-def*:
‹*stats-forward-rounds-st S* = (*case S of IsaSAT M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs* ⇒ *stats-forward-rounds stats*)›
by (*cases S*) (*auto simp*: *stats-forward-rounds-st-def*)

sepref-def *stats-forward-rounds-st-impl*
is ‹*RETURN o stats-forward-rounds-st*›
:: ‹*isasat-bounded-assn*^{k} →$_a$ *word64-assn*›
unfolding *stats-forward-rounds-st-alt-def isasat-bounded-assn-def*
by *sepref*

lemma *incr-purelit-rounds-st-alt-def*:
‹*incr-purelit-rounds-st S* = (*let* (*stats*, *S*) = *extract-stats-wl-heur S*; *stats* = *incr-purelit-rounds stats in update-stats-wl-heur stats S*)›
by (*auto simp*: *incr-purelit-rounds-st-def state-extractors split*: *isasat-int-splits intro*!: *ext*)

sepref-def *incr-purelit-rounds-st-impl*
is ‹*RETURN o incr-purelit-rounds-st*›
:: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
unfolding *incr-purelit-rounds-st-alt-def*
by *sepref*

end
theory *IsaSAT-Setup-LLVM*
imports
IsaSAT-Setup1-LLVM
IsaSAT-Setup2-LLVM
IsaSAT-Setup3-LLVM
IsaSAT-Setup4-LLVM
IsaSAT-Profile-LLVM
begin

Lift Operations to State

sepref-def *mark-added-clause-heur2-impl*
is ‹*uncurry mark-added-clause-heur2*›
:: ‹*isasat-bounded-assn*^{d} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *isasat-bounded-assn*›
unfolding *mark-added-clause-heur2-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *maybe-mark-added-clause-heur2-impl*
is ‹*uncurry maybe-mark-added-clause-heur2*›
:: ‹*isasat-bounded-assn*^{d} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *isasat-bounded-assn*›
unfolding *maybe-mark-added-clause-heur2-def*
by *sepref*

experiment begin
lemma *from-bool1*: *from-bool True* = *1*
by *auto*
lemmas [*llvm-pre-simp*] = *from-bool1*
lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
remove-d-code-def[*unfolded isasat-state.remove-d-code-def*]

export-llvm
ema-update-impl
VMTF-Node-impl
VMTF-stamp-impl
VMTF-get-prev-impl
VMTF-get-next-impl
get-conflict-wl-is-None-fast-code
count-decided-st-heur-fast-code
polarity-st-heur-pol-fast
count-decided-st-heur-fast-code
access-lit-in-clauses-heur-fast-code
rewatch-heur-fast-code
rewatch-heur-st-fast-code
set-zero-wasted-impl
opts-restart-st-fast-code
opts-unbounded-mode-st-fast-code

end

end
theory *IsaSAT-Rephase-State*
imports *IsaSAT-Rephase IsaSAT-Setup IsaSAT-Show*
begin

definition *rephase-heur-stats* :: ‹*64 word* ⇒ *64 word* ⇒ *64 word* ⇒ *restart-heuristics* ⇒ *restart-heuristics nres*› **where**
‹*rephase-heur-stats* = (λ*end-of-phase lrephase b* (*fast-ema*, *slow-ema*, *restart-info*, *wasted*, φ, *relu*).
do {
φ ← *phase-rephase end-of-phase lrephase b* φ;
RETURN (*fast-ema*, *slow-ema*, *restart-info*, *wasted*, φ, *relu*)
})›

definition *rephase-heur* :: ‹*64 word* ⇒ *64 word* ⇒ *64 word* ⇒ *isasat-restart-heuristics* ⇒ *isasat-restart-heuristics nres*› **where**
‹*rephase-heur* = (λ*end-of-phase lrephase b heur*.
do {
φ ← *rephase-heur-stats end-of-phase lrephase b* (*get-content heur*);
RETURN (*Restart-Heuristics* φ)
})›

lemma *rephase-heur-spec*:
‹*heuristic-rel* A *heur* ⟹ *rephase-heur end-of-phase lrephase b heur* ≤ ⇓*Id* (*SPEC*(*heuristic-rel* A))›
unfolding *rephase-heur-def rephase-heur-stats-def*
apply (*refine-vcg phase-rephase-spec*[*THEN order-trans*])
apply (*auto simp*: *heuristic-rel-def heuristic-rel-stats-def*)
done

definition *rephase-heur-st* :: ‹*isasat* ⇒ *isasat nres*› **where**
‹*rephase-heur-st* = (λ*S*. *do* {
let lcount = *get-global-conflict-count S*;
let heur = *get-heur S*;
let stats = *get-stats-heur S*;
let rephase-count = *stats-rephase stats*;
let stats = *incr-rephase-total stats*;
let b = *current-restart-phase heur*;
heur ← *rephase-heur lcount rephase-count b heur*;
let - = *isasat-print-progress* (*current-phase-letter* (*current-rephasing-phase heur*))
b stats (*get-learned-count S*);
RETURN (*set-stats-wl-heur stats* (*set-heur-wl-heur heur S*))
})›

lemma *rephase-heur-st-spec*:
‹(*S*, *S'*) ∈ *twl-st-heur* ⟹ *rephase-heur-st S* ≤ *SPEC*(λ*S*. (*S*, *S'*) ∈ *twl-st-heur*)›
unfolding *rephase-heur-st-def*
apply (*cases S'*)
apply (*refine-vcg rephase-heur-spec*[*THEN order-trans*, *of* ‹*all-atms-st S'*›])
apply (*simp-all add*: *twl-st-heur-def all-atms-st-def*)
done

definition *save-rephase-heur-stats* :: ‹*64 word* ⇒ *restart-heuristics* ⇒ *restart-heuristics nres*› **where**
‹*save-rephase-heur-stats* = (λ*n* (*fast-ema*, *slow-ema*, *restart-info*, *wasted*, φ, *relu*).
do {

```
    φ ← phase-save-phase n φ;
    RETURN (fast-ema, slow-ema, restart-info, wasted, φ, relu)
  })›

definition save-rephase-heur :: ‹64 word ⇒ isasat-restart-heuristics ⇒ isasat-restart-heuristics nres›
where
  ‹save-rephase-heur = (λn heur.
  do {
  φ ← save-rephase-heur-stats n (get-content heur);
  RETURN (Constructor φ)
  })›

lemma save-phase-heur-spec:
  ‹heuristic-rel A heur ⟹ save-rephase-heur n heur ≤ ⇓Id (SPEC(heuristic-rel A))›
  unfolding save-rephase-heur-def save-rephase-heur-stats-def
  apply (refine-vcg phase-save-phase-spec[THEN order-trans])
  apply (auto simp: heuristic-rel-def heuristic-rel-stats-def)
  done


definition save-phase-st :: ‹isasat ⇒ isasat nres› where
  ‹save-phase-st = (λS. do {
    let stats = get-stats-heur S;
    let n = no-conflict-until stats;
    let heur = get-heur S;
    heur ← save-rephase-heur n heur;
    RETURN (set-heur-wl-heur heur S)
   })›

lemma save-phase-st-spec:
  ‹(S, S') ∈ twl-st-heur ⟹ save-phase-st S ≤ SPEC(λS. (S, S') ∈ twl-st-heur)›
  unfolding save-phase-st-def
  apply (cases S')
  apply (refine-vcg save-phase-heur-spec[THEN order-trans, of ‹all-atms-st S'›])
  apply (simp-all add:  twl-st-heur-def isa-length-trail-pre all-atms-st-def flip: all-lits-st-alt-def)
  done

end
theory IsaSAT-Show-LLVM
  imports
    IsaSAT-Show
    IsaSAT-Setup0-LLVM
begin


sepref-register isasat-current-information print-c print-uint64

sepref-def print-c-impl
  is ‹RETURN o print-c›
  :: ‹word-assn^k →_a unit-assn›
  unfolding print-c-def
  by sepref

sepref-def print-uint64-impl
  is ‹RETURN o print-uint64›
  :: ‹word-assn^k →_a unit-assn›
```

unfolding *print-uint64-def*
by *sepref*

sepref-def *print-open-colour-impl*
is ‹*RETURN o print-open-colour*›
:: ‹*word-assn*^{k} →$_a$ *unit-assn*›
unfolding *print-open-colour-def*
by *sepref*

sepref-def *print-close-colour-impl*
is ‹*RETURN o print-close-colour*›
:: ‹*word-assn*^{k} →$_a$ *unit-assn*›
unfolding *print-close-colour-def*
by *sepref*

sepref-def *print-char-impl*
is ‹*RETURN o print-char*›
:: ‹*word-assn*^{k} →$_a$ *unit-assn*›
unfolding *print-char-def*
by *sepref*

sepref-def *isasat-current-information-impl* [*llvm-code*]
is ‹*uncurry2* (*RETURN ooo isasat-current-information-stats*)›
:: ‹*word-assn*^{k} *$_a$ *isasat-stats-assn*^{d} *$_a$ *lcount-assn*^{k} →$_a$ *isasat-stats-assn*›
unfolding *isasat-current-information-stats-def*
isasat-current-information-def lcount-assn-def
by *sepref*

lemma *isasat-current-status-alt-def*:
‹*isasat-current-status* =
(λ*S*.
let
(*heur*, *S*) = *extract-heur-wl-heur S*;
(*stats*, *S*) = *extract-stats-wl-heur S*;
(*lcount*, *S*) = *extract-lcount-wl-heur S*;
curr-phase = *current-restart-phase* (*heur*);
stats = (*isasat-current-information curr-phase stats lcount*)
in RETURN (*update-stats-wl-heur stats* (*update-heur-wl-heur heur* (*update-lcount-wl-heur lcount S*))))›
by (*auto simp*: *isasat-current-status-def state-extractors split*: *isasat-int-splits intro*!: *ext*)

sepref-def *isasat-current-status-fast-code*
is ‹*isasat-current-status*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *isasat-current-status-alt-def isasat-current-information-def*
by *sepref*

lemma *isasat-current-progress-alt-def*:
‹*isasat-current-progress* =
(λ*c S*. *case S of Tuple17 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs* ⇒
let
curr-phase = *current-restart-phase heur*;

```
      - = isasat-print-progress c curr-phase stats clss
    in RETURN ())›
  unfolding isasat-current-progress-def
 apply (intro ext, rename-tac c S, case-tac S)
 apply (auto simp: Let-def intro!: ext)
 done


sepref-def isasat-print-progress-impl
 is ‹uncurry3 (RETURN oooo isasat-print-progress)›
 :: ‹word-assn^k *_a word-assn^k *_a isasat-stats-assn^k *_a lcount-assn^k →_a unit-assn›
 unfolding isasat-current-progress-alt-def isasat-print-progress-def lcount-assn-def
by sepref

sepref-register isasat-current-progress
sepref-def isasat-current-progress-impl
 is ‹uncurry isasat-current-progress›
 :: ‹word-assn^k *_a isasat-assn^k →_a unit-assn›
 supply [[goals-limit=1]]
 unfolding isasat-current-progress-alt-def isasat-bounded-assn-def
 by sepref
end
theory IsaSAT-Rephase-State-LLVM
imports
 IsaSAT-Rephase-State IsaSAT-Rephase-LLVM IsaSAT-Show-LLVM IsaSAT-Setup-LLVM
begin
hide-const (open) NEMonad.ASSERT NEMonad.RETURN

sepref-def save-phase-heur-stats-impl
 is ‹uncurry save-rephase-heur-stats›
 :: ‹word64-assn^k *_a heuristic-int-assn^d →_a heuristic-int-assn›
 supply [[goals-limit=1]]
 unfolding save-rephase-heur-stats-def heuristic-int-assn-def
 by sepref

sepref-register save-rephase-heur-stats
sepref-def save-phase-heur-impl
 is ‹uncurry save-rephase-heur›
 :: ‹word64-assn^k *_a heuristic-assn^d →_a heuristic-assn›
 supply [[goals-limit=1]]
 unfolding save-rephase-heur-def
 by sepref

lemma save-phase-st-alt-def:
 ‹save-phase-st = (λS. do {
     let (heur, S) = extract-heur-wl-heur S;
     let (stats, S) = extract-stats-wl-heur S;
     let n = no-conflict-until stats;
     heur ← save-rephase-heur n heur;
     RETURN (update-heur-wl-heur heur (update-stats-wl-heur stats S))
  })›
 by (auto simp: save-phase-st-def state-extractors split: isasat-int-splits intro!: ext)

sepref-def save-phase-heur-st
 is save-phase-st
 :: ‹isasat-bounded-assn^d →_a isasat-bounded-assn›
```

supply [[*goals-limit=1*]]
unfolding *save-phase-st-alt-def*
by *sepref*

sepref-def *phase-save-rephase-impl*
is ‹*uncurry3 phase-rephase*›
:: ‹*word-assn*^{k} $*_a$ *word-assn*^{k} $*_a$ *word-assn*^{k} $*_a$ *phase-heur-assn*^{d} $\rightarrow_a$ *phase-heur-assn*›
unfolding *phase-rephase-def copy-phase2-def*[*symmetric*] *phase-heur-assn-def*
supply *of-nat-snat*[*sepref-import-param*]
apply (*subst copy-phase2-def*)
by *sepref*

sepref-def *rephase-heur-stats-impl*
is ‹*uncurry3 rephase-heur-stats*›
:: ‹*word-assn*^{k} $*_a$ *word-assn*^{k} $*_a$ *word-assn*^{k} $*_a$ *heuristic-int-assn*^{d} $\rightarrow_a$ *heuristic-int-assn*›
unfolding *rephase-heur-stats-def heuristic-int-assn-def*
by *sepref*

sepref-register *rephase-heur-stats isasat-print-progress*

sepref-def *rephase-heur-impl*
is ‹*uncurry3 rephase-heur*›
:: ‹*word-assn*^{k} $*_a$ *word-assn*^{k} $*_a$ *word-assn*^{k} $*_a$ *heuristic-assn*^{d} $\rightarrow_a$ *heuristic-assn*›
unfolding *rephase-heur-def*
by *sepref*

lemma *rephase-heur-st-alt-def*:
‹*rephase-heur-st* = (λ*S*. *do* {
let lc = *get-global-conflict-count S*;
let (*heur*, *S*) = *extract-heur-wl-heur S*;
let (*stats*, *S*) = *extract-stats-wl-heur S*;
let lrephase = *stats-rephase stats*;
let stats = *incr-rephase-total stats*;
let (*lcount*, *S*) = *extract-lcount-wl-heur S*;
let b = *current-restart-phase heur*;
heur ← *rephase-heur lc lrephase b heur*;
let - = *isasat-print-progress* (*current-phase-letter* (*current-rephasing-phase heur*))
b stats (*lcount*);
RETURN (*update-heur-wl-heur heur* (*update-stats-wl-heur stats* (*update-lcount-wl-heur lcount S*)))
})›
by (*auto simp*: *rephase-heur-st-def state-extractors split*: *isasat-int-splits intro*!: *ext*)

sepref-register *rephase-heur*
sepref-def *rephase-heur-st-impl*
is *rephase-heur-st*
:: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *rephase-heur-st-alt-def*
by *sepref*

experiment
begin

export-llvm *rephase-heur-st-impl*

save-phase-heur-st

end

end
theory *IsaSAT-LBD*
imports *IsaSAT-Setup*
begin

definition *mark-lbd-from-clause-heur* :: ‹*trail-pol* ⇒ *arena* ⇒ *nat* ⇒ *lbd* ⇒ *lbd nres*› **where**
‹*mark-lbd-from-clause-heur M N C lbd* = *do* {
n ← *mop-arena-length N C*;
nfoldli [*1*..<*n*] (λ-. *True*)
(λ*i lbd*. *do* {
L ← *mop-arena-lit2 N C i*;
ASSERT(*get-level-pol-pre* (*M*, *L*));
let lev = *get-level-pol M L*;
ASSERT(*lev* ≤ *Suc* (*unat32-max div 2*));
RETURN (*if lev* = *0 then lbd else lbd-write lbd lev*)})
lbd}›

lemma *count-decided-le-length*: ‹*count-decided M* ≤ *length M*›
unfolding *count-decided-def* **by** (*rule length-filter-le*)

lemma *mark-lbd-from-clause-heur-correctness*:
assumes ‹(*M*, *M'*) ∈ *trail-pol* *A*› **and** ‹*valid-arena N N' vdom*› ‹*C* ∈# *dom-m N'*› **and**
‹*literals-are-in-*$\mathcal{L}_{in}$ *A* (*mset* (*N'* ∝ *C*))›
shows ‹*mark-lbd-from-clause-heur M N C lbd* ≤ ⇓ *Id* (*SPEC*(λ-::*bool list*. *True*))›
using *assms*
unfolding *mark-lbd-from-clause-heur-def*
apply (*refine-vcg mop-arena-length*[*THEN fref-to-Down-curry*, *THEN order-trans*, *of N' C - - vdom*]
nfoldli-rule[**where** *I* = ‹ λ- - -. *True*›])
subgoal by *auto*
subgoal by *auto*
unfolding *Down-id-eq comp-def*
apply (*refine-vcg mop-arena-length*[*THEN fref-to-Down-curry*, *THEN order-trans*, *of N' C - - vdom*]
nfoldli-rule[**where** *I* = ‹ λ- - -. *True*›] *mop-arena-lit*[*THEN order-trans*])
subgoal by *auto*
apply *assumption*+
subgoal by *simp*
apply *auto*[]
subgoal *H*
by (*metis append-cons-eq-upt*(*2*) *eq-upt-Cons-conv*)
subgoal for *x l1 l2* σ *xa* **using** *H*[*of l1 x l2*] **apply** −
by (*auto intro*!: *get-level-pol-pre literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$)
subgoal for *x l1 l2* σ *xa* **using** *H*[*of l1 x l2*] **apply** −
using *count-decided-ge-get-level*[*of M' xa*] *count-decided-le-length*[*of M'*]
by (*auto simp*: *trail-pol-alt-def literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$ *simp flip*: *get-level-get-level-pol*)
done

definition *calculate-LBD-st* :: ‹(*nat*, *nat*) *ann-lits* ⇒ *nat clauses-l* ⇒ *nat* ⇒ *nat clauses-l nres*› **where**
‹*calculate-LBD-st* = (λ*M N C*. *RETURN N*)›

abbreviation *TIER-TWO-MAXIMUM* **where**
‹*TIER-TWO-MAXIMUM* ≡ *6*›

abbreviation *TIER-ONE-MAXIMUM* **where**
‹*TIER-ONE-MAXIMUM* ≡ *2*›

definition *calculate-LBD-heur-st* :: ‹- ⇒ *arena* ⇒ *lbd* ⇒ *nat* ⇒ (*arena* × *lbd*) *nres*› **where**
‹*calculate-LBD-heur-st* = (λ*M N lbd C*. *do*{
old-glue ← *mop-arena-lbd N C*;
st ← *mop-arena-status N C*;
if st = *IRRED then RETURN* (*N*, *lbd*)
else if old-glue < *TIER-TWO-MAXIMUM then do* {
N ← *mop-arena-mark-used2 N C*;
RETURN (*N*, *lbd*)
}
else do {
lbd ← *mark-lbd-from-clause-heur M N C lbd*;
glue ← *get-LBD lbd*;
lbd ← *lbd-empty lbd*;
N ← (*if glue* < *old-glue then mop-arena-update-lbd C glue N else RETURN N*);
N ← (*if glue* < *TIER-TWO-MAXIMUM* ∨ *old-glue* < *TIER-TWO-MAXIMUM then mop-arena-mark-used2 N C else mop-arena-mark-used N C*);
RETURN (*N*, *lbd*)
}})›

lemma *calculate-LBD-st-alt-def*:
‹*calculate-LBD-st* = (λ*M N C*. *do* {
old-glue :: *nat* ← *SPEC*(λ- . *True*);
st :: *clause-status* ← *SPEC*(λ- . *True*);
if st = *IRRED then RETURN N*
else if old-glue < *6 then do* {
- ← *RETURN N*;
RETURN N
}
else do {
lbd::*bool list* ← *SPEC*(λ-. *True*);
glue::*nat* ← *get-LBD lbd*;
-::*bool list* ← *lbd-empty lbd*;
- ← *RETURN N*;
- ← *RETURN N*;
RETURN N
})› (**is** ‹*?A* = *?B*›)
unfolding *calculate-LBD-st-def get-LBD-def lbd-empty-def*
by (*auto intro*!: *ext rhs-step-bind-RES split*: *if-splits cong*: *if-cong*)

lemma *RF-COME-ON*: ‹(*x*, *y*) ∈ *Id* ⟹ *f x* ≤ ⇓ *Id* (*f y*)›
by *auto*

lemma *mop-arena-update-lbd*:
‹*C* ∈# *dom-m N* ⟹ *valid-arena arena N vdom* ⟹
mop-arena-update-lbd C glue arena ≤ *SPEC*(λ*c*. (*c*, *N*) ∈ {(*c*, *N'*). *N'*=*N* ∧ *valid-arena c N vdom* ∧
length c = *length arena*})›
unfolding *mop-arena-update-lbd-def*
by (*auto simp*: *update-lbd-pre-def arena-is-valid-clause-idx-def*
intro!: *ASSERT-leI valid-arena-update-lbd*)

lemma *mop-arena-mark-used-valid*:
  ‹*C* ∈# *dom-m N* ⟹ *valid-arena arena N vdom* ⟹
    mop-arena-mark-used arena C ≤ *SPEC*(λ*c*. (*c*, *N*) ∈ {(*c*, *N'*). *N'*=*N* ∧ *valid-arena c N vdom* ∧
      length c = *length arena*})›
  unfolding *mop-arena-mark-used-def*
  by (*auto simp*: *arena-act-pre-def arena-is-valid-clause-idx-def*
    intro!: *ASSERT-leI valid-arena-mark-used*)

lemma *mop-arena-mark-used2-valid*:
  ‹*C* ∈# *dom-m N* ⟹ *valid-arena arena N vdom* ⟹
    mop-arena-mark-used2 arena C ≤ *SPEC*(λ*c*. (*c*, *N*) ∈ {(*c*, *N'*). *N'*=*N* ∧ *valid-arena c N vdom* ∧
      length c = *length arena*})›
  unfolding *mop-arena-mark-used2-def*
  by (*auto simp*: *arena-act-pre-def arena-is-valid-clause-idx-def*
    intro!: *ASSERT-leI valid-arena-mark-used2*)

abbreviation *twl-st-heur-conflict-ana'*
  :: ‹*nat* ⇒ *clss-size* ⇒ (*isasat* × *nat twl-st-wl*) *set*›
where
  ‹*twl-st-heur-conflict-ana' r lcount* ≡ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-conflict-ana* ∧
    length (*get-clauses-wl-heur S*) = *r* ∧ *get-learned-count S* = *lcount*}›

lemma *calculate-LBD-heur-st-calculate-LBD-st*:
  assumes ‹*valid-arena arena N vdom*›
    ‹(*M*, *M'*) ∈ *trail-pol* 𝒜›
    ‹*C* ∈# *dom-m N*›
    ‹*literals-are-in-*$\mathcal{L}_{in}$ 𝒜 (*mset* (*N* ∝ *C*))› ‹(*C*, *C'*) ∈ *nat-rel*›
  shows ‹*calculate-LBD-heur-st M arena lbd C* ≤
    ⇓{((*arena'*, *lbd*), *N'*). *valid-arena arena' N' vdom* ∧ *N* = *N'* ∧ *length arena* = *length arena'*}
      (*calculate-LBD-st M' N C'*)›
proof −
  have *WTF*: ‹(*a*, *b*) ∈ *R* ⟹ *b*=*b'* ⟹ (*a*, *b'*) ∈ *R*› **for** *a a' b b' R*
    by *auto*
  show *?thesis*
    using *assms*
    unfolding *calculate-LBD-heur-st-def calculate-LBD-st-alt-def*
    apply (*refine-vcg mark-lbd-from-clause-heur-correctness*[*of* - *M'*
      𝒜 - *N vdom*]
      mop-arena-update-lbd[*of* - - - *vdom*]
      mop-arena-mark-used-valid[*of* - *N* - *vdom*]
      mop-arena-mark-used2-valid[*of* - *N* - *vdom*])
    subgoal
      unfolding *twl-st-heur-conflict-ana-def*
      by (*auto simp*: *mop-arena-lbd-def get-clause-LBD-pre-def arena-is-valid-clause-idx-def*
        intro!: *ASSERT-leI exI*[*of* - *N*] *exI*[*of* - *vdom*])
    subgoal
      unfolding *twl-st-heur-conflict-ana-def*
      by (*auto simp*: *mop-arena-status-def arena-is-valid-clause-vdom-def arena-is-valid-clause-idx-def*
        intro!: *ASSERT-leI exI*[*of* - *N*] *exI*[*of* - *vdom*])
    subgoal
      by (*auto simp*: *twl-st-heur-conflict-ana-def RETURN-RES-refine-iff*)
    subgoal
      by (*auto simp*: *twl-st-heur-conflict-ana-def RETURN-RES-refine-iff*)
    subgoal
      by (*auto simp*: *twl-st-heur-conflict-ana-def RETURN-RES-refine-iff*)

```
    subgoal
      by (force simp: twl-st-heur-conflict-ana-def)
    apply (rule RF-COME-ON)
    subgoal
      by auto
    apply (rule RF-COME-ON)
    subgoal
      by auto
    subgoal
      unfolding twl-st-heur-conflict-ana-def
      by (auto simp: mop-arena-lbd-def get-clause-LBD-pre-def arena-is-valid-clause-idx-def
        intro!: ASSERT-leI exI[of - ‹get-clauses-wl (fst y)›] exI[of - ‹set (get-vdom (fst x))›])
    subgoal
      by (force simp: twl-st-heur-conflict-ana-def)
    subgoal
      by (force simp: twl-st-heur-conflict-ana-def)
    subgoal
      by (force simp: twl-st-heur-conflict-ana-def)
   done
qed


definition mark-lbd-from-list :: ‹-› where
  ‹mark-lbd-from-list M C lbd = do {
    nfoldli (drop 1 C) (λ-. True)
      (λL lbd. RETURN (lbd-write lbd (get-level M L))) lbd
  }›

definition mark-lbd-from-list-heur :: ‹trail-pol ⇒ nat clause-l ⇒ lbd ⇒ lbd nres› where
  ‹mark-lbd-from-list-heur M C lbd = do {
  let n = length C;
  nfoldli [1..<n] (λ-. True)
    (λi lbd. do {
       ASSERT(i < length C);
       let L = C ! i;
       ASSERT(get-level-pol-pre (M, L));
       let lev = get-level-pol M L;
       ASSERT(lev ≤ Suc (unat32-max div 2));
       RETURN (if lev = 0 then lbd else lbd-write lbd lev)})
    lbd}›

definition mark-lbd-from-conflict :: ‹isasat ⇒ isasat nres› where
  ‹mark-lbd-from-conflict = (λS. do{
     lbd ← mark-lbd-from-list-heur (get-trail-wl-heur S) (get-outlearned-heur S) (get-lbd S);
     RETURN (set-lbd-wl-heur lbd S)
    })›


lemma mark-lbd-from-list-heur-correctness:
  assumes ‹(M, M') ∈ trail-pol A› and ‹literals-are-in-ℒ_in A (mset (tl C))›
  shows ‹mark-lbd-from-list-heur M C lbd ≤ ⇓ Id (SPEC(λ-::bool list. True))›
  using assms
  unfolding mark-lbd-from-list-heur-def
  apply (refine-vcg nfoldli-rule[where I = ‹λ- - -. True›])
  subgoal by auto
  subgoal
```

by (*auto simp*: *upt-eq-lel-conv nth-tl*)
subgoal for *x l1 l2 σ*
using *literals-are-in-$\mathcal{L}_{in}$-in-$\mathcal{L}_{all}$*[*of* $\mathcal{A}$ ‹*tl C*› ‹*x − 1*›]
by (*auto intro*!: *get-level-pol-pre simp*: *upt-eq-lel-conv nth-tl*)
subgoal for *x l1 l2 σ*
using *count-decided-ge-get-level*[*of M′* ‹*C ! x*›] *count-decided-le-length*[*of M′*]
using *literals-are-in-$\mathcal{L}_{in}$-in-$\mathcal{L}_{all}$*[*of* $\mathcal{A}$ ‹*tl C*› ‹*x − 1*›]
by (*auto simp*: *upt-eq-lel-conv nth-tl simp flip*: *get-level-get-level-pol*)
(*auto simp*: *trail-pol-alt-def*)
done

definition *mark-LBD-st* :: ‹*'v twl-st-wl* ⇒ (*'v twl-st-wl*) *nres*› **where**
‹*mark-LBD-st* = (λ*S*. *SPEC* (λ(*T*). *S* = *T*))›

lemma *mark-LBD-st-alt-def*:
‹*mark-LBD-st S* = *do* {*n* :: *bool list* ← *SPEC* (λ-. *True*); *SPEC* (λ(*T*). *S* = *T*)}›
unfolding *mark-LBD-st-def*
by *auto*

lemma *mark-lbd-from-conflict-mark-LBD-st*:
‹(*mark-lbd-from-conflict*, *mark-LBD-st*) ∈
[λ*S*. *get-conflict-wl S* ≠ *None* ∧ *literals-are-in-$\mathcal{L}_{in}$* (*all-atms-st S*) (*the* (*get-conflict-wl S*))]$_f$
twl-st-heur-conflict-ana → ⟨*twl-st-heur-conflict-ana*⟩*nres-rel*›
unfolding *mark-lbd-from-conflict-def mark-LBD-st-alt-def*
apply (*intro frefI nres-relI*)
subgoal for *x y*
apply (*refine-rcg mark-lbd-from-list-heur-correctness*[*of* - ‹*get-trail-wl y*› ‹*all-atms-st y*›,
THEN order-trans])
subgoal
by (*force simp*: *twl-st-heur-conflict-ana-def*)
subgoal
by (*rule literals-are-in-$\mathcal{L}_{in}$-mono*[*of* - ‹(*the* (*get-conflict-wl y*))›])
(*auto simp*: *twl-st-heur-conflict-ana-def out-learned-def*)
subgoal by *auto*
subgoal by (*auto simp*: *twl-st-heur-conflict-ana-def RETURN-RES-refine-iff*)
done
done

definition *update-lbd-and-mark-used* **where**
‹*update-lbd-and-mark-used i glue N* =
(*let N* = *update-lbd i glue N in*
(*if glue* ≤ *TIER-TWO-MAXIMUM then mark-used2 N i else mark-used N i*))›

lemma *length-update-lbd-and-mark-used*[*simp*]: ‹*length* (*update-lbd-and-mark-used i glue N*) = *length N*›
by (*auto simp*: *update-lbd-and-mark-used-def Let-def split*: *if-splits*)

CaDiCaL sets the used flags of clauses only as 1, not as two.

definition *update-lbd-shrunk-clause* **where**
‹*update-lbd-shrunk-clause C N* = *do* {
old-glue ← *mop-arena-lbd N C*;
st ← *mop-arena-status N C*;
le ← *mop-arena-length N C*;
ASSERT (*le* ≥ *2*);
if st = *IRRED then RETURN N*
else do {

```
      let new-glue = (if le − 1 ≥ old-glue then old-glue else le − 1);
      ASSERT (update-lbd-pre ((C, new-glue), N));
      RETURN (update-lbd-and-mark-used C new-glue N)
    }
}›
```

```
lemma update-lbd-shrunk-clause-valid:
  ‹C ∈# dom-m N ⟹ valid-arena arena N vdom ⟹
    update-lbd-shrunk-clause C arena ≤ SPEC(λc. (c, N) ∈ {(c, N'). N'=N ∧ valid-arena c N vdom ∧
      length c = length arena})›
  unfolding update-lbd-shrunk-clause-def mop-arena-lbd-def nres-monad3 mop-arena-status-def
    mop-arena-length-def update-lbd-and-mark-used-def
  apply refine-vcg
  subgoal
    unfolding get-clause-LBD-pre-def arena-is-valid-clause-idx-def
    by auto
  subgoal
    unfolding arena-is-valid-clause-vdom-def
    by auto
  subgoal
    unfolding arena-is-valid-clause-idx-def
    by auto
  subgoal
    by (rule arena-lifting(18))
  subgoal by auto
  subgoal by (auto simp: update-lbd-pre-def)
  subgoal by (auto simp: Let-def intro!: valid-arena-mark-used2 valid-arena-mark-used
    valid-arena-update-lbd)
  done

end
theory IsaSAT-Inner-Propagation-Defs
  imports IsaSAT-Setup IsaSAT-Bump-Heuristics
    IsaSAT-Clauses IsaSAT-VMTF IsaSAT-LBD
begin


definition find-non-false-literal-between where
  ‹find-non-false-literal-between M a b C =
    find-in-list-between (λL. polarity M L ≠ Some False) a b C›


definition isa-find-unwatched-between
 :: ‹- ⇒ trail-pol ⇒ arena ⇒ nat ⇒ nat ⇒ nat ⇒ (nat option) nres› where
‹isa-find-unwatched-between P M' NU a b C = do {
  ASSERT(C+a ≤ length NU);
  ASSERT(C+b ≤ length NU);
  (x, -) ← WHILE_T^λ(found, i). True
    (λ(found, i). found = None ∧ i < C + b)
    (λ(-, i). do {
      ASSERT(i < C + (arena-length NU C));
      ASSERT(i ≥ C);
      ASSERT(i < C + b);
      ASSERT(arena-lit-pre NU i);
      L ← mop-arena-lit NU i;
```

```
      ASSERT(polarity-pol-pre M' L);
      if P L then RETURN (Some (i − C), i) else RETURN (None, i+1)
    })
    (None, C+a);
  RETURN x
}
›
definition isa-find-unwatched
  :: ‹(nat literal ⇒ bool) ⇒ trail-pol ⇒ arena ⇒ nat ⇒ (nat option) nres›
where
‹isa-find-unwatched P M' arena C = do {
    l ← mop-arena-length arena C;
    b ← RETURN(l ≤ MAX-LENGTH-SHORT-CLAUSE);
    if b then isa-find-unwatched-between P M' arena 2 l C
    else do {
      ASSERT(get-saved-pos-pre arena C);
      pos ← mop-arena-pos arena C;
      n ← isa-find-unwatched-between P M' arena pos l C;
      if n = None then isa-find-unwatched-between P M' arena 2 pos C
      else RETURN n
    }
  }
›

definition isa-save-pos :: ‹nat ⇒ nat ⇒ isasat ⇒ isasat nres›
where
  ‹isa-save-pos C i = (λS. do {
      ASSERT(arena-is-valid-clause-idx (get-clauses-wl-heur S) C);
      if arena-length (get-clauses-wl-heur S) C > MAX-LENGTH-SHORT-CLAUSE then do {
        ASSERT(isa-update-pos-pre ((C, i), get-clauses-wl-heur S));
        let N = arena-update-pos C i  (get-clauses-wl-heur S);
        RETURN (set-clauses-wl-heur N S)
      } else RETURN S
    })
  ›


definition mark-conflict-to-rescore :: ‹nat ⇒ isasat ⇒ isasat nres› where
  ‹mark-conflict-to-rescore C S = do {
  let M = get-trail-wl-heur S;
  let N = get-clauses-wl-heur S;
  let D = get-conflict-wl-heur S;
  let vm = get-vmtf-heur S;
  n ← mop-arena-length N C;
  ASSERT (n ≤ length N);
  (-, vm) ← WHILE_T (λ(i, vm). i < n)
  (λ(i, vm). do{
      ASSERT (i < n);
     L ← mop-arena-lit2 N C i;
     vm ← isa-vmtf-bump-to-rescore-also-reasons-cl M N C (−L) vm;
    RETURN (i+1, vm)
   })
    (0, vm);
  let lbd = get-lbd S;
  (N, lbd) ← calculate-LBD-heur-st M N lbd C;
  let S = set-vmtf-wl-heur vm S;
```

```
  let S = set-clauses-wl-heur N S;
  let S = set-lbd-wl-heur lbd S;
  RETURN S
}›

definition set-conflict-wl-heur
  :: ‹nat ⇒ isasat ⇒ isasat nres›
where
  ‹set-conflict-wl-heur = (λC S. do {
    let n = 0;
    let M = get-trail-wl-heur S;
    let N = get-clauses-wl-heur S;
    let D = get-conflict-wl-heur S;
    let outl = get-outlearned-heur S;
    ASSERT(curry5 isa-set-lookup-conflict-aa-pre M N C D n outl);
    (D, clvls, outl) ← isa-set-lookup-conflict-aa M N C D n outl;
    j ← mop-isa-length-trail M;
    let S = IsaSAT-Setup.set-conflict-wl-heur D S;
    let S = set-outl-wl-heur outl S;
    let S = set-count-max-wl-heur clvls S;
    let S = set-literals-to-update-wl-heur j S;
    RETURN S})›

definition update-clause-wl-code-pre where
  ‹update-clause-wl-code-pre = (λ(((((((L, L'), C), b), j), w), i), f), S).
      w < length (get-watched-wl-heur S ! nat-of-lit L) )›

definition update-clause-wl-heur
   :: ‹nat literal ⇒ nat literal ⇒ nat ⇒ bool ⇒ nat ⇒ nat ⇒ nat ⇒ isasat ⇒
    (nat × nat × isasat) nres›
where
  ‹update-clause-wl-heur = (λ(L::nat literal) L' C b j w i f S. do {
     let N = get-clauses-wl-heur S;
     let W = get-watched-wl-heur S;
     K' ← mop-arena-lit2' (set (get-vdom S)) N C f;
     ASSERT(w < length N);
     N' ← mop-arena-swap C i f N;
     ASSERT(nat-of-lit K' < length W);
     ASSERT(length (W ! (nat-of-lit K')) < length N);
     let W = W[nat-of-lit K' := W ! (nat-of-lit K') @ [(C, L, b)]];
     let S = set-watched-wl-heur W S;
     let S = set-clauses-wl-heur N' S;
     RETURN (j, w+1, S)
  })›

definition propagate-lit-wl-heur
  :: ‹nat literal ⇒ nat ⇒ nat ⇒ isasat ⇒ isasat nres›
where
  ‹propagate-lit-wl-heur = (λL' C i S. do {
      let M = get-trail-wl-heur S;
      let N = get-clauses-wl-heur S;
      let heur = get-heur S;
      ASSERT(i ≤ 1);
      M ← cons-trail-Propagated-tr L' C M;
```

```
    N' ← mop-arena-swap C 0 (1 − i) N;
    heur ← mop-save-phase-heur (atm-of L') (is-pos L') heur;
    let S = set-trail-wl-heur M S;
    let S = set-clauses-wl-heur N' S;
    let S = set-heur-wl-heur heur S;
    RETURN S
  })›
```

definition *propagate-lit-wl-bin-heur*
:: ‹*nat literal* ⇒ *nat* ⇒ *isasat* ⇒ *isasat nres*›
where

```
  ‹propagate-lit-wl-bin-heur = (λL' C S. do {
    let M = get-trail-wl-heur S;
    let heur = get-heur S;
    M ← cons-trail-Propagated-tr L' C M;
    heur ← mop-save-phase-heur (atm-of L') (is-pos L') heur;
    let S = set-trail-wl-heur M S;
    let S = set-heur-wl-heur heur S;
    RETURN S
  })›
```

definition *unit-prop-body-wl-heur-inv* **where**

```
  ‹unit-prop-body-wl-heur-inv S j w L ⟷
    (∃ S'. (S, S') ∈ twl-st-heur ∧ unit-prop-body-wl-inv S' j w L)›
```

definition *unit-prop-body-wl-D-find-unwatched-heur-inv* **where**

```
  ‹unit-prop-body-wl-D-find-unwatched-heur-inv f C S ⟷
    (∃ S'. (S, S') ∈ twl-st-heur ∧ unit-prop-body-wl-find-unwatched-inv f C S')›
```

definition *keep-watch-heur* **where**

```
  ‹keep-watch-heur = (λL i j S. do {
    let W = get-watched-wl-heur S;
    ASSERT(nat-of-lit L < length W);
    ASSERT(i < length (W ! nat-of-lit L));
    ASSERT(j < length (W ! nat-of-lit L));
    let W =  W[nat-of-lit L := (W!(nat-of-lit L))[i := W ! (nat-of-lit L) ! j]];
    RETURN (set-watched-wl-heur W S)
   })›
```

definition *update-blit-wl-heur*
:: ‹*nat literal* ⇒ *nat* ⇒ *bool* ⇒ *nat* ⇒ *nat* ⇒ *nat literal* ⇒ *isasat* ⇒
(*nat* × *nat* × *isasat*) *nres*›
where

```
  ‹update-blit-wl-heur = (λ(L::nat literal) C b j w K S. do {
    let W = get-watched-wl-heur S;
    ASSERT(nat-of-lit L < length W);
    ASSERT(j < length (W ! nat-of-lit L));
    ASSERT(j < length (get-clauses-wl-heur S));
    ASSERT(w < length (get-clauses-wl-heur S));
    let W = W[nat-of-lit L := (W!nat-of-lit L)[j:= (C, K, b)]];
    RETURN (j+1, w+1, set-watched-wl-heur W S)
  })›
```

definition *pos-of-watched-heur* :: ‹*isasat⇒ nat ⇒ nat literal ⇒ nat nres*› **where**
‹*pos-of-watched-heur S C L = do* {
  L′ ← mop-access-lit-in-clauses-heur S C 0;
  RETURN (*if L = L′ then 0 else 1*)
}›

definition *unit-propagation-inner-loop-wl-loop-D-heur-inv0* **where**
  ‹*unit-propagation-inner-loop-wl-loop-D-heur-inv0 L* =
   (λ(*j*, *w*, *S′*). ∃ *S*. (*S′*, *S*) ∈ *twl-st-heur* ∧ *unit-propagation-inner-loop-wl-loop-inv L* (*j*, *w*, *S*) ∧
     length (*watched-by S L*) ≤ *length* (*get-clauses-wl-heur S′*) − *MIN-HEADER-SIZE*)›

definition *other-watched-wl-heur* :: ‹*isasat ⇒ nat literal ⇒ nat ⇒ nat ⇒ nat literal nres*› **where**
‹*other-watched-wl-heur S L C i = do* {
    ASSERT(*i < 2* ∧ *arena-lit-pre2* (*get-clauses-wl-heur S*) *C i* ∧
      arena-lit (*get-clauses-wl-heur S*) (*C + i*) = *L* ∧ *arena-lit-pre2* (*get-clauses-wl-heur S*) *C* (*1 − i*));
    mop-access-lit-in-clauses-heur S C (*1 − i*)
  }›

definition *isa-find-unwatched-wl-st-heur*
  :: ‹*isasat ⇒ nat ⇒ nat option nres*› **where**
‹*isa-find-unwatched-wl-st-heur* = (λ*S i. do* {
     isa-find-unwatched (λ*L. polarity-pol* (*get-trail-wl-heur S*) *L* ≠ *Some False*) (*get-trail-wl-heur S*)
(*get-clauses-wl-heur S*) *i*
  })›

definition *unit-propagation-inner-loop-body-wl-heur*
   :: ‹*nat literal ⇒ nat ⇒ nat ⇒ isasat ⇒* (*nat* × *nat* × *isasat*) *nres*›
   where
  ‹*unit-propagation-inner-loop-body-wl-heur L j w S0 = do* {
      ASSERT(*unit-propagation-inner-loop-wl-loop-D-heur-inv0 L* (*j*, *w*, *S0*));
      (*C*, *K*, *b*) ← *mop-watched-by-app-heur S0 L w*;
      S ← *keep-watch-heur L j w S0*;
      ASSERT(*length* (*get-clauses-wl-heur S*) = *length* (*get-clauses-wl-heur S0*));
      val-K ← *mop-polarity-st-heur S K*;
      if val-K = Some True
      then RETURN (*j+1*, *w+1*, *S*)
      else do {
        if b then do {
          if val-K = Some False
          then do {
            S ← *set-conflict-wl-heur C S*;
            RETURN (*j+1*, *w+1*, *S*)}
          else do {
            S ← *propagate-lit-wl-bin-heur K C S*;
            RETURN (*j+1*, *w+1*, *S*)}
        }
        else do {
   — Now the costly operations:
   ASSERT(*clause-not-marked-to-delete-heur-pre* (*S*, *C*));
   if ¬*clause-not-marked-to-delete-heur S C*
   then RETURN (*j*, *w+1*, *S*)
   else do {
     i ← *pos-of-watched-heur S C L*;
          ASSERT(*i* ≤ *1*);
     L′ ← *other-watched-wl-heur S L C i*;

```
    val-L' ← mop-polarity-st-heur S L';
    if val-L' = Some True
    then update-blit-wl-heur L C b j w L' S
    else do {
      f ← isa-find-unwatched-wl-st-heur S C;
      case f of
  None ⇒ do {
    if val-L' = Some False
    then do {
      S ← set-conflict-wl-heur C S;
      RETURN (j+1, w+1, S)}
    else do {
      S ← propagate-lit-wl-heur L' C i S;
      RETURN (j+1, w+1, S)}
  }
      | Some f ⇒ do {
    S ← isa-save-pos C f S;
    ASSERT(length (get-clauses-wl-heur S) = length (get-clauses-wl-heur S0));
    K ← mop-access-lit-in-clauses-heur S C f;
    val-L' ← mop-polarity-st-heur S K;
    if val-L' = Some True
    then update-blit-wl-heur L C b j w K S
    else do {
      update-clause-wl-heur L L' C b j w i f S
    }
       }
     }
         }
       }
     }
   }›
```

definition *unit-propagation-inner-loop-wl-loop-D-heur-inv* **where**
‹*unit-propagation-inner-loop-wl-loop-D-heur-inv* S_0 L =
($\lambda(j, w, S')$. $\exists S_0'\, S$. $(S_0, S_0') \in$ *twl-st-heur* $\wedge$ $(S', S) \in$ *twl-st-heur* $\wedge$ *unit-propagation-inner-loop-wl-loop-inv* L (j, w, S) $\wedge$
L $\in\#$ $\mathcal{L}_{all}$ (*all-atms-st S*) $\wedge$ *dom-m* (*get-clauses-wl S*) = *dom-m* (*get-clauses-wl* S_0') $\wedge$
length (*get-clauses-wl-heur* S_0) = *length* (*get-clauses-wl-heur* S'))›

definition *unit-propagation-inner-loop-wl-loop-D-heur*
:: ‹*nat literal* ⇒ *isasat* ⇒ (*nat* × *nat* × *isasat*) *nres*›
where
‹*unit-propagation-inner-loop-wl-loop-D-heur L* S_0 = *do* {
ASSERT(*length* (*watched-by-int* S_0 *L*) ≤ *length* (*get-clauses-wl-heur* S_0));
n ← *mop-length-watched-by-int* S_0 *L*;
WHILE$_T$*unit-propagation-inner-loop-wl-loop-D-heur-inv* S_0 *L*
($\lambda(j, w, S)$. $w < n \wedge$ *get-conflict-wl-is-None-heur S*)
($\lambda(j, w, S)$. *do* {
unit-propagation-inner-loop-body-wl-heur L j w S
})
(0, 0, S_0)
}›

definition *cut-watch-list-heur*
:: ‹*nat* ⇒ *nat* ⇒ *nat literal* ⇒ *isasat* ⇒ *isasat nres*›
where
‹*cut-watch-list-heur j w L* =(λ*S*. *do* {
let W = *get-watched-wl-heur S*;
ASSERT(*j* ≤ *length* (*W*!*nat-of-lit L*) ∧ *j* ≤ *w* ∧ *nat-of-lit L* < *length W* ∧
w ≤ *length* (*W* ! (*nat-of-lit L*)));
let W = *W*[*nat-of-lit L* := *take j* (*W*!(*nat-of-lit L*)) @ *drop w* (*W*!(*nat-of-lit L*))];
RETURN (*set-watched-wl-heur W S*)
})›
definition *cut-watch-list-heur2*
:: ‹*nat* ⇒ *nat* ⇒ *nat literal* ⇒ *isasat* ⇒ *isasat nres*›
where
‹*cut-watch-list-heur2* = (λ*j w L S*. *do* {
let W = *get-watched-wl-heur S*;
ASSERT(*j* ≤ *length* (*W* ! *nat-of-lit L*) ∧ *j* ≤ *w* ∧ *nat-of-lit L* < *length W* ∧
w ≤ *length* (*W* ! (*nat-of-lit L*)));
let n = *length* (*W*!(*nat-of-lit L*));
(*j*, *w*, *W*) ← *WHILE*$_T$λ(*j*, *w*, *W*). *j* ≤ *w* ∧ *w* ≤ *n* ∧ *nat-of-lit L* < *length W*
(λ(*j*, *w*, *W*). *w* < *n*)
(λ(*j*, *w*, *W*). *do* {
ASSERT(*w* < *length* (*W*!(*nat-of-lit L*)));
RETURN (*j*+*1*, *w*+*1*, *W*[*nat-of-lit L* := (*W*!(*nat-of-lit L*))[*j* := *W*!(*nat-of-lit L*)!*w*]])
})
(*j*, *w*, *W*);
ASSERT(*j* ≤ *length* (*W* ! *nat-of-lit L*) ∧ *nat-of-lit L* < *length W*);
let W = *W*[*nat-of-lit L* := *take j* (*W* ! *nat-of-lit L*)];
RETURN (*set-watched-wl-heur W S*)
})›
definition *unit-propagation-inner-loop-wl-D-heur*
:: ‹*nat literal* ⇒ *isasat* ⇒ *isasat nres*› **where**
‹*unit-propagation-inner-loop-wl-D-heur L S*$_0$ = *do* {
(*j*, *w*, *S*) ← *unit-propagation-inner-loop-wl-loop-D-heur L S*$_0$;
ASSERT(*length* (*watched-by-int S L*) ≤ *length* (*get-clauses-wl-heur S*$_0$) − *MIN-HEADER-SIZE*);
cut-watch-list-heur2 j w L S
}›

definition *select-and-remove-from-literals-to-update-wl-heur*
:: ‹*isasat* ⇒ (*isasat* × *nat literal*) *nres*›
where
‹*select-and-remove-from-literals-to-update-wl-heur S* = *do* {
ASSERT(*literals-to-update-wl-heur S* < *length* (*fst* (*get-trail-wl-heur S*)));
ASSERT(*literals-to-update-wl-heur S* + *1* ≤ *unat32-max*);
L ← *isa-trail-nth* (*get-trail-wl-heur S*) (*literals-to-update-wl-heur S*);
RETURN (*set-literals-to-update-wl-heur* (*literals-to-update-wl-heur S* + *1*) *S*, −*L*)
}›

definition *unit-propagation-outer-loop-wl-D-heur-inv*
:: ‹*isasat* ⇒ *isasat* ⇒ *bool*›
where
‹*unit-propagation-outer-loop-wl-D-heur-inv S*$_0$ *S*′ ⟷
(∃ *S*$_0$′ *S*. (*S*$_0$, *S*$_0$′) ∈ *twl-st-heur* ∧ (*S*′, *S*) ∈ *twl-st-heur* ∧
unit-propagation-outer-loop-wl-inv S ∧
dom-m (*get-clauses-wl S*) = *dom-m* (*get-clauses-wl S*$_0$′) ∧

length (get-clauses-wl-heur S′) = length (get-clauses-wl-heur S₀) ∧
isa-length-trail-pre (get-trail-wl-heur S′))⟩

definition *unit-propagation-update-statistics* :: ⟨*64 word ⇒ 64 word ⇒ isasat ⇒ isasat nres*⟩ **where**
⟨*unit-propagation-update-statistics p q S = do {*
let stats = get-stats-heur S;
let pq = q − p;
let stats = incr-propagation-by pq stats;
let stats = (if get-conflict-wl-is-None-heur S then stats else incr-conflict stats);
let stats = (if count-decided-pol (get-trail-wl-heur S) = 0 then incr-units-since-last-GC-by pq (incr-uset-by pq stats) else stats);
height ← (if get-conflict-wl-is-None-heur S then RETURN q else do {j ← trail-height-before-conflict (get-trail-wl-heur S); RETURN (of-nat j)});
let stats = set-no-conflict-until q stats;
RETURN (set-stats-wl-heur stats S)}⟩

definition *unit-propagation-outer-loop-wl-D-heur*
:: ⟨*isasat ⇒ isasat nres*⟩ **where**
⟨*unit-propagation-outer-loop-wl-D-heur S₀ = do {*
let j1 = isa-length-trail (get-trail-wl-heur S₀);
- ← RETURN (IsaSAT-Profile.start-propagate);
S ← WHILE_T^unit-propagation-outer-loop-wl-D-heur-inv S₀
(λS. literals-to-update-wl-heur S < isa-length-trail (get-trail-wl-heur S))
(λS. do {
ASSERT(literals-to-update-wl-heur S < isa-length-trail (get-trail-wl-heur S));
(S′, L) ← select-and-remove-from-literals-to-update-wl-heur S;
ASSERT(length (get-clauses-wl-heur S′) = length (get-clauses-wl-heur S));
unit-propagation-inner-loop-wl-D-heur L S′
})
S₀;
let j2 = isa-length-trail (get-trail-wl-heur S);
S ← unit-propagation-update-statistics (of-nat j1) (of-nat j2) S;
- ← RETURN (IsaSAT-Profile.stop-propagate);
RETURN S}
⟩

definition *isa-find-unset-lit* :: ⟨*trail-pol ⇒ arena ⇒ nat ⇒ nat ⇒ nat ⇒ nat option nres*⟩ **where**
⟨*isa-find-unset-lit M = isa-find-unwatched-between (λL. polarity-pol M L ≠ Some False) M*⟩

end
theory *IsaSAT-Inner-Propagation*
imports *IsaSAT-Setup*
IsaSAT-Clauses IsaSAT-VMTF IsaSAT-LBD
IsaSAT-Inner-Propagation-Defs
begin

Chapter 14

Propagation: Inner Loop

declare *all-atms-def*[*symmetric*,*simp*]
lemma *map-fun-rel-def2*:
  ‹⟨*R*⟩*map-fun-rel* (*D*$_0$ (*all-atms-st u*)) =
    {(*m*, *f*). ∀ (*i*, *j*)∈(λ*L*. (*nat-of-lit L*, *L*)) ‘ *set-mset* (*all-lits-st u*). *i* < *length m* ∧ (*m* ! *i*, *f j*) ∈ *R*}›
  unfolding *map-fun-rel-def*[*of* ‹*D*$_0$ (*all-atms-st u*)› *R*] **unfolding** *all-lits-st-alt-def*[*symmetric*] **..**

14.1 Find replacement

lemma *literals-are-in-*$\mathcal{L}_{in}$*-nth2*:
  fixes *C* :: *nat*
  assumes *dom*: ‹*C* ∈# *dom-m* (*get-clauses-wl S*)›
  shows ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st S*) (*mset* (*get-clauses-wl S* ∝ *C*))›
proof −
  let *?N* = ‹*get-clauses-wl S*›
  have ‹*?N* ∝ *C* ∈# *ran-mf ?N*›
    using *dom* **by** (*auto simp*: *ran-m-def*)
  then have ‹*mset* (*?N* ∝ *C*) ∈# *mset* ‘# (*ran-mf ?N*)›
    by *blast*
  from *all-lits-of-m-subset-all-lits-of-mmD*[*OF this*] **show** *?thesis*
    unfolding *is-*$\mathcal{L}_{all}$*-def literals-are-in-*$\mathcal{L}_{in}$*-def literals-are-*$\mathcal{L}_{in}$*-def*
      all-lits-st-alt-def[*symmetric*]
    by (*auto simp add*: *all-lits-st-def all-lits-of-mm-union all-lits-def*)
qed

lemma *isa-find-unwatched-between-find-in-list-between-spec*:
  assumes ‹*a* ≤ *length* (*N* ∝ *C*)› **and** ‹*b* ≤ *length* (*N* ∝ *C*)› **and** ‹*a* ≤ *b*› **and**
    ‹*valid-arena arena N vdom*› **and** ‹*C* ∈# *dom-m N*› **and** *eq*: ‹*a*′ = *a*› ‹*b*′ = *b*› ‹*C*′ = *C*› **and**
    ‹⋀*L*. *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹ *P*′ *L* = *P L*› **and**
    M′*M*: ‹(*M*′, *M*) ∈ *trail-pol* $\mathcal{A}$›
  assumes *lits*: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*N* ∝ *C*))›
  shows
    ‹*isa-find-unwatched-between P*′ *M*′ *arena a*′ *b*′ *C*′ ≤ ⇓ *Id* (*find-in-list-between P a b* (*N* ∝ *C*))›
proof −
  have *find-in-list-between-alt*:
    ‹*find-in-list-between P a b C* = *do* {
     (*x*, -) ← *WHILE*$_T$$^{\lambda(found,\ i).\ i \geq a \wedge i \leq length\ C \wedge i \leq b \wedge (\forall j \in \{a..<i\}.\ \neg P\ (C!j)) \wedge}$ (∀ *j*. *found* = *Some j*
        (λ(*found*, *i*). *found* = *None* ∧ *i* < *b*)
        (λ(-, *i*). *do* {
          ASSERT(*i* < *length C*);

```
        let L = C!i;
        if P L then RETURN (Some i, i) else RETURN (None, i+1)
      })
      (None, a);
    RETURN x
  }› for P a b c C
 by (auto simp: find-in-list-between-def)
have [refine0]: ‹((None, x2m + a), None, a) ∈ ⟨Id⟩option-rel ×r {(n', n). n' = x2m + n}›
 for x2m
 by auto
have [simp]: ‹arena-lit arena (C + x2) ∈# 𝓛all 𝒜› if ‹x2 < length (N ∝ C)› for x2
 using that lits assms by (auto simp: arena-lifting
   dest!: literals-are-in-𝓛in-in-𝓛all[of 𝒜 - x2])
have arena-lit-pre: ‹arena-lit-pre arena x2a›
 if
   ‹(x, x') ∈ ⟨nat-rel⟩option-rel ×f {(n', n). n' = C + n}› and
   ‹case x of (found, i) ⇒ found = None ∧ i < C + b› and
   ‹case x' of (found, i) ⇒ found = None ∧ i < b› and
   ‹case x of (found, i) ⇒ True› and
   ‹case x' of
   (found, i) ⇒
     a ≤ i ∧
     i ≤ length (N ∝ C) ∧
     i ≤ b ∧
     (∀j∈{a..<i}. ¬ P (N ∝ C ! j)) ∧
     (∀j. found = Some j ⟶ i = j ∧ P (N ∝ C ! j) ∧ j < b ∧ a ≤ j)› and
   ‹x' = (x1, x2)› and
   ‹x = (x1a, x2a)› and
   ‹x2 < length (N ∝ C)› and
   ‹x2a < C + (arena-length arena C)› and
   ‹C ≤ x2a›
 for x x' x1 x2 x1a x2a
proof −
 show ?thesis
   unfolding arena-lit-pre-def arena-is-valid-clause-idx-and-access-def
   apply (rule bex-leI[of - C])
   apply (rule exI[of - N])
   apply (rule exI[of - vdom])
   using assms that by auto
qed

show ?thesis
 unfolding isa-find-unwatched-between-def find-in-list-between-alt eq
 apply (refine-vcg mop-arena-lit)
 subgoal using assms by (auto dest!: arena-lifting(10))
 subgoal using assms by (auto dest!: arena-lifting(10))
 subgoal by auto
 subgoal by auto
 subgoal using assms by (auto simp: arena-lifting)
 subgoal using assms by (auto simp: arena-lifting)
 subgoal by auto
 subgoal by (rule arena-lit-pre)
 apply (rule assms)
 subgoal using assms by (auto simp: arena-lifting)
 subgoal using assms by (auto simp: arena-lifting)
 subgoal
```

by (*rule polarity-pol-pre*[*OF M'M*]) (*use assms* **in** ‹*auto simp*: *arena-lifting*›)
subgoal using *assms* **by** (*auto simp*: *arena-lifting*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

definition *isa-find-non-false-literal-between* **where**
‹*isa-find-non-false-literal-between M arena a b C* =
isa-find-unwatched-between (λ*L*. *polarity-pol M L* ≠ *Some False*) *M arena a b C*›

definition *find-unwatched*
:: ‹(*nat literal* ⇒ *bool*) ⇒ (*nat*, *nat literal list* × *bool*) *fmap* ⇒ *nat* ⇒ (*nat option*) *nres*› **where**
‹*find-unwatched M N C* = *do* {
ASSERT(*C* ∈# *dom-m N*);
b ← *SPEC*(λ*b*::*bool*. *True*); — non-deterministic between full iteration (used in minisat), or starting in the middle (use in cadical)
if b then find-in-list-between M 2 (*length* (*N* ∝ *C*)) (*N* ∝ *C*)
else do {
pos ← *SPEC* (λ*i*. *i* ≤ *length* (*N* ∝ *C*) ∧ *i* ≥ *2*);
n ← *find-in-list-between M pos* (*length* (*N* ∝ *C*)) (*N* ∝ *C*);
if n = *None then find-in-list-between M 2 pos* (*N* ∝ *C*)
else RETURN n
}
}
›

definition *find-unwatched-wl-st-heur-pre* **where**
‹*find-unwatched-wl-st-heur-pre* =
(λ(*S*, *i*). *arena-is-valid-clause-idx* (*get-clauses-wl-heur S*) *i*)›

definition *find-unwatched-wl-st'*
:: ‹*nat twl-st-wl* ⇒ *nat* ⇒ *nat option nres*› **where**
‹*find-unwatched-wl-st'* = (λ(*M*, *N*, *D*, *Q*, *W*, *vm*, *φ*) *i*. *do* {
find-unwatched (λ*L*. *polarity M L* ≠ *Some False*) *N i*
})›

lemma *find-unwatched-alt-def*:
‹*find-unwatched M N C* = *do* {
ASSERT(*C* ∈# *dom-m N*);
- ← *RETURN*(*length* (*N* ∝ *C*));
b ← *SPEC*(λ*b*::*bool*. *True*); — non-deterministic between full iteration (used in minisat), or starting in the middle (use in cadical)
if b then find-in-list-between M 2 (*length* (*N* ∝ *C*)) (*N* ∝ *C*)
else do {
pos ← *SPEC* (λ*i*. *i* ≤ *length* (*N* ∝ *C*) ∧ *i* ≥ *2*);
n ← *find-in-list-between M pos* (*length* (*N* ∝ *C*)) (*N* ∝ *C*);
if n = *None then find-in-list-between M 2 pos* (*N* ∝ *C*)
else RETURN n
}
}

›
  unfolding *find-unwatched-def* **by** *auto*

lemma *isa-find-unwatched-find-unwatched*:
  assumes *valid*: ‹*valid-arena arena N vdom*› **and**
    ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*N* ∝ *C*))› **and**
    ge2: ‹2 ≤ *length* (*N* ∝ *C*)› **and**
    M'M: ‹(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$›
  shows ‹*isa-find-unwatched P M' arena C* ≤ ⇓ *Id* (*find-unwatched P N C*)›
proof −
  have [*refine0*]:
    ‹*C* ∈# *dom-m N* ⟹ (*l*, *l'*) ∈ {(*l*, *l'*). (*l*, *l'*) ∈ *nat-rel* ∧ *l'* = *length* (*N* ∝ *C*)} ⟹ *RETURN*(*l* ≤
MAX-LENGTH-SHORT-CLAUSE) ≤
      ⇓ {(*b*,*b'*). *b* = *b'* ∧ (*b* ⟷ *is-short-clause* (*N*∝*C*))}
        (*SPEC* (λ-. *True*))›
    for *l l'*
    using *assms*
    by (*auto simp*: *RETURN-RES-refine-iff is-short-clause-def arena-lifting*)
  have [*refine*]: ‹*C* ∈# *dom-m N* ⟹ *mop-arena-length arena C* ≤ *SPEC* (λ*c*. (*c*, *length* (*N* ∝ *C*)) ∈
{(*l*, *l'*). (*l*, *l'*) ∈ *nat-rel* ∧ *l'* = *length* (*N* ∝ *C*)})›
    using *assms* **unfolding** *mop-arena-length-def*
    by *refine-vcg* (*auto simp*: *arena-lifting arena-is-valid-clause-idx-def*)
  show *?thesis*
    unfolding *isa-find-unwatched-def find-unwatched-alt-def*
    apply (*refine-vcg isa-find-unwatched-between-find-in-list-between-spec*[*of* - - - - - *vdom* - - - $\mathcal{A}$ - -])
    apply *assumption*
    subgoal by *auto*
    subgoal using *ge2* **.**
    subgoal by *auto*
    subgoal using *ge2* **.**
    subgoal using *valid* **.**
    subgoal by *fast*
    subgoal using *assms* **by** (*auto simp*: *arena-lifting*)
    subgoal using *assms* **by** *auto*
    subgoal using *assms* **by** (*auto simp*: *arena-lifting*)
    apply (*rule M'M*)
    subgoal using *assms* **by** *auto*
    subgoal using *assms* **unfolding** *get-saved-pos-pre-def arena-is-valid-clause-idx-def*
      by (*auto simp*: *arena-lifting*)
    subgoal using *assms arena-lifting*[*OF valid*] **unfolding** *get-saved-pos-pre-def*
        mop-arena-pos-def
      by (*auto simp*: *arena-lifting arena-pos-def*)
    subgoal by (*auto simp*: *arena-pos-def*)
    subgoal using *assms arena-lifting*[*OF valid*] **by** *auto*
    subgoal using *assms* **by** *auto*
    subgoal using *assms arena-lifting*[*OF valid*] **by** *auto*
    subgoal using *assms* **by** *auto*
    subgoal using *assms* **by** (*auto simp*: *arena-lifting*)
    subgoal using *assms* **by** *auto*
    subgoal using *assms arena-lifting*[*OF valid*] **by** *auto*
    apply (*rule M'M*)
    subgoal using *assms* **by** *auto*
    subgoal using *assms* **by** *auto*
    subgoal using *assms* **by** *auto*
    subgoal using *assms arena-lifting*[*OF valid*] **by** *auto*

subgoal by (*auto simp*: *arena-pos-def*)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
apply (*rule M'M*)
subgoal using *assms* **by** *auto*
done
qed

lemma *find-unwatched*:
assumes *n-d*: ‹*no-dup M*› **and** ‹*length* (*N* ∝ *C*) ≥ *2*› **and** ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*N* ∝ *C*))›
shows ‹*find-unwatched* (λ*L*. *polarity M L* ≠ *Some False*) *N C* ≤ ⇓ *Id* (*find-unwatched-l M N C*)›
proof −
have [*refine0*]: ‹*find-in-list-between* (λ*L*. *polarity M L* ≠ *Some False*) *2* (*length* (*N* ∝ *C*)) (*N* ∝ *C*)
≤ *SPEC*
(λ*found*.
(*found* = *None*) = (∀ *L*∈*set* (*unwatched-l* (*N* ∝ *C*)). − *L* ∈ *lits-of-l M*) ∧
(∀ *j*. *found* = *Some j* ⟶
j < *length* (*N* ∝ *C*) ∧
(*undefined-lit M* ((*N* ∝ *C*) ! *j*) ∨ (*N* ∝ *C*) ! *j* ∈ *lits-of-l M*) ∧ *2* ≤ *j*))›
proof −
show *?thesis*
apply (*rule order-trans*)
apply (*rule find-in-list-between-spec*)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal
using *n-d*
by (*auto simp add*: *polarity-def in-set-drop-conv-nth Ball-def*
Decided-Propagated-in-iff-in-lits-of-l split: *if-splits dest*: *no-dup-consistentD*)
done
qed
have [*refine0*]: ‹*find-in-list-between* (λ*L*. *polarity M L* ≠ *Some False*) *xa* (*length* (*N* ∝ *C*)) (*N* ∝ *C*)
≤ *SPEC*
(λ*n*. (*if n* = *None*
then find-in-list-between (λ*L*. *polarity M L* ≠ *Some False*) *2 xa* (*N* ∝ *C*)
else RETURN n)
≤ *SPEC*
(λ*found*.
(*found* = *None*) =
(∀ *L*∈*set* (*unwatched-l* (*N* ∝ *C*)). − *L* ∈ *lits-of-l M*) ∧
(∀ *j*. *found* = *Some j* ⟶
j < *length* (*N* ∝ *C*) ∧
(*undefined-lit M* ((*N* ∝ *C*) ! *j*) ∨ (*N* ∝ *C*) ! *j* ∈ *lits-of-l M*) ∧
2 ≤ *j*)))›
if
‹*xa* ≤ *length* (*N* ∝ *C*) ∧ *2* ≤ *xa*›
for *xa*
proof −
show *?thesis*
apply (*rule order-trans*)
apply (*rule find-in-list-between-spec*)

```
    subgoal using that by auto
    subgoal using assms by auto
    subgoal using that by auto
    subgoal
      apply (rule SPEC-rule)
      subgoal for x
        apply (cases ‹x = None›; simp only: if-True if-False refl)
      subgoal
        apply (rule order-trans)
        apply (rule find-in-list-between-spec)
        subgoal using that by auto
        subgoal using that by auto
        subgoal using that by auto
        subgoal
          apply (rule SPEC-rule)
          apply (intro impI conjI iffI ballI)
          unfolding in-set-drop-conv-nth Ball-def
          apply normalize-goal
          subgoal for x L xaa
            apply (cases ‹xaa ≥ xa›)
            subgoal
              using n-d
              by (auto simp add: polarity-def  Ball-def all-conj-distrib
              Decided-Propagated-in-iff-in-lits-of-l split: if-splits dest: no-dup-consistentD)
            subgoal
              using n-d
              by (auto simp add: polarity-def  Ball-def all-conj-distrib
              Decided-Propagated-in-iff-in-lits-of-l split: if-splits dest: no-dup-consistentD)
            done
          subgoal for x
            using n-d that assms
            apply (auto simp add: polarity-def  Ball-def all-conj-distrib
            Decided-Propagated-in-iff-in-lits-of-l split: if-splits dest: no-dup-consistentD,
              force)
            by (blast intro: dual-order.strict-trans1 dest: no-dup-consistentD)
          subgoal
            using n-d assms that
            by (auto simp add: polarity-def Ball-def all-conj-distrib
              Decided-Propagated-in-iff-in-lits-of-l
                split: if-splits dest: no-dup-consistentD)
          done
        done
      subgoal
        using n-d that assms le-trans
        by (auto simp add: polarity-def  Ball-def all-conj-distrib in-set-drop-conv-nth
            Decided-Propagated-in-iff-in-lits-of-l split: if-splits dest: no-dup-consistentD)
          (use le-trans no-dup-consistentD in blast)+
      done
    done
  done
qed

show ?thesis
  unfolding find-unwatched-def find-unwatched-l-def
  apply (refine-vcg)
  subgoal by blast
```

subgoal by *blast*
subgoal by *blast*
done
qed

definition *find-unwatched-wl-st-pre* **where**
‹*find-unwatched-wl-st-pre* = (λ(*S*, *i*).
i ∈# *dom-m* (*get-clauses-wl S*) ∧ 2 ≤ *length* (*get-clauses-wl S* ∝ *i*) ∧
literals-are-in-$\mathcal{L}_{in}$ (*all-atms-st S*) (*mset* (*get-clauses-wl S* ∝ *i*))
)›

theorem *find-unwatched-wl-st-heur-find-unwatched-wl-s*:
‹(*uncurry isa-find-unwatched-wl-st-heur*, *uncurry find-unwatched-wl-st'*)
∈ [*find-unwatched-wl-st-pre*]$_f$
twl-st-heur ×$_f$ *nat-rel* → ⟨*Id*⟩*nres-rel*›
proof −
have [*refine0*]: ‹((*None*, *x2m* + 2), *None*, 2) ∈ ⟨*Id*⟩*option-rel* ×$_r$ {(*n'*, *n*). *n'* = *x2m* + *n*}›
for *x2m*
by *auto*
have [*refine0*]:
‹(*polarity M* (*arena-lit arena i'*), *polarity M'* (*N* ∝ *C'* ! *j*)) ∈ ⟨*Id*⟩*option-rel*›
if ‹∃ *vdom*. *valid-arena arena N vdom*› **and**
‹*C'* ∈# *dom-m N*› **and**
‹*i'* = *C'* + *j* ∧ *j* < *length* (*N* ∝ *C'*)› **and**
‹*M* = *M'*›
for *M arena i i' N j M' C'*
using *that* **by** (*auto simp*: *arena-lifting*)
have [*refine0*]: ‹*RETURN* (*arena-pos arena C*) ≤ ⇓ {(*pos*, *pos'*). *pos* = *pos'* ∧ *pos* ≥ 2 ∧ *pos* ≤ *length* (*N* ∝ *C*)}
(*SPEC* (λ*i*. *i* ≤ *length* (*N* ∝ *C'*) ∧ 2 ≤ *i*))›
if *valid*: ‹*valid-arena arena N vdom*› **and** *C*: ‹*C* ∈# *dom-m N*› **and** ‹*C* = *C'*› **and**
‹*is-long-clause* (*N* ∝ *C'*)›
for *arena N vdom C C'*
using *that arena-lifting*[*OF valid C*] **by** (*auto simp*: *RETURN-RES-refine-iff*
arena-pos-def)
have [*refine0*]:
‹*RETURN* (*arena-length arena C* ≤ *MAX-LENGTH-SHORT-CLAUSE*) ≤ ⇓ {(*b*, *b'*). *b* = *b'* ∧ (*b* ⟷ *is-short-clause* (*N* ∝ *C*))}
(*SPEC*(λ-. *True*))›
if *valid*: ‹*valid-arena arena N vdom*› **and** *C*: ‹*C* ∈# *dom-m N*›
for *arena N vdom C*
using *that arena-lifting*[*OF valid C*] **by** (*auto simp*: *RETURN-RES-refine-iff is-short-clause-def*)

have [*refine0*]:
‹*C* ∈# *dom-m N* ⟹ (*l*, *l'*) ∈ {(*l*, *l'*). (*l*, *l'*) ∈ *nat-rel* ∧ *l'* = *length* (*N* ∝ *C*)} ⟹ *RETURN*(*l* ≤ *MAX-LENGTH-SHORT-CLAUSE*) ≤
⇓ {(*b*,*b'*). *b* = *b'* ∧ (*b* ⟷ *is-short-clause* (*N*∝*C*))}
(*SPEC* (λ-. *True*))›
for *l l' C N*
by (*auto simp*: *RETURN-RES-refine-iff is-short-clause-def arena-lifting*)
have [*refine*]: ‹*C* ∈# *dom-m N* ⟹ *valid-arena arena N vdom* ⟹
mop-arena-length arena C ≤ *SPEC* (λ*c*. (*c*, *length* (*N* ∝ *C*)) ∈ {(*l*, *l'*). (*l*, *l'*) ∈ *nat-rel* ∧ *l'* = *length* (*N* ∝ *C*)})›
for *N C arena vdom*
unfolding *mop-arena-length-def*
by *refine-vcg* (*auto simp*: *arena-lifting arena-is-valid-clause-idx-def*)

```
  have H: ‹isa-find-unwatched P M' arena C ≤ ⇓ Id (find-unwatched P' N C')›
    if ‹valid-arena arena N vdom›
      ‹⋀L. L ∈# 𝓛_all 𝒜 ⟹ P L = P' L› and
      ‹C = C'› and
      ‹2 ≤ length (N ∝ C')› and ‹literals-are-in-𝓛_in 𝒜 (mset (N ∝ C'))› and
      ‹(M', M) ∈ trail-pol 𝒜›
    for arena P N C vdom P' C' 𝒜 M' M
    using that unfolding isa-find-unwatched-def find-unwatched-alt-def supply [[goals-limit=1]]
   apply (refine-vcg isa-find-unwatched-between-find-in-list-between-spec[of - - - - - vdom, where 𝒜=𝒜])
    unfolding that apply assumption+
    subgoal by simp
    subgoal by auto
    subgoal using that by (simp add: arena-lifting)
    subgoal using that by auto
    subgoal using that by (auto simp: arena-lifting)
    apply assumption
    subgoal using that by (auto simp: arena-lifting get-saved-pos-pre-def
       arena-is-valid-clause-idx-def)
    subgoal using arena-lifting[OF ‹valid-arena arena N vdom›] unfolding get-saved-pos-pre-def
        mop-arena-pos-def
      by (auto simp: arena-lifting arena-pos-def)
    subgoal using that by (auto simp: arena-lifting)
    subgoal using that by (auto simp: arena-lifting)
    subgoal using that by (auto simp: arena-lifting)
    subgoal using that by (auto simp: arena-lifting)
    subgoal using that by (auto simp: arena-lifting)
    subgoal using that by (auto simp: arena-lifting)
    apply assumption
    subgoal using that by (auto simp: arena-lifting)
    subgoal using that by (auto simp: arena-lifting)
    subgoal using that by (auto simp: arena-lifting)
    subgoal using that by (auto simp: arena-lifting)
    subgoal using that by (auto simp: arena-lifting)
    subgoal using that by (auto simp: arena-lifting)
    apply assumption
    done

  show ?thesis
    unfolding isa-find-unwatched-wl-st-heur-def find-unwatched-wl-st'-def
       uncurry-def twl-st-heur-def
      find-unwatched-wl-st-pre-def
    apply (intro frefI nres-relI)
    apply refine-vcg
    subgoal for x y
      apply (case-tac x, case-tac y)
      by (rule H[where 𝒜3 = ‹all-atms-st (fst y)›, of - - ‹set (get-vdom (fst x))›])
        (auto simp: polarity-pol-polarity[of ‹all-atms-st (fst y)›,
     unfolded option-rel-id-simp, THEN fref-to-Down-unRET-uncurry-Id]
      all-atms-def[symmetric] literals-are-in-𝓛_in-nth2)
    done
qed


lemma isa-save-pos-is-Id:
  assumes
```

‹(S, T) ∈ twl-st-heur›
‹C ∈# dom-m (get-clauses-wl T)› **and**
‹i ≤ length (get-clauses-wl T ∝ C)› **and**
‹i ≥ 2›
shows ‹isa-save-pos C i S ≤ ⇓ {(S′, T′). (S′, T′) ∈ twl-st-heur ∧ length (get-clauses-wl-heur S′) = length (get-clauses-wl-heur S) ∧
get-watched-wl-heur S′ = get-watched-wl-heur S ∧ get-vdom S′ = get-vdom S ∧ get-learned-count S′ = get-learned-count S}
(RETURN T)›
proof −
have ‹isa-update-pos-pre ((C, i), get-clauses-wl-heur S)› **if** ‹is-long-clause (get-clauses-wl T ∝ C)›
unfolding *isa-update-pos-pre-def*
using *assms that*
by (*cases S*; *cases T*)
(*auto simp*: *isa-save-pos-def twl-st-heur-def arena-update-pos-alt-def*
isa-update-pos-pre-def arena-is-valid-clause-idx-def arena-lifting)
then show *?thesis*
using *assms*
by (*cases S*; *cases T*)
(*auto simp*: *isa-save-pos-def twl-st-heur-def arena-update-pos-alt-def*
isa-update-pos-pre-def arena-is-valid-clause-idx-def arena-lifting
intro!: *valid-arena-update-pos ASSERT-leI*)
qed

14.2 Updates

lemma *isa-vmtf-bump-to-rescore-also-reasons-cl-isa-vmtf*:
assumes ‹(M,M′)∈trail-pol 𝒜› ‹isasat-input-bounded 𝒜› **and** *vm*: ‹vm ∈ bump-heur 𝒜 M′› **and**
valid: ‹valid-arena N N′ vd› **and**
C: ‹C ∈# dom-m N′› **and**
H:‹ ∀ L∈set (N′ ∝ C). L ∈# ℒ_all 𝒜›
‹∀ L∈set (N′ ∝ C).
∀ C. Propagated (− L) C ∈ set M′ ⟶
C ≠ 0 ⟶ C ∈# dom-m N′ ∧ (∀ C∈set [C..<C + arena-length N C]. arena-lit N C ∈# ℒ_all 𝒜)›
and
bound: ‹isasat-input-bounded 𝒜›
shows
‹isa-vmtf-bump-to-rescore-also-reasons-cl M N C L vm ≤ RES (bump-heur 𝒜 M′)›
proof −
show *?thesis*
apply (*rule order-trans*)
apply (*rule isa-vmtf-bump-to-rescore-also-reasons-cl-vmtf-mark-to-rescore-also-reasons-cl*[
where 𝒜 = 𝒜,
THEN fref-to-Down-curry4,
of - - - - - ‹M′› ‹N› C L vm])
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule vmtf-mark-to-rescore-also-reasons-cl-spec*[*OF vm valid C bound H*])
subgoal by (*auto simp*: *conc-fun-RES*)
done
done
qed

lemma *mark-conflict-to-rescore*:
assumes ‹(S,T)∈*twl-st-heur-up''* $\mathcal{D}$ *r s K lcount*›
‹*set-conflict-wl-pre C T*›
shows ‹*mark-conflict-to-rescore C S* ≤ *SPEC*(λS'. (S', *T*)∈ *twl-st-heur-up''* $\mathcal{D}$ *r s K lcount*)›
proof −
obtain *U V b* **where**
TU: ‹(*T*, *U*) ∈ *state-wl-l b*› **and**
C: ‹*C* ∈# *dom-m* (*get-clauses-wl T*)› **and**
list: ‹*twl-list-invs U*›
using *assms*(2) **unfolding** *set-conflict-wl-pre-def set-conflict-l-pre-def* **apply** −
apply *normalize-goal+*
by *auto*
have *valid*: ‹*valid-arena* (*get-clauses-wl-heur S*) (*get-clauses-wl T*) (*set* (*get-vdom S*))› **and**
vm: ‹*get-vmtf-heur S* ∈ *bump-heur* (*all-atms-st T*) (*get-trail-wl T*)› **and**
bounded: ‹*isasat-input-bounded* (*all-atms-st T*)› **and**
trail: ‹ (*get-trail-wl-heur S*, *get-trail-wl T*) ∈ *trail-pol* (*all-atms-st T*)›
using *assms* **unfolding** *arena-is-valid-clause-idx-def* **unfolding** *twl-st-heur'-def*
twl-st-heur-def **by** *auto*
have *H*:‹ ∀ *L*∈*set* (*get-clauses-wl T* ∝ *C*). *L* ∈# $\mathcal{L}_{all}$ (*all-atms-st T*)›
‹∀ *L*∈*set* (*get-clauses-wl T* ∝ *C*).
∀ *C*. *Propagated* (− *L*) *C* ∈ *set* (*get-trail-wl T*) ⟶
C ≠ *0* ⟶ *C* ∈# *dom-m* (*get-clauses-wl T*) ∧ (∀ *C*∈*set* [*C*..<*C* + *arena-length* (*get-clauses-wl-heur S*) *C*]. *arena-lit* (*get-clauses-wl-heur S*) *C* ∈# $\mathcal{L}_{all}$ (*all-atms-st T*))›
subgoal
using *valid C* **by** (*auto simp*: *arena-lifting in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$
all-atms-st-def all-atms-def all-lits-def all-lits-of-mm-union ran-m-def all-lits-of-mm-add-mset image-Un
atm-of-all-lits-of-m(2)
dest!: *multi-member-split*[*of C*]
simp del: *all-atms-def*[*symmetric*])[]
apply (*intro ballI allI*)
subgoal for *L D*
apply (*intro conjI ballI impI*)
subgoal
using *TU list* **by** (*auto simp*: *twl-list-invs-def dest*!: *spec*[*of* - ‹−*L*›] *spec*[*of* - ‹*D*›])
subgoal for *C*
apply (*subgoal-tac* ‹*C* − *D* < *length* (*get-clauses-wl T* ∝ *D*)›)
using *TU list arena-lifting*(5)[*OF valid*, *of D* ‹*C* − *D*›,*symmetric*]
apply (*auto simp*: *twl-list-invs-def dest*!: *spec*[*of* - ‹−*L*›] *spec*[*of* - ‹*D*›])
using *valid*
apply (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$
all-atms-st-def all-atms-def all-lits-def all-lits-of-mm-union ran-m-def all-lits-of-mm-add-mset image-Un
atm-of-all-lits-of-m(2) *arena-lifting*
dest!: *multi-member-split*[*of D*] *spec*[*of* - ‹−*L*›] *spec*[*of* - ‹*D*›]
simp del: *all-atms-def*[*symmetric*])
apply (*metis* (*no-types*, *opaque-lifting*) ‹⟦*D* ∈# *dom-m* (*get-clauses-wl T*); *C* − *D* < *length* (*get-clauses-wl T* ∝ *D*)⟧ ⟹ *arena-lit* (*get-clauses-wl-heur S*) (*D* + (*C* − *D*)) = *get-clauses-wl T* ∝ *D* ! (*C* − *D*)› *add.commute add.right-neutral add-diff-inverse-nat arena-lifting*(4) *imageI less-diff-conv2 less-nat-zero-code member-add-mset nth-mem*)
apply (*metis* (*no-types*, *opaque-lifting*) ‹⟦*D* ∈# *dom-m* (*get-clauses-wl T*); *C* − *D* < *length* (*get-clauses-wl T* ∝ *D*)⟧ ⟹ *arena-lit* (*get-clauses-wl-heur S*) (*D* + (*C* − *D*)) = *get-clauses-wl T* ∝ *D* ! (*C* − *D*)› *add.commute add.right-neutral add-diff-inverse-nat arena-lifting*(4) *imageI less-diff-conv2 less-nat-zero-code member-add-mset nth-mem*)
done

done
done
have *H'*: ‹*literals-are-in-$\mathcal{L}_{in}$* (*all-atms-st T*) (*mset* (*get-clauses-wl T* ∝ *C*))›
by (*simp add*: *C literals-are-in-$\mathcal{L}_{in}$-nth2*)
show *?thesis*
unfolding *mark-conflict-to-rescore-def mop-arena-length-def nres-monad3 mop-arena-lit2-def*
apply (*refine-vcg WHILET-rule*[**where** Φ = ‹λ(*i*,*vm*). *vm* ∈ *bump-heur* (*all-atms-st T*) (*get-trail-wl T*)› **and**
I = ‹λ(*i*,*vm*). *i* ≤ *length* (*get-clauses-wl T* ∝ *C*) ∧ *vm* ∈ *bump-heur* (*all-atms-st T*) (*get-trail-wl T*)› **and**
R = ‹*measure* (λ(*i*,*vm*). *length* (*get-clauses-wl T* ∝ *C*) −*i*)›] *trail bounded valid*
isa-vmtf-bump-to-rescore-also-reasons-cl-isa-vmtf[*THEN order-trans*]
calculate-LBD-heur-st-calculate-LBD-st[**where**
vdom = ‹*set* (*get-vdom* (*S*))› **and** $\mathcal{A}$ = ‹*all-atms-st T*› **and** *C'*=*C*, *unfolded calculate-LBD-st-def conc-fun-RES RETURN-def*, *THEN order-trans*])
subgoal using *C valid* **unfolding** *arena-is-valid-clause-idx-def* **by** *auto*
subgoal using *valid C arena-lifting*(*7*)[*OF valid C*, *of* ‹*length* (*get-clauses-wl T* ∝ *C*) − *1*›] **by** (*auto simp*: *arena-lifting*)
subgoal by *auto*
subgoal by *auto*
subgoal using *vm* **by** *auto*
subgoal by *auto*
subgoal using *valid C* **by** (*auto simp*:*arena-lifting arena-lit-pre-def*
arena-is-valid-clause-idx-and-access-def intro!: *exI*[*of* - *C*] *exI*[*of* - ‹*get-clauses-wl T*›])
subgoal by *auto*
subgoal using *C* **by** *auto*
subgoal by (*rule H*)
subgoal by (*rule H*)
subgoal using *valid C* **by** (*auto simp*: *arena-lifting*)
subgoal by *auto*
subgoal using *valid C* **by** (*auto simp*: *arena-lifting*)
subgoal by *auto*
subgoal using *C* **by** *auto*
subgoal by (*rule H'*)
subgoal by *auto*
subgoal using *assms* **unfolding** *twl-st-heur'-def twl-st-heur-def* **by** *auto*
done
qed

definition *set-conflict-wl-heur-pre* **where**
‹*set-conflict-wl-heur-pre* =
(λ(*C*, *S*). *True*)›

definition *update-clause-wl-pre* **where**
‹*update-clause-wl-pre K r* = (λ(((((((*L*, *L'*), *C*), *b*), *j*), *w*), *i*), *f*), *S*).
L = *K*)›
lemma *arena-lit-pre*:
‹*valid-arena NU N vdom* ⟹ *C* ∈# *dom-m N* ⟹ *i* < *length* (*N* ∝ *C*) ⟹ *arena-lit-pre NU* (*C* + *i*)›
unfolding *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
by (*rule bex-leI*[*of* - *C*], *rule exI*[*of* - *N*], *rule exI*[*of* - *vdom*]) *auto*

lemma *all-atms-swap*[*simp*]:
‹*C* ∈# *dom-m N* ⟹ *i* < *length* (*N* ∝ *C*) ⟹ *j* < *length* (*N* ∝ *C*) ⟹

all-atms (*N*(*C* ↪ *swap* (*N* ∝ *C*) *i j*)
) = *all-atms N*›
unfolding *all-atms-def*
by (*auto simp del*: *all-atms-def*[*symmetric*] *simp*: *all-atms-def intro*!: *ext*)

lemma *mop-arena-swap*[*mop-arena-lit*]:
assumes *valid*: ‹*valid-arena arena N vdom*› **and**
i: ‹(*C*, *C'*) ∈ *nat-rel*› ‹(*i*, *i'*) ∈ *nat-rel*› ‹(*j*, *j'*) ∈ *nat-rel*›
shows
‹*mop-arena-swap C i j arena* ≤ ⇓{(*N''*, *N'*). *valid-arena N'' N' vdom* ∧ *N''* = *swap-lits C' i' j'*
arena
∧ *N'* = *op-clauses-swap N C' i' j'* ∧ *all-atms N'* = *all-atms N* ∧ *i'* < *length* (*N* ∝ *C'*) ∧
j' < *length* (*N* ∝ *C'*)} (*mop-clauses-swap N C' i' j'*)›
using *assms* **unfolding** *mop-clauses-swap-def mop-arena-swap-def swap-lits-pre-def*
by *refine-rcg*
(*auto simp*: *arena-lifting valid-arena-swap-lits op-clauses-swap-def*)

lemma *update-clause-wl-alt-def*:
‹*update-clause-wl* = (λ(*L*::'*v literal*) *L' C b j w i f* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*,
Q, *W*). *do* {
ASSERT(*C* ∈# *dom-m N* ∧ *j* ≤ *w* ∧ *w* < *length* (*W L*) ∧
correct-watching-except (*Suc j*) (*Suc w*) *L* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*,
W));
ASSERT(*L* ∈# *all-lits-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*));
ASSERT(*L'* ∈# *all-lits-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*));
K' ← *mop-clauses-at N C f*;
ASSERT(*K'* ∈# *all-lits-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*) ∧ *L* ≠ *K'*);
N' ← *mop-clauses-swap N C i f*;
RETURN (*j*, *w*+*1*, (*M*, *N'*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*(*K'* := *W K'* @ [(*C*,
L, *b*)]))))
})›
unfolding *update-clause-wl-def* **by** (*auto intro*!: *ext bind-cong*[*OF refl*] *simp flip*: *all-lits-alt-def2*)

lemma *all-atms-st-simps*[*simp*]:
‹*all-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*(*K* := *WK*)) =
all-atms-st (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*)›
‹*all-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *add-mset K Q*, *W*) =
all-atms-st (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*)› — actually covered below, but
still useful for 'unfolding' by hand
‹*x1* ∈# *dom-m x1aa* ⟹ *n* < *length* (*x1aa* ∝ *x1*) ⟹ *n'* < *length* (*x1aa* ∝ *x1*) ⟹
all-atms-st (*x1b*, *x1aa*(*x1* ↪ *WB-More-Refinement-List.swap* (*x1aa* ∝ *x1*) *n n'*), *D*, *x1c*, *x1d*, *NEk*,
UEk, *NS*, *US*, *N0*, *U0*, *x1e*,
x2e) =
all-atms-st
(*x1b*, *x1aa*, *D*, *x1c*, *x1d*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *x1e*,
x2e)›
‹*all-atms-st* (*L* # *M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*) =
all-atms-st (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*)›
‹*NO-MATCH* {#} *Q* ⟹ *all-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*) =
all-atms-st (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, {#}, *W*)›
‹*NO-MATCH* [] *M* ⟹ *all-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*) =
all-atms-st ([], *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*)›
‹*NO-MATCH None D* ⟹ *all-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*) =
all-atms-st (*M*, *N*, *None*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*)›
‹*all-atms-st* (*set-literals-to-update-wl WS S*) = *all-atms-st S*›
by (*cases S*; *auto simp*: *all-atms-st-def all-atms-def ran-m-clause-upd*

image-mset-remove1-mset-if simp del: *all-atms-def*[*symmetric*]; *fail*)+

lemma *update-clause-wl-heur-update-clause-wl*:
‹(*uncurry8 update-clause-wl-heur*, *uncurry8* (*update-clause-wl*)) ∈
[*update-clause-wl-pre K r*]$_f$
Id $\times_f$ *Id* $\times_f$ *nat-rel* $\times_f$ *bool-rel* $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *nat-rel* $\times_f$ *twl-st-heur-up''* $\mathcal{D}$ *r s K lcount* →
⟨*nat-rel* $\times_r$ *nat-rel* $\times_r$ *twl-st-heur-up''* $\mathcal{D}$ *r s K lcount*⟩*nres-rel*›
unfolding *update-clause-wl-heur-def update-clause-wl-alt-def uncurry-def*
update-clause-wl-pre-def all-lits-of-all-atms-of all-lits-of-all-atms-of Let-def
apply (*intro frefI nres-relI*, *case-tac x*, *case-tac y*)
apply (*refine-rcg*)
apply (*rule mop-arena-lit2'*)
subgoal by (*auto 0 0 simp*: *update-clause-wl-heur-def update-clause-wl-def twl-st-heur-def Let-def*
map-fun-rel-def2 twl-st-heur'-def update-clause-wl-pre-def arena-lifting arena-lit-pre-def
arena-is-valid-clause-idx-and-access-def swap-lits-pre-def
intro!: *ASSERT-refine-left valid-arena-swap-lits*
intro!: *bex-leI exI*)
subgoal by *auto*
subgoal by *auto*
subgoal by
(*auto 0 0 simp*: *update-clause-wl-heur-def update-clause-wl-def twl-st-heur-def Let-def*
map-fun-rel-def2 twl-st-heur'-def update-clause-wl-pre-def arena-lifting arena-lit-pre-def
arena-is-valid-clause-idx-and-access-def swap-lits-pre-def
intro!: *ASSERT-refine-left valid-arena-swap-lits*
intro!: *bex-leI exI*)
apply (*rule-tac vdom*= ‹*set* (*get-vdom* ((λ(((((((*L*,*C*),*b*),*j*),*w*),-),-),*x*). *x*) *x*))› **in** *mop-arena-swap*)
subgoal
by (*auto 0 0 simp*: *twl-st-heur-def Let-def*
map-fun-rel-def2 twl-st-heur'-def update-clause-wl-pre-def arena-lifting arena-lit-pre-def
intro!: *ASSERT-refine-left valid-arena-swap-lits dest*!: *multi-member-split*[*of* ‹*arena-lit* - -›])
subgoal
by (*auto 0 0 simp*: *twl-st-heur-def Let-def*
map-fun-rel-def2 twl-st-heur'-def update-clause-wl-def arena-lifting arena-lit-pre-def
intro!: *ASSERT-refine-left valid-arena-swap-lits dest*!: *multi-member-split*[*of* ‹*arena-lit* - -›])
subgoal
by (*auto 0 0 simp*: *twl-st-heur-def Let-def*
map-fun-rel-def2 twl-st-heur'-def update-clause-wl-def arena-lifting arena-lit-pre-def
intro!: *ASSERT-refine-left valid-arena-swap-lits dest*!: *multi-member-split*[*of* ‹*arena-lit* - -›])
subgoal
by (*auto 0 0 simp*: *twl-st-heur-def Let-def*
map-fun-rel-def2 twl-st-heur'-def update-clause-wl-pre-def arena-lifting arena-lit-pre-def
intro!: *ASSERT-refine-left valid-arena-swap-lits dest*!: *multi-member-split*[*of* ‹*arena-lit* - -›])
subgoal
by (*auto simp*: *twl-st-heur-def Let-def add-mset-eq-add-mset all-lits-of-all-atms-of ac-simps*
twl-st-heur'-def update-clause-wl-pre-def arena-lifting arena-lit-pre-def map-fun-rel-def2
dest: *multi-member-split simp flip*: *all-lits-def all-lits-alt-def2*
intro!: *ASSERT-refine-left valid-arena-swap-lits*)
subgoal for *x y a b aa ba c d e f g h i j k l m n x1 x1a x1b x1c x1d x1e x1f x2 x2a x2b x2c x2d x2e x2f x1g x2g x1h x2h x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m*
x1n x2n x1o x2o x1p x2p x1q x2q x1r x2r x1s x1t x1u x1v x1w x1x x1y x2s x2t x2u x2v x2w x2x x2y - - - - *K' K'a N' N'a*
by (*auto dest*!: *length-watched-le2*[*of* - - - - ‹*b*› $\mathcal{D}$ *r lcount K'a*])
(*simp-all add*: *twl-st-heur'-def twl-st-heur-def map-fun-rel-def2*)
subgoal

by
(*clarsimp simp*: *twl-st-heur-def Let-def*
map-fun-rel-def2 twl-st-heur'-def update-clause-wl-pre-def
op-clauses-swap-def)
done

definition *propagate-lit-wl-heur-pre* **where**
‹*propagate-lit-wl-heur-pre* =
(λ((*L*, *C*), *S*). *C* ≠ *DECISION-REASON*)›

definition *propagate-lit-wl-pre* **where**
‹*propagate-lit-wl-pre* = (λ(((*L*, *C*), *i*), *S*).
undefined-lit (*get-trail-wl S*) *L* ∧ *get-conflict-wl S* = *None* ∧
C ∈# *dom-m* (*get-clauses-wl S*) ∧ *L* ∈# *all-lits-st S* ∧
1 − *i* < *length* (*get-clauses-wl S* ∝ *C*) ∧
0 < *length* (*get-clauses-wl S* ∝ *C*))›

lemma *propagate-lit-wl-heur-propagate-lit-wl*:
‹(*uncurry3 propagate-lit-wl-heur*, *uncurry3* (*propagate-lit-wl*)) ∈
[λ-. *True*]$_f$
Id ×$_f$ *nat-rel* ×$_f$ *nat-rel* ×$_f$ *twl-st-heur-up''* 𝒟 *r s K lcount* → ⟨*twl-st-heur-up''* 𝒟 *r s K lcount*⟩*nres-rel*›
supply [[*goals-limit=1*]]
unfolding *propagate-lit-wl-heur-def propagate-lit-wl-def Let-def*
apply (*intro frefI nres-relI*) **unfolding** *uncurry-def mop-save-phase-heur-def*
nres-monad3
apply (*refine-rcg*)
subgoal by *auto*
apply (*rule-tac* 𝒜 = ‹*all-atms-st* (*snd y*)› **in** *cons-trail-Propagated-tr2*)
subgoal by (*auto 4 3 simp*: *twl-st-heur-def propagate-lit-wl-heur-def propagate-lit-wl-def*
isa-vmtf-consD twl-st-heur'-def propagate-lit-wl-pre-def swap-lits-pre-def
valid-arena-swap-lits arena-lifting phase-saving-def atms-of-def save-phase-def
ac-simps
intro!: *ASSERT-refine-left cons-trail-Propagated-tr2 cons-trail-Propagated-tr-pre*
dest: *multi-member-split valid-arena-DECISION-REASON*)
subgoal
by (*auto simp*: *twl-st-heur-def twl-st-heur'-def all-lits-st-alt-def*[*symmetric*]
ac-simps)
subgoal by (*auto 4 3 simp*: *twl-st-heur-def propagate-lit-wl-heur-def propagate-lit-wl-def*
isa-vmtf-consD twl-st-heur'-def propagate-lit-wl-pre-def swap-lits-pre-def
valid-arena-swap-lits arena-lifting phase-saving-def atms-of-def save-phase-def
intro!: *ASSERT-refine-left cons-trail-Propagated-tr2 cons-trail-Propagated-tr-pre*
dest: *multi-member-split valid-arena-DECISION-REASON*)
apply (*rule-tac vdom* = ‹*set* (*get-vdom* (*snd x*))› **in** *mop-arena-swap*)
subgoal by (*auto 4 3 simp*: *twl-st-heur-def propagate-lit-wl-heur-def propagate-lit-wl-def*
isa-vmtf-consD twl-st-heur'-def propagate-lit-wl-pre-def swap-lits-pre-def
valid-arena-swap-lits arena-lifting phase-saving-def atms-of-def save-phase-def
intro!: *ASSERT-refine-left cons-trail-Propagated-tr2 cons-trail-Propagated-tr-pre*
dest: *multi-member-split valid-arena-DECISION-REASON*)
subgoal by (*auto 4 3 simp*: *twl-st-heur-def propagate-lit-wl-heur-def propagate-lit-wl-def*
isa-vmtf-consD twl-st-heur'-def propagate-lit-wl-pre-def swap-lits-pre-def
valid-arena-swap-lits arena-lifting phase-saving-def atms-of-def save-phase-def
intro!: *ASSERT-refine-left cons-trail-Propagated-tr2 cons-trail-Propagated-tr-pre*
dest: *multi-member-split valid-arena-DECISION-REASON*)
subgoal by (*auto 4 3 simp*: *twl-st-heur-def propagate-lit-wl-heur-def propagate-lit-wl-def*

isa-vmtf-consD twl-st-heur'-def propagate-lit-wl-pre-def swap-lits-pre-def
valid-arena-swap-lits arena-lifting phase-saving-def atms-of-def save-phase-def
intro!: *ASSERT-refine-left cons-trail-Propagated-tr2 cons-trail-Propagated-tr-pre*
dest: *multi-member-split valid-arena-DECISION-REASON*)
subgoal by (*auto simp*: *twl-st-heur-def propagate-lit-wl-heur-def propagate-lit-wl-def*
isa-vmtf-consD twl-st-heur'-def propagate-lit-wl-pre-def swap-lits-pre-def
valid-arena-swap-lits arena-lifting phase-saving-def atms-of-def save-phase-def
intro!: *ASSERT-refine-left cons-trail-Propagated-tr2 cons-trail-Propagated-tr-pre*
dest: *multi-member-split valid-arena-DECISION-REASON*)
subgoal by (*auto simp*: *twl-st-heur-def propagate-lit-wl-heur-def propagate-lit-wl-def*
isa-vmtf-consD twl-st-heur'-def propagate-lit-wl-pre-def swap-lits-pre-def
valid-arena-swap-lits arena-lifting phase-saving-def
all-lits-st-alt-def[*symmetric*] *ac-simps*
intro!: *save-phase-heur-preI*)
subgoal for *x y*
by (*cases x*; *cases y*; *hypsubst*)
(*clarsimp simp add*: *twl-st-heur-def twl-st-heur'-def isa-vmtf-consD*
op-clauses-swap-def ac-simps)
done

definition *propagate-lit-wl-bin-pre* **where**
‹*propagate-lit-wl-bin-pre* = (λ(((*L*, *C*), *i*), *S*).
undefined-lit (*get-trail-wl S*) *L* ∧ *get-conflict-wl S* = *None* ∧
C ∈# *dom-m* (*get-clauses-wl S*) ∧ *L* ∈# *all-lits-st S*)›

lemma *propagate-lit-wl-bin-heur-propagate-lit-wl-bin*:
‹(*uncurry2 propagate-lit-wl-bin-heur*, *uncurry2* (*propagate-lit-wl-bin*)) ∈
[λ-. *True*]$_f$
nat-lit-lit-rel ×$_f$ *nat-rel* ×$_f$ *twl-st-heur-up''* $\mathcal{D}$ *r s K lcount* → ⟨*twl-st-heur-up''* $\mathcal{D}$ *r s K lcount*⟩*nres-rel*›
supply [[*goals-limit=1*]]
unfolding *propagate-lit-wl-bin-heur-def propagate-lit-wl-bin-def Let-def*
apply (*intro frefI nres-relI*) **unfolding** *uncurry-def mop-save-phase-heur-def nres-monad3*
apply (*refine-rcg*)
apply (*rule-tac* $\mathcal{A}$ = ‹*all-atms-st* (*snd y*)› **in** *cons-trail-Propagated-tr2*)
subgoal by (*auto 4 3 simp*: *twl-st-heur-def propagate-lit-wl-bin-heur-def propagate-lit-wl-bin-def*
isa-vmtf-consD twl-st-heur'-def propagate-lit-wl-bin-pre-def swap-lits-pre-def
arena-lifting phase-saving-def atms-of-def save-phase-def $\mathcal{L}_{all}$*-all-atms-all-lits*
all-lits-def ac-simps
intro!: *ASSERT-refine-left cons-trail-Propagated-tr2 cons-trail-Propagated-tr-pre*
dest: *multi-member-split valid-arena-DECISION-REASON*)
subgoal by (*auto 4 3 simp*: *twl-st-heur-def twl-st-heur'-def propagate-lit-wl-bin-pre-def swap-lits-pre-def*
arena-lifting phase-saving-def atms-of-def save-phase-def all-lits-st-alt-def[*symmetric*]
intro!: *ASSERT-refine-left cons-trail-Propagated-tr2 cons-trail-Propagated-tr-pre*
dest: *multi-member-split valid-arena-DECISION-REASON*
intro!: *save-phase-heur-preI*)
subgoal by (*auto 4 3 simp*: *twl-st-heur-def twl-st-heur'-def propagate-lit-wl-bin-pre-def swap-lits-pre-def*
arena-lifting phase-saving-def atms-of-def save-phase-def all-lits-st-alt-def[*symmetric*] *ac-simps*
intro!: *ASSERT-refine-left cons-trail-Propagated-tr2 cons-trail-Propagated-tr-pre*
dest: *multi-member-split valid-arena-DECISION-REASON*)
subgoal by (*auto 4 3 simp*: *twl-st-heur-def twl-st-heur'-def propagate-lit-wl-bin-pre-def swap-lits-pre-def*
arena-lifting phase-saving-def atms-of-def save-phase-def all-lits-st-alt-def[*symmetric*]
intro!: *ASSERT-refine-left cons-trail-Propagated-tr2 cons-trail-Propagated-tr-pre*
dest: *multi-member-split valid-arena-DECISION-REASON*
intro!: *save-phase-heur-preI*)
subgoal for *x y*
by (*cases x*; *cases y*; *hypsubst*)

(*clarsimp simp add*: *ac-simps twl-st-heur-def twl-st-heur'-def isa-vmtf-consD*
op-clauses-swap-def)
done

lemma *pos-of-watched-alt*:
‹*pos-of-watched N C L* = *do* {
ASSERT(*length* (*N* ∝ *C*) > *0* ∧ *C* ∈# *dom-m N*);
let L' = (*N* ∝ *C*) ! *0*;
RETURN (*if L'* = *L then 0 else 1*)
}›
unfolding *pos-of-watched-def Let-def* **by** *auto*

lemma *pos-of-watched-heur*:
‹(*S*, *S'*) ∈ {(*T*, *T'*). *get-vdom T* = *get-vdom x2e* ∧ (*T*, *T'*) ∈ *twl-st-heur-up''* 𝒟 *r s t lcount*} ⟹
((*C*, *L*), (*C'*, *L'*)) ∈ *Id* ×$_r$ *Id* ⟹
pos-of-watched-heur S C L ≤ ⇓ *nat-rel* (*pos-of-watched* (*get-clauses-wl S'*) *C' L'*)›
unfolding *pos-of-watched-heur-def pos-of-watched-alt mop-access-lit-in-clauses-heur-def*
by (*refine-rcg mop-arena-lit*[**where** *vdom* = ‹*set* (*get-vdom S*)›])
(*auto simp*: *twl-st-heur'-def twl-st-heur-def*)

lemma *other-watched-heur*:
‹(*S*, *S'*) ∈ {(*T*, *T'*). *get-vdom T* = *get-vdom x2e* ∧ (*T*, *T'*) ∈ *twl-st-heur-up''* 𝒟 *r s t lcount*} ⟹
((*L*, *C*, *i*), (*L'*, *C'*, *i'*)) ∈ *Id* ×$_r$ *Id* ⟹
other-watched-heur S L C i ≤ ⇓ *Id* (*other-watched-wl S' L' C' i'*)›
using *arena-lifting*(*5*,*7*)[*of* ‹*get-clauses-wl-heur S*› ‹*get-clauses-wl S'*› - *C i*]
unfolding *other-watched-wl-heur-def other-watched-wl-def*
mop-access-lit-in-clauses-heur-def
by (*refine-rcg mop-arena-lit*[**where** *vdom* = ‹*set* (*get-vdom S*)›])
(*auto simp*: *twl-st-heur'-def twl-st-heur-def*
arena-lit-pre2-def
intro!: *exI*[*of* - ‹*get-clauses-wl S'*›])

14.3 Full inner loop

declare *RETURN-as-SPEC-refine*[*refine2 del*]

definition *set-conflict-wl'-pre* **where**
‹*set-conflict-wl'-pre i S* ⟷
get-conflict-wl S = *None* ∧ *i* ∈# *dom-m* (*get-clauses-wl S*) ∧
literals-are-in-$\mathcal{L}_{in}$*-mm* (*all-atms-st S*) (*mset* ‘# *ran-mf* (*get-clauses-wl S*)) ∧
¬ *tautology* (*mset* (*get-clauses-wl S* ∝ *i*)) ∧
distinct (*get-clauses-wl S* ∝ *i*) ∧
literals-are-in-$\mathcal{L}_{in}$*-trail* (*all-atms-st S*) (*get-trail-wl S*)›

lemma *literals-are-in-*$\mathcal{L}_{in}$*-mm-clauses*[*simp*]: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* (*all-atms-st S*) (*mset* ‘# *ran-mf* (*get-clauses-wl S*))›
‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* (*all-atms-st S*) ((λ*x*. *mset* (*fst x*)) ‘# *ran-m* (*get-clauses-wl S*))›
apply (*auto simp*: $\mathcal{L}_{all}$*-all-atms-all-lits literals-are-in-*$\mathcal{L}_{in}$*-mm-def all-lits-st-alt-def*[*symmetric*]
all-lits-st-def all-lits-def all-lits-of-mm-union)
done

lemma *set-conflict-wl-alt-def*:
‹*set-conflict-wl* = (λ*C S*. *do* {
ASSERT(*set-conflict-wl-pre C S*);
let (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*) = *S*;

let D = Some (mset (N ∝ C));
j ← RETURN (length M);
RETURN (M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, {#}, W)
})
unfolding *set-conflict-wl-def Let-def* **by** (*auto simp*: *ac-simps*)

lemma *set-conflict-wl-pre-set-conflict-wl'-pre*:
assumes ‹*set-conflict-wl-pre C S*›
shows ‹*set-conflict-wl'-pre C S*›
proof −
obtain *S' T b b'* **where**
SS': ‹(*S*, *S'*) ∈ *state-wl-l b*› **and**
‹*blits-in-$\mathcal{L}_{in}$ S*› **and**
confl: ‹*get-conflict-l S'* = *None*› **and**
dom: ‹*C* ∈# *dom-m* (*get-clauses-l S'*)› **and**
tauto: ‹¬ *tautology* (*mset* (*get-clauses-l S'* ∝ *C*))› **and**
dist: ‹*distinct* (*get-clauses-l S'* ∝ *C*)› **and**
‹*get-trail-l S'* ⊨*as CNot* (*mset* (*get-clauses-l S'* ∝ *C*))› **and**
T: ‹(*set-clauses-to-update-l* (*clauses-to-update-l S'* + {#*C*#}) *S'*, *T*)
∈ *twl-st-l b'*› **and**
struct: ‹*twl-struct-invs T*› **and**
‹*twl-stgy-invs T*›
using *assms*
unfolding *set-conflict-wl-pre-def set-conflict-l-pre-def* **apply** −
by *blast*
have *lits-trail*: ‹*lits-of-l* (*get-trail T*) ⊆ *set-mset* (*all-lits-of-st T*)›
using *twl-struct-invs-no-alien-in-trail*[*OF struct*] **by** *auto*
then have ‹*literals-are-in-$\mathcal{L}_{in}$-trail* (*all-atms-st S*) (*get-trail-wl S*)›
using *T SS'*
by (*auto simp*: *literals-are-in-$\mathcal{L}_{in}$-trail-atm-of lits-of-def all-lits-st-alt-def*)
then show *?thesis*
using *SS' T dom tauto dist confl* **unfolding** *set-conflict-wl'-pre-def*
by (*auto simp*: *literals-are-in-$\mathcal{L}_{in}$-def twl-st-l*
mset-take-mset-drop-mset' simp del: *all-atms-def*[*symmetric*])
qed

lemma *set-conflict-wl-heur-set-conflict-wl'*:
‹(*uncurry set-conflict-wl-heur*, *uncurry* (*set-conflict-wl*)) ∈
[λ-. *True*]$_f$
nat-rel ×$_r$ *twl-st-heur-up'' $\mathcal{D}$ r s K lcount* → ⟨*twl-st-heur-up'' $\mathcal{D}$ r s K lcount*⟩*nres-rel*›
proof −
have *H*:
‹*isa-set-lookup-conflict-aa x y z a b d*
≤ ⇓ (*option-lookup-clause-rel $\mathcal{A}$* ×$_f$ (*nat-rel* ×$_f$ *Id*))
(*set-conflict-m x' y' z' a' b' d'*)›
if
‹(((((((*x*, *y*), *z*), *a*), *b*)), *d*), (((((*x'*, *y'*), *z'*), *a'*), *b'*)), *d'*)
∈ *trail-pol $\mathcal{A}$* ×$_f$ {(*arena*, *N*). *valid-arena arena N vdom*} ×$_f$
nat-rel ×$_f$
option-lookup-clause-rel $\mathcal{A}$ ×$_f$
nat-rel ×$_f$ *Id*› **and**
‹*z'* ∈# *dom-m y'* ∧ *a'* = *None* ∧ *distinct* (*y'* ∝ *z'*) ∧
literals-are-in-$\mathcal{L}_{in}$-mm $\mathcal{A}$ (*mset* ‘# *ran-mf y'*) ∧
¬ *tautology* (*mset* (*y'* ∝ *z'*)) ∧ *b'* = *0* ∧ *out-learned x' None d'* ∧
isasat-input-bounded $\mathcal{A}$›
for *x x' y y' z z' a a' b b' c c' d d' vdom $\mathcal{A}$*

by (*rule isa-set-lookup-conflict*[*THEN fref-to-Down-curry5*,
unfolded prod.case, *OF that*(*2*,*1*)])

have [*refine0*]: ‹*isa-set-lookup-conflict-aa* (*get-trail-wl-heur x2g*) (*get-clauses-wl-heur x2g*) *x1g*
(*get-conflict-wl-heur x2g*) *0* (*get-outlearned-heur x2g*)
≤ ⇓ {((*C*, *n*, *outl*), *D*). (*C*, *D*) ∈ *option-lookup-clause-rel* (*all-atms-st x2*) ∧
n = *card-max-lvl x1a* (*the D*) ∧ *out-learned x1a D outl*}
(*RETURN* (*Some* (*mset* (*x1b* ∝ *x1*))))›
if
‹(*x*, *y*) ∈ *nat-rel* ×*f* *twl-st-heur-up''* 𝒟 *r s K lcount*› **and**
‹*x2i'* = (*x1j'*, *x2j'*)› **and**
‹*x2h'* = (*x1i'*, *x2i'*)› **and**
‹*x2g'* = (*x1h'*, *x2h'*)› **and**
‹*x2f* = (*x1g'*, *x2g'*)› **and**
‹*x2e* = (*x1f*, *x2f*)› **and**
‹*x2d* = (*x1e*, *x2e*)› **and**
‹*x2c* = (*x1d*, *x2d*)› **and**
‹*x2b* = (*x1c*, *x2c*)› **and**
‹*x2a* = (*x1b*, *x2b*)› **and**
‹*x2* = (*x1a*, *x2a*)› **and**
‹*y* = (*x1*, *x2*)› **and**
‹*x* = (*x1g*, *S*)› **and**
‹(*x2g*, *x2*) ∈ *twl-st-heur-up''* 𝒟 *r s K lcount*›
‹*case y of* (*x*, *xa*) ⇒ *set-conflict-wl'-pre x xa*›
for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h*
x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q x2q
x1r x2r x1s x2s x1t x2t x1g' x2g' x1h' x2h' x1i' x2i' x1j' x2j' S
proof −
have [*iff*]: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm*
(*all-atms-st* ([], *x1b*, *None*, *x1d*, *x1e*, *x1f*, *xaa*, *af*, *ag*, *ah*, *ai*, {#}, *bb*))
{#*mset* (*fst x*). *x* ∈# *ran-m x1b*#}› **for** *xaa af ag ah ai bb*
by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-mm-def in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *all-atms-st-def*
in-all-lits-of-mm-ain-atms-of-iff all-atms-def all-lits-def
simp del: *all-atms-def*[*symmetric*])

show *?thesis*
apply (*rule order-trans*)
apply (*rule H*[*of - - - - - - x1a x1b x1g x1c 0* ‹*get-outlearned-heur x2g*› ‹*all-atms-st x2*›
‹*set* (*get-vdom* (*x2g*))›])
subgoal
using *that*
by (*auto simp*: *twl-st-heur'-def twl-st-heur-def get-clauses-wl.simps ac-simps*)
subgoal
using *that* **apply** *auto*
by (*auto 0 0 simp add*: *RETURN-def conc-fun-RES set-conflict-m-def twl-st-heur'-def*
twl-st-heur-def set-conflict-wl'-pre-def ac-simps)
subgoal
using *that*
by (*auto 0 0 simp add*: *RETURN-def conc-fun-RES set-conflict-m-def twl-st-heur'-def*
twl-st-heur-def)
done
qed

have *isa-set-lookup-conflict-aa-pre*:
‹(*x*, *y*) ∈ *nat-rel* ×*f* *twl-st-heur-up''* 𝒟 *r s K lcount* ⟹
y = (*x1*, *x2*) ⟹

```
    x = (x1a, x2a) ⟹
    set-conflict-wl'-pre x1 x2 ⟹
    (S, x2) ∈ twl-st-heur-up'' 𝒟 r s K lcount ⟹
    curry2 (curry2 (curry isa-set-lookup-conflict-aa-pre)) (get-trail-wl-heur S)
    (get-clauses-wl-heur S) x1a (get-conflict-wl-heur S) 0 (get-outlearned-heur S)›
    for x1 x2 x1a x2a S y x
    unfolding isa-set-lookup-conflict-aa-pre-def set-conflict-wl'-pre-def
      twl-st-heur'-def twl-st-heur-def
    by (auto simp: arena-lifting)


  have set-conflict-wl-heur-alt-def: ‹set-conflict-wl-heur = (λC S. do {
    let S = S;
    let n = 0;
    let M = get-trail-wl-heur S;
    let N = get-clauses-wl-heur S;
    let D = get-conflict-wl-heur S;
    let outl = get-outlearned-heur S;
    ASSERT(curry5 isa-set-lookup-conflict-aa-pre M N C D n outl);
    (D, clvls, outl) ← isa-set-lookup-conflict-aa M N C D n outl;
    j ← mop-isa-length-trail M;
    let S = IsaSAT-Setup.set-conflict-wl-heur D S;
    let S = set-outl-wl-heur outl S;
    let S = set-count-max-wl-heur clvls S;
    let S = set-literals-to-update-wl-heur j S;
      RETURN S})›
    unfolding set-conflict-wl-heur-def by auto

  show ?thesis
    supply [[goals-limit=1]]
    apply (intro nres-relI frefI)
    subgoal for x y
    unfolding uncurry-def RES-RETURN-RES4 set-conflict-wl-alt-def  set-conflict-wl-heur-alt-def
    apply (rewrite at ‹let - = 0 in -› Let-def)
    apply (rewrite at ‹let - = get-trail-wl-heur - in -› Let-def)
    apply (rewrite at ‹let - = get-clauses-wl-heur - in -› Let-def)
    apply (rewrite at ‹let - = get-conflict-wl-heur - in -› Let-def)
    apply (rewrite at ‹let - = get-outlearned-heur - in -› Let-def)
    apply (refine-vcg mop-isa-length-trail-length-u[of ‹all-atms-st (snd y)›, THEN fref-to-Down-Id-keep,
unfolded length-uint32-nat-def
     comp-def] mark-conflict-to-rescore[where 𝒟 = 𝒟 and r=r and s=s and K=K and lcount=lcount,
unfolded conc-fun-RETURN[symmetric]])
    subgoal for x1 x2 x1a x2a
      by (rule isa-set-lookup-conflict-aa-pre) (auto dest!: set-conflict-wl-pre-set-conflict-wl'-pre)
    apply assumption+
    subgoal by auto
    subgoal by (auto dest!: set-conflict-wl-pre-set-conflict-wl'-pre)
    subgoal for x y
      unfolding arena-is-valid-clause-idx-def
      by (auto simp: twl-st-heur'-def twl-st-heur-def)
    subgoal
      by (auto simp: twl-st-heur'-def twl-st-heur-def counts-maximum-level-def ac-simps
        set-conflict-wl'-pre-def dest!: set-conflict-wl-pre-set-conflict-wl'-pre
 intro!: valid-arena-mark-used)
    done
    done
```

qed

lemma *in-Id-in-Id-option-rel*[*refine*]:
‹(*f*, *f'*) ∈ *Id* ⟹ (*f*, *f'*) ∈ ⟨*Id*⟩ *option-rel*›
by *auto*

The assumption that that accessed clause is active has not been checked at this point!

definition *keep-watch-heur-pre* **where**
‹*keep-watch-heur-pre* =
(λ(((*L*, *j*), *w*), *S*).
L ∈# $\mathcal{L}_{all}$ (*all-atms-st S*))›

lemma *vdom-m-update-subset'*:
‹*fst C* ∈ *vdom-m* $\mathcal{A}$ *bh N* ⟹ *vdom-m* $\mathcal{A}$ (*bh*(*ap* := (*bh ap*)[*bf* := *C*])) *N* ⊆ *vdom-m* $\mathcal{A}$ *bh N*›
unfolding *vdom-m-def*
by (*fastforce split*: *if-splits elim*!: *in-set-upd-cases*)

lemma *vdom-m-update-subset*:
‹*bg* < *length* (*bh ap*) ⟹ *vdom-m* $\mathcal{A}$ (*bh*(*ap* := (*bh ap*)[*bf* := *bh ap* ! *bg*])) *N* ⊆ *vdom-m* $\mathcal{A}$ *bh N*›
unfolding *vdom-m-def*
by (*fastforce split*: *if-splits elim*!: *in-set-upd-cases*)

lemma *keep-watch-heur-keep-watch*:
‹(*uncurry3 keep-watch-heur*, *uncurry3* (*mop-keep-watch*)) ∈
[λ-. *True*]$_f$
Id ×$_f$ *nat-rel* ×$_f$ *nat-rel* ×$_f$ *twl-st-heur-up''* $\mathcal{D}$ *r s K lcount* → ⟨*twl-st-heur-up''* $\mathcal{D}$ *r s K lcount*⟩
nres-rel›
unfolding *keep-watch-heur-def mop-keep-watch-def uncurry-def*
$\mathcal{L}_{all}$-*all-atms-all-lits*[*symmetric*]
apply (*intro frefI nres-relI*)
apply *refine-rcg*
subgoal
by (*auto 5 4 simp*: *keep-watch-heur-def keep-watch-def twl-st-heur'-def keep-watch-heur-pre-def*
twl-st-heur-def map-fun-rel-def2 all-atms-def[*symmetric*] *mop-keep-watch-def*
intro!: *ASSERT-leI*
dest: *vdom-m-update-subset*)
subgoal
by (*auto 5 4 simp*: *keep-watch-heur-def keep-watch-def twl-st-heur'-def keep-watch-heur-pre-def*
twl-st-heur-def map-fun-rel-def2 all-atms-def[*symmetric*] *mop-keep-watch-def watched-by-alt-def*
intro!: *ASSERT-leI*
dest: *vdom-m-update-subset*)
subgoal
by (*auto 5 4 simp*: *keep-watch-heur-def keep-watch-def twl-st-heur'-def keep-watch-heur-pre-def*
twl-st-heur-def map-fun-rel-def2 all-atms-def[*symmetric*] *mop-keep-watch-def watched-by-alt-def*
intro!: *ASSERT-leI*
dest: *vdom-m-update-subset*)
subgoal
by (*clarsimp simp*: *keep-watch-heur-def keep-watch-def twl-st-heur'-def keep-watch-heur-pre-def*
twl-st-heur-def map-fun-rel-def2 all-atms-def[*symmetric*] *mop-keep-watch-def watched-by-alt-def*
intro!: *ASSERT-leI split*: *prod.splits*)
(*auto dest*!: *vdom-m-update-subset*)
done

This is a slightly stronger version of the previous lemma:

lemma *keep-watch-heur-keep-watch'*:

‹((((L′, j′), w′), S′), ((L, j), w), S)
∈ nat-lit-lit-rel ×$_f$ nat-rel ×$_f$ nat-rel ×$_f$ twl-st-heur-up′′ $\mathcal{D}$ r s K lcount ⟹
keep-watch-heur L′ j′ w′ S′ ≤ ⇓ {(T, T′). get-vdom T = get-vdom S′ ∧
(T, T′) ∈ twl-st-heur-up′′ $\mathcal{D}$ r s K lcount}
(mop-keep-watch L j w S)›
unfolding keep-watch-heur-def mop-keep-watch-def uncurry-def
$\mathcal{L}_{all}$-all-atms-all-lits[symmetric]
apply refine-rcg
subgoal
by (auto 5 4 simp: keep-watch-heur-def keep-watch-def twl-st-heur′-def keep-watch-heur-pre-def
twl-st-heur-def map-fun-rel-def2 all-atms-def[symmetric] mop-keep-watch-def
intro!: ASSERT-leI
dest: vdom-m-update-subset)
subgoal
by (auto 5 4 simp: keep-watch-heur-def keep-watch-def twl-st-heur′-def keep-watch-heur-pre-def
twl-st-heur-def map-fun-rel-def2 all-atms-def[symmetric] mop-keep-watch-def watched-by-alt-def
intro!: ASSERT-leI
dest: vdom-m-update-subset)
subgoal
by (auto 5 4 simp: keep-watch-heur-def keep-watch-def twl-st-heur′-def keep-watch-heur-pre-def
twl-st-heur-def map-fun-rel-def2 all-atms-def[symmetric] mop-keep-watch-def watched-by-alt-def
intro!: ASSERT-leI
dest: vdom-m-update-subset)
subgoal
by (clarsimp simp: keep-watch-heur-def keep-watch-def twl-st-heur′-def keep-watch-heur-pre-def
twl-st-heur-def map-fun-rel-def2 all-atms-def[symmetric] mop-keep-watch-def watched-by-alt-def
intro!: ASSERT-leI split: prod.splits
dest: vdom-m-update-subset)
(auto dest!: vdom-m-update-subset)
done

definition update-blit-wl-heur-pre **where**
‹update-blit-wl-heur-pre r K′ = (λ((((((L, C), b), j), w), K), S). L = K′)›

lemma update-blit-wl-heur-update-blit-wl:
‹(uncurry6 update-blit-wl-heur, uncurry6 update-blit-wl) ∈
[update-blit-wl-heur-pre r K]$_f$
nat-lit-lit-rel ×$_f$ nat-rel ×$_f$ bool-rel ×$_f$ nat-rel ×$_f$ nat-rel ×$_f$ Id ×$_f$
twl-st-heur-up′′ $\mathcal{D}$ r s K lcount →
⟨nat-rel ×$_r$ nat-rel ×$_r$ twl-st-heur-up′′ $\mathcal{D}$ r s K lcount⟩ nres-rel›
apply (intro frefI nres-relI) — TODO proof
apply (auto simp: update-blit-wl-heur-def update-blit-wl-def twl-st-heur′-def keep-watch-heur-pre-def
twl-st-heur-def map-fun-rel-def2 update-blit-wl-heur-pre-def all-atms-def[symmetric]
$\mathcal{L}_{all}$-all-atms-all-lits
simp flip: all-lits-alt-def2
intro!: ASSERT-leI ASSERT-refine-right
simp: vdom-m-update-subset)
subgoal for aa ab ac ad ae be af ag ah bf aj ak al am an bg ao bh ap
apply (subgoal-tac ‹vdom-m (all-atms-st ([], ag, None, ah, bf, aj, ak, al, am, an, bg, {#}, ao)) (ao(K := (ao K)[ac := (aa, ae, ab)])) ag ⊆
vdom-m (all-atms-st ([], ag, None, ah, bf, aj, ak, al, am, an, bg, {#}, ao)) ao ag›)
apply fast
apply (rule vdom-m-update-subset′)
apply auto
done
subgoal for aa ab ac ad ae be af ag ah bf aj ak al am an bg ao bh ap

apply (*subgoal-tac* ‹*vdom-m* (*all-atms-st* ([], *ag*, *None*, *bf*, *aj*, *ak*, *al*, *am*, *an*, *bg*, *ao*, {#}, *bh*)) (*bh*(*K* := (*bh K*)[*ac* := (*aa*, *ae*, *ab*)])) *ag* ⊆
vdom-m (*all-atms-st* ([], *ag*, *None*, *bf*, *aj*, *ak*, *al*, *am*, *an*, *bg*, *ao*, {#}, *bh*)) *bh ag* ›)
apply *fast*
apply (*rule vdom-m-update-subset′*)
apply *auto*
done
done

lemma *mop-access-lit-in-clauses-heur*:
‹(*S*, *T*) ∈ *twl-st-heur* ⟹ (*i*, *i′*) ∈ *Id* ⟹ (*j*, *j′*) ∈ *Id* ⟹ *mop-access-lit-in-clauses-heur S i j*
≤ ⇓ *Id*
(*mop-clauses-at* (*get-clauses-wl T*) *i′ j′*)›
unfolding *mop-access-lit-in-clauses-heur-def*
by (*rule mop-arena-lit2*[**where** *vdom*=‹*set* (*get-vdom S*)›])
(*auto simp*: *twl-st-heur-def intro*!: *mop-arena-lit2*)

lemma *isa-find-unwatched-wl-st-heur-find-unwatched-wl-st*:
‹*isa-find-unwatched-wl-st-heur x′ y′*
≤ ⇓ *Id* (*find-unwatched-l* (*get-trail-wl x*) (*get-clauses-wl x*) *y*)›
if
xy: ‹((*x′*, *y′*), *x*, *y*) ∈ *twl-st-heur* ×*f* *nat-rel*›
for *x y x′ y′*
proof −
have *find-unwatched-l-alt-def*: ‹*find-unwatched-l M N C* = *do* {
ASSERT(*C* ∈# *dom-m N* ∧ *length* (*N* ∝ *C*) ≥ *2* ∧ *distinct* (*N* ∝ *C*) ∧ *no-dup M*);
find-unwatched-l M N C
}› **for** *M N C*
unfolding *find-unwatched-l-def* **by** (*auto simp*: *summarize-ASSERT-conv*)
have *K*: ‹*find-unwatched-wl-st′ x y* ≤ *find-unwatched-l* (*get-trail-wl x*) (*get-clauses-wl x*) *y*›
unfolding *find-unwatched-wl-st′-def*
apply (*subst find-unwatched-l-alt-def*)
unfolding *le-ASSERT-iff*
apply (*cases x*)
apply *clarify*
apply (*rule order-trans*)
apply (*rule find-unwatched*[*of* - - - ‹*all-atms-st x*›])
subgoal
by *simp*
subgoal
by *auto*
subgoal
using *literals-are-in-*$\mathcal{L}_{in}$*-nth2*[*of y x*]
by *simp*
subgoal by *auto*
done
show *?thesis*
apply (*subst find-unwatched-l-alt-def*)
apply (*intro ASSERT-refine-right*)
apply (*rule order-trans*)
apply (*rule find-unwatched-wl-st-heur-find-unwatched-wl-s*[*THEN fref-to-Down-curry*,
OF - *that*(*1*)])
by (*simp-all add*: *K find-unwatched-wl-st-pre-def literals-are-in-*$\mathcal{L}_{in}$*-nth2*)
qed

```
lemma unit-propagation-inner-loop-body-wl-alt-def:
  ‹unit-propagation-inner-loop-body-wl L j w S = do {
      ASSERT(unit-propagation-inner-loop-wl-loop-pre L (j, w, S));
      (C, K, b) ← mop-watched-by-at S L w;
      S ← mop-keep-watch L j w S;
      ASSERT(is-nondeleted-clause-pre C L S);
      val-K ← mop-polarity-wl S K;
      if val-K = Some True
      then RETURN (j+1, w+1, S)
      else do {
        if b then do {
          ASSERT(propagate-proper-bin-case L K S C);
          if val-K = Some False
          then do {S ← set-conflict-wl C S;
            RETURN (j+1, w+1, S)}
          else do {
            S ← propagate-lit-wl-bin K C S;
            RETURN (j+1, w+1, S)}
        } — Now the costly operations:
        else if C ∉# dom-m (get-clauses-wl S)
        then RETURN (j, w+1, S)
        else do {
          ASSERT(unit-prop-body-wl-inv S j w L);
          i ← pos-of-watched (get-clauses-wl S) C L;
          ASSERT(i ≤ 1);
          L' ← other-watched-wl S L C i;
          val-L' ← mop-polarity-wl S L';
          if val-L' = Some True
          then update-blit-wl L C b j w L' S
          else do {
            f ← find-unwatched-l (get-trail-wl S) (get-clauses-wl S) C;
            ASSERT (unit-prop-body-wl-find-unwatched-inv f C S);
            case f of
              None ⇒ do {
                if val-L' = Some False
                then do {S ← set-conflict-wl C S;
                   RETURN (j+1, w+1, S)}
                else do {S ← propagate-lit-wl L' C i S; RETURN (j+1, w+1, S)}
              }
            | Some f ⇒ do {
                ASSERT(C ∈# dom-m (get-clauses-wl S) ∧ f < length (get-clauses-wl S ∝ C) ∧ f ≥ 2);
                let S = S; — position saving
                K ← mop-clauses-at (get-clauses-wl S) C f;
                val-L' ← mop-polarity-wl S K;
                if val-L' = Some True
                then update-blit-wl L C b j w K S
                else update-clause-wl L L' C b j w i f S
              }
          }
        }
      }
    }›
  unfolding unit-propagation-inner-loop-body-wl-def Let-def by auto

lemma fref-to-Down-curry8:
  ‹(uncurry8 ff, uncurry8 g) ∈ [P]_f A → ⟨B⟩nres-rel ⟹
```

$(\bigwedge x\ x'\ y\ y'\ z\ z'\ a\ a'\ b\ b'\ c\ c'\ d\ d'\ e\ e'\ f\ f'.\ P\ ((((((((x',\ y'),\ z'),\ a'),\ b'),\ c'),\ d'),\ e'),\ f') \Longrightarrow$
$(((((((((x,\ y),\ z),\ a),\ b),\ c),\ d),\ e),\ f),\ ((((((((x',\ y'),\ z'),\ a'),\ b'),\ c'),\ d'),\ e'),\ f')) \in A \Longrightarrow$
$ff\ x\ y\ z\ a\ b\ c\ d\ e\ f \leq \Downarrow B\ (g\ x'\ y'\ z'\ a'\ b'\ c'\ d'\ e'\ f'))\rangle$
unfolding *fref-def uncurry-def nres-rel-def* **by** *auto*

lemma *unit-propagation-inner-loop-body-wl-heur-unit-propagation-inner-loop-body-wl-D*:
$\langle$(*uncurry3 unit-propagation-inner-loop-body-wl-heur*,
uncurry3 unit-propagation-inner-loop-body-wl)
$\in [\lambda(((L,\ i),\ j),\ S).\ length\ (watched\text{-}by\ S\ L) \leq r - MIN\text{-}HEADER\text{-}SIZE \wedge L = K \wedge$
$length\ (watched\text{-}by\ S\ L) = s]_f$
$nat\text{-}lit\text{-}lit\text{-}rel \times_f nat\text{-}rel \times_f nat\text{-}rel \times_f twl\text{-}st\text{-}heur\text{-}up''\ \mathcal{D}\ r\ s\ K\ lcount \rightarrow$
$\langle nat\text{-}rel \times_r nat\text{-}rel \times_r twl\text{-}st\text{-}heur\text{-}up''\ \mathcal{D}\ r\ s\ K\ lcount\rangle nres\text{-}rel\rangle$
proof −

have [*refine*]: $\langle clause\text{-}not\text{-}marked\text{-}to\text{-}delete\text{-}heur\text{-}pre\ (S',\ C')\rangle$
if $\langle is\text{-}nondeleted\text{-}clause\text{-}pre\ C\ L\ S\rangle$ **and** $\langle((C',\ S'),\ (C,\ S)) \in nat\text{-}rel \times_r twl\text{-}st\text{-}heur\rangle$ **for** *C C' S S' L*
using *that* **apply** −
unfolding *clause-not-marked-to-delete-heur-pre-def prod.case arena-is-valid-clause-vdom-def*
by (*rule exI*[*of* - $\langle get\text{-}clauses\text{-}wl\ S\rangle$], *rule exI*[*of* - $\langle set\ (get\text{-}vdom\ S')\rangle$])
(*use that* **in** $\langle$*auto 5 3 simp*: *is-nondeleted-clause-pre-def twl-st-heur-def vdom-m-def watched-by-alt-def*
simp flip: *all-lits-st-alt-def dest*!: *multi-member-split*[*of L*]$\rangle$)

note [*refine*] = *mop-watched-by-app-heur-mop-watched-by-at''*[*of* $\mathcal{D}$ *r lcount K s*, *THEN fref-to-Down-curry2*]
keep-watch-heur-keep-watch'[*of* - - - - - - - - $\mathcal{D}$ *r lcount K s*]
mop-polarity-st-heur-mop-polarity-wl''[*of* $\mathcal{D}$ *r lcount K s*, *THEN fref-to-Down-curry*, *unfolded comp-def*]
set-conflict-wl-heur-set-conflict-wl'[*of* $\mathcal{D}$ *r lcount K s*, *THEN fref-to-Down-curry*, *unfolded comp-def*]
propagate-lit-wl-bin-heur-propagate-lit-wl-bin
[*of* $\mathcal{D}$ *r lcount K s*, *THEN fref-to-Down-curry2*, *unfolded comp-def*]
pos-of-watched-heur[*of* - - - $\mathcal{D}$ *r lcount K s*]
mop-access-lit-in-clauses-heur
update-blit-wl-heur-update-blit-wl[*of r K* $\mathcal{D}$ *lcount s*, *THEN fref-to-Down-curry6*]
isa-find-unwatched-wl-st-heur-find-unwatched-wl-st
propagate-lit-wl-heur-propagate-lit-wl[*of* $\mathcal{D}$ *r lcount K s*, *THEN fref-to-Down-curry3*, *unfolded comp-def*]
isa-save-pos-is-Id
update-clause-wl-heur-update-clause-wl[*of K r* $\mathcal{D}$ *lcount s*, *THEN fref-to-Down-curry8*]
other-watched-heur[*of* - - - $\mathcal{D}$ *r lcount K s*]

have [*simp*]: $\langle is\text{-}nondeleted\text{-}clause\text{-}pre\ x1f\ x1b\ Sa \Longrightarrow$
$clause\text{-}not\text{-}marked\text{-}to\text{-}delete\text{-}pre\ (Sa,\ x1f)\rangle$ **for** *x1f x1b Sa*
unfolding *is-nondeleted-clause-pre-def clause-not-marked-to-delete-pre-def vdom-m-def*
all-lits-st-alt-def[*symmetric*] **by** (*cases Sa*; *auto dest*!: *multi-member-split*)

show *?thesis*
supply [[*goals-limit=1*]] *twl-st-heur'-def*[*simp*]
supply *RETURN-as-SPEC-refine*[*refine2 del*]
apply (*intro frefI nres-relI*)
unfolding *unit-propagation-inner-loop-body-wl-heur-def*
unit-propagation-inner-loop-body-wl-alt-def
uncurry-def clause-not-marked-to-delete-def[*symmetric*]
watched-by-app-heur-def access-lit-in-clauses-heur-def

apply (*refine-rcg*)

subgoal unfolding *unit-propagation-inner-loop-wl-loop-D-heur-inv0-def twl-st-heur'-def*
unit-propagation-inner-loop-wl-loop-pre-def
by *fastforce*
subgoal by *fast*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *fast*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *fast*
subgoal by *simp*
subgoal by *simp*
subgoal by *fast*
subgoal by *simp*
subgoal by *simp*
apply *assumption*
subgoal by *auto*
subgoal
unfolding *Not-eq-iff*
by (*rule clause-not-marked-to-delete-rel*[*THEN fref-to-Down-unRET-Id-uncurry*])
(*simp-all add*: *clause-not-marked-to-delete-rel*[*THEN fref-to-Down-unRET-Id-uncurry*])
subgoal by *auto*
apply *assumption*
subgoal by *auto*
subgoal by *auto*
apply *assumption*
subgoal by *auto*
subgoal by *fast*
subgoal by *simp*
subgoal by *simp*
subgoal
unfolding *update-blit-wl-heur-pre-def unit-propagation-inner-loop-wl-loop-D-heur-inv0-def*
prod.case unit-propagation-inner-loop-wl-loop-pre-def
by *normalize-goal+ simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *force*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by (*simp add*: *clause-not-marked-to-delete-def*)
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*

subgoal by *simp*
subgoal by (*simp add*: *update-blit-wl-heur-pre-def*)
subgoal by *simp*
subgoal by (*simp add*: *update-clause-wl-pre-def*)
subgoal by *simp*
done
qed

lemma *unit-propagation-inner-loop-wl-loop-D-heur-unit-propagation-inner-loop-wl-loop-D*:
‹(*uncurry unit-propagation-inner-loop-wl-loop-D-heur*,
uncurry unit-propagation-inner-loop-wl-loop)
∈ [λ(*L*, *S*). *length* (*watched-by S L*) ≤ *r* − *MIN-HEADER-SIZE* ∧ *L* = *K* ∧ *length* (*watched-by S L*)
= *s* ∧
length (*watched-by S L*) ≤ *r*]$_f$
nat-lit-lit-rel ×$_f$ *twl-st-heur-up''* 𝒟 *r s K lcount* →
⟨*nat-rel* ×$_r$ *nat-rel* ×$_r$ *twl-st-heur-up''* 𝒟 *r s K lcount*⟩*nres-rel*›
proof −
have *unit-propagation-inner-loop-wl-loop-D-heur-inv*:
‹*unit-propagation-inner-loop-wl-loop-D-heur-inv x2a x1a xa*›
if
‹(*x*, *y*) ∈ *nat-lit-lit-rel* ×$_f$ *twl-st-heur-up''* 𝒟 *r s K lcount*› **and**
‹*y* = (*x1*, *x2*)› **and**
‹*x* = (*x1a*, *x2a*)› **and**
‹(*xa*, *x'*) ∈ *nat-rel* ×$_r$ *nat-rel* ×$_r$ *twl-st-heur-up''* 𝒟 *r s K lcount*› **and**
H: ‹*unit-propagation-inner-loop-wl-loop-inv x1 x'*›
for *x y x1 x2 x1a x2a xa x'*
proof −
obtain *w S w' S' j j'* **where**
xa: ‹*xa* = (*j*, *w*, *S*)› **and** *x'*: ‹*x'* = (*j'*, *w'*, *S'*)›
by (*cases xa*; *cases x'*) *auto*
show *?thesis*
unfolding *xa unit-propagation-inner-loop-wl-loop-D-heur-inv-def prod.case*
apply (*rule exI*[*of* - *x2*])
apply (*rule exI*[*of* - *S'*])
using *that xa x' that* **apply** −
unfolding *prod.case* **apply** *hypsubst*
apply (*auto simp*: *twl-st-heur'-def*
dest!: *twl-struct-invs-no-alien-in-trail*[*of* - ‹−*x1*›] *simp flip*: *all-lits-st-alt-def*)
unfolding *unit-propagation-inner-loop-wl-loop-inv-def unit-propagation-inner-loop-l-inv-def*
unfolding *prod.case* **apply** *normalize-goal+*
apply (*drule twl-struct-invs-no-alien-in-trail*[*of* - ‹−*x1*›])
apply (*simp-all only*: *twl-st-l multiset.map-comp comp-def*
clause-twl-clause-of twl-st-wl in-all-lits-of-mm-uminus-iff ac-simps in-all-lits-uminus-iff
flip: *all-lits-st-alt-def*)
done
qed
have *length*: ‹⋀*x y x1 x2 x1a x2a*.
case y of
(*L*, *S*) ⇒
length (*watched-by S L*) ≤ *r* − *MIN-HEADER-SIZE* ∧
L = *K* ∧ *length* (*watched-by S L*) = *s* ∧ *length* (*watched-by S L*) ≤ *r* ⟹
(*x*, *y*) ∈ *nat-lit-lit-rel* ×$_f$ *twl-st-heur-up''* 𝒟 *r s K lcount* ⟹ *y* = (*x1*, *x2*) ⟹
x = (*x1a*, *x2a*) ⟹
x1 ∈# *all-lits-st x2* ⟹
length (*watched-by-int x2a x1a*) ≤ *length* (*get-clauses-wl-heur x2a*) ⟹

mop-length-watched-by-int x2a x1a
≤ ⇓ Id (RETURN (length (watched-by x2 x1)))›
unfolding *mop-length-watched-by-int-def*
by *refine-rcg*
(*auto simp*: *twl-st-heur'-def map-fun-rel-def2 twl-st-heur-def*
simp flip: $\mathcal{L}_{all}$*-all-atms-all-lits intro*!: *ASSERT-leI*)

note *H*[*refine*] = *unit-propagation-inner-loop-body-wl-heur-unit-propagation-inner-loop-body-wl-D*
[*THEN fref-to-Down-curry3*] *init*
show *?thesis*
unfolding *unit-propagation-inner-loop-wl-loop-D-heur-def*
unit-propagation-inner-loop-wl-loop-def uncurry-def
unit-propagation-inner-loop-wl-loop-inv-def[*symmetric*]
apply (*intro frefI nres-relI*)
apply (*refine-vcg*)
subgoal by (*auto simp*: *twl-st-heur'-def twl-st-heur-state-simp-watched watched-by-alt-def*
simp flip: *all-lits-st-alt-def*)
apply (*rule length*; *assumption*)
subgoal by *auto*
subgoal by (*rule unit-propagation-inner-loop-wl-loop-D-heur-inv*)
subgoal
by (*subst get-conflict-wl-is-None-heur-get-conflict-wl-is-None*[*THEN fref-to-Down-unRET-Id*])
(*auto simp*: *get-conflict-wl-is-None-heur-get-conflict-wl-is-None twl-st-heur-state-simp-watched*
twl-st-heur'-def
get-conflict-wl-is-None-def simp flip: $\mathcal{L}_{all}$*-all-atms-all-lits*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

lemma *cut-watch-list-heur2-cut-watch-list-heur*:
shows
‹*cut-watch-list-heur2 j w L S ≤ ⇓ Id (cut-watch-list-heur j w L S)*›
proof −

let *?N* = ‹*get-clauses-wl-heur S*›
let *?W* = ‹*get-watched-wl-heur S*›
define *n* **where** *n*: ‹*n* = *length* (*?W* ! *nat-of-lit L*)›
let *?R* = ‹*measure* (λ(*j'*::*nat*, *w'* :: *nat*, - :: (*nat* × *nat literal* × *bool*) *list list*). *length* (*?W*!*nat-of-lit L*) − *w'*)›
define *I'* **where**
‹*I'* ≡ λ(*j'*, *w'*, *W'*). *length* (*W'* ! (*nat-of-lit L*)) = *length* (*?W* ! (*nat-of-lit L*)) ∧ *j'* ≤ *w'* ∧ *w'* ≥ *w* ∧
w' − *w* = *j'* − *j* ∧ *j'* ≥ *j* ∧
drop w' (*W'* ! (*nat-of-lit L*)) = *drop w'* (*?W* ! (*nat-of-lit L*)) ∧
w' ≤ *length* (*W'* ! (*nat-of-lit L*)) ∧
W'[*nat-of-lit L* := *take* (*j* + *w'* − *w*) (*W'* ! *nat-of-lit L*)] =
?W[*nat-of-lit L* := *take* (*j* + *w'* − *w*) ((*take j* (*?W*!(*nat-of-lit L*)) @ *drop w* (*?W*!(*nat-of-lit L*))))]›

have *cut-watch-list-heur-alt-def*:
‹*cut-watch-list-heur j w L* =(λ*S*. *do* {
ASSERT(*j* ≤ *length* (*get-watched-wl-heur S*!*nat-of-lit L*) ∧ *j* ≤ *w* ∧ *nat-of-lit L* < *length* (*get-watched-wl-heur S*) ∧
w ≤ *length* (*get-watched-wl-heur S* ! (*nat-of-lit L*)));

```
    let W = (get-watched-wl-heur S)[nat-of-lit L := take j (get-watched-wl-heur S!(nat-of-lit L)) @ drop
w (get-watched-wl-heur S!(nat-of-lit L))];
    RETURN (set-watched-wl-heur W S)
   })›
   unfolding cut-watch-list-heur-def Let-def by auto
  have REC: ‹ASSERT (x1k < length (x2k ! nat-of-lit L)) ≫=
    (λ-. RETURN (x1j + 1, x1k + 1, x2k [nat-of-lit L := (x2k ! nat-of-lit L) [x1j :=
                x2k ! nat-of-lit L !
                x1k]]))
    ≤ SPEC (λs'. ∀ x1 x2 x1a x2a. x2 = (x1a, x2a) ⟶ s' = (x1, x2) ⟶
       (x1 ≤ x1a ∧ nat-of-lit L < length x2a) ∧ I' s' ∧
       (s', s) ∈ measure (λ(j', w', -). length (?W ! nat-of-lit L) − w'))›
   if
    ‹j ≤ length (?W ! nat-of-lit L) ∧ j ≤ w ∧ nat-of-lit L < length ?W ∧
       w ≤ length (?W ! nat-of-lit L)› and
    pre: ‹j ≤ length (?W ! nat-of-lit L) ∧ j ≤ w ∧ nat-of-lit L < length ?W ∧
       w ≤ length (?W ! nat-of-lit L)› and
    I: ‹case s of (j, w, W) ⇒ j ≤ w ∧ nat-of-lit L < length W› and
    I': ‹I' s› and
    cond: ‹case s of (j, w, W) ⇒ w < length (W ! nat-of-lit L)› and
    [simp]: ‹x2 = (x1k, x2k)› and
    [simp]: ‹s = (x1j, x2)›
   for s x1j x2 x1k x2k
  proof −
    have [simp]: ‹x1k < length (x2k ! nat-of-lit L)› and
      ‹length (?W ! nat-of-lit L) − Suc x1k < length (?W ! nat-of-lit L) − x1k›
      using cond I I' unfolding I'-def by auto
    moreover have ‹x1j ≤ x1k› ‹nat-of-lit L < length x2k›
      using I I' unfolding I'-def by auto
    moreover have ‹I' (Suc x1j, Suc x1k, x2k
      [nat-of-lit L := (x2k ! nat-of-lit L)[x1j := x2k ! nat-of-lit L ! x1k]])›
    proof −
      have ball-leI: ‹(⋀x. x < A ⟹ P x) ⟹ (∀ x < A. P x)› for A P
        by auto
      have H: ‹⋀i. x2k[nat-of-lit L := take (j + x1k − w) (x2k ! nat-of-lit L)] ! i = ?W
  [nat-of-lit L :=
     take (min (j + x1k − w) j) (?W ! nat-of-lit L) @
     take (j + x1k − (w + min (length (?W ! nat-of-lit L)) j))
      (drop w (?W ! nat-of-lit L))] ! i› and
        H': ‹x2k[nat-of-lit L := take (j + x1k − w) (x2k ! nat-of-lit L)] = ?W
        [nat-of-lit L :=
     take (min (j + x1k − w) j) (?W ! nat-of-lit L) @
     take (j + x1k − (w + min (length (?W ! nat-of-lit L)) j))
      (drop w (?W ! nat-of-lit L))]› and
        ‹j < length (?W ! nat-of-lit L)› and
        ‹(length (?W ! nat-of-lit L) − w) ≥ (Suc x1k − w)› and
        ‹x1k ≥ w›
        ‹nat-of-lit L < length ?W› and
        ‹j + x1k − w = x1j› and
        ‹x1j − j = x1k − w› and
        ‹x1j < length (?W ! nat-of-lit L)› and
        ‹length (x2k ! nat-of-lit L) = length (?W ! nat-of-lit L)› and
        ‹drop x1k (x2k ! (nat-of-lit L)) = drop x1k (?W ! (nat-of-lit L))›
        ‹x1j ≥ j› and
        ‹w + x1j − j = x1k›
        using I I' pre cond unfolding I'-def by auto
```

```
      have
        [simp]: ‹min x1j j = j›
        using ‹x1j ≥ j› unfolding min-def by auto
      have ‹x2k[nat-of-lit L := take (Suc (j + x1k) − w) (x2k[nat-of-lit L := (x2k ! nat-of-lit L)
              [x1j := x2k ! nat-of-lit L ! x1k]] ! nat-of-lit L)] =
        ?W[nat-of-lit L := take j (?W ! nat-of-lit L) @ take (Suc (j + x1k) − (w + min (length (?W
! nat-of-lit L)) j))
            (drop w (?W ! nat-of-lit L))]›
        using cond I ‹j < length (?W ! nat-of-lit L)› and
        ‹(length (?W ! nat-of-lit L) − w) ≥ (Suc x1k − w)› and
        ‹x1k ≥ w›
        ‹nat-of-lit L < length ?W›
        ‹j + x1k − w = x1j› ‹x1j < length (?W ! nat-of-lit L)›
        apply (subst list-eq-iff-nth-eq)
        apply −
        apply (intro conjI ball-leI)
        subgoal using arg-cong[OF H′, of length] by auto
        subgoal for k
          apply (cases ‹k ≠ nat-of-lit L›)
          subgoal using H[of k] by auto
          subgoal
            using H[of k] ‹x1j < length (?W ! nat-of-lit L)›
              ‹length (x2k ! nat-of-lit L) = length (?W ! nat-of-lit L)›
              arg-cong[OF ‹drop x1k (x2k ! (nat-of-lit L)) = drop x1k (?W ! (nat-of-lit L))›,
                of ‹λxs. xs ! 0›] ‹x1j ≥ j›
            apply (cases ‹Suc x1j = length (?W ! nat-of-lit L)›)
            apply (auto simp add: Suc-diff-le take-Suc-conv-app-nth ‹j + x1k − w = x1j›
              ‹x1j − j = x1k − w›[symmetric] ‹w + x1j − j = x1k›)
              apply (metis (no-types, lifting) append-eq-appendI append-eq-append-conv-if
                nat-in-between-eq(1) take-update-last)
              by (metis (no-types, lifting) ‹x1j < length (?W ! nat-of-lit L)› append.assoc
                take-Suc-conv-app-nth take-update-last)
          done
        done
      then show ?thesis
        unfolding I′-def prod.case
        using I I′ cond unfolding I′-def by (auto simp: Cons-nth-drop-Suc[symmetric])
    qed
    ultimately show ?thesis
      by auto
  qed

  have step: ‹(s, ?W[nat-of-lit L := take j (?W ! nat-of-lit L) @ drop w (?W ! nat-of-lit L)])
    ∈ {((i, j, W′), W). (W′[nat-of-lit L := take i (W′ ! nat-of-lit L)], W) ∈ Id ∧
      i ≤ length (W′ ! nat-of-lit L) ∧ nat-of-lit L < length W′ ∧
 n = length (W′ ! nat-of-lit L)}›
    if
      pre: ‹j ≤ length (?W ! nat-of-lit L) ∧ j ≤ w ∧ nat-of-lit L < length ?W ∧
    w ≤ length (?W ! nat-of-lit L)› and
      ‹j ≤ length (?W ! nat-of-lit L) ∧ j ≤ w ∧ nat-of-lit L < length ?W ∧
    w ≤ length (?W ! nat-of-lit L)› and
      ‹case s of (j, w, W) ⇒ j ≤ w ∧ nat-of-lit L < length W› and
      ‹I′ s› and
      ‹¬ (case s of (j, w, W) ⇒ w < length (W ! nat-of-lit L))›
    for s
  proof −
```

```
    obtain j' w' W' where s: ‹s = (j', w', W')› by (cases s)
    have
      ‹¬ w' < length (W' ! nat-of-lit L)› and
      ‹j ≤ length (?W ! nat-of-lit L)› and
      ‹j' ≤ w'› and
      ‹nat-of-lit L < length W'› and
      [simp]: ‹length (W' ! nat-of-lit L) = length (?W ! nat-of-lit L)› and
      ‹j ≤ w› and
      ‹j' ≤ w'› and
      ‹nat-of-lit L < length ?W› and
      ‹w ≤ length (?W ! nat-of-lit L)› and
      ‹w ≤ w'› and
      ‹w' − w = j' − j› and
      ‹j ≤ j'› and
      ‹drop w' (W' ! nat-of-lit L) = drop w' (?W ! nat-of-lit L)› and
      ‹w' ≤ length (W' ! nat-of-lit L)› and
      L-le-W: ‹nat-of-lit L < length ?W› and
      eq: ‹W'[nat-of-lit L := take (j + w' − w) (W' ! nat-of-lit L)] =
          ?W[nat-of-lit L := take (j + w' − w) (take j (?W ! nat-of-lit L) @ drop w (?W ! nat-of-lit
L))]›
      using that unfolding I'-def that prod.case s
      by blast+
    then have
      j-j': ‹j + w' − w = j'› and
      j-le: ‹j + w' − w = length (take j (?W ! nat-of-lit L) @ drop w (?W ! nat-of-lit L))› and
      w': ‹w' = length (?W ! nat-of-lit L)›
      by auto
    have [simp]: ‹length ?W = length W'›
      using arg-cong[OF eq, of length] by auto
    show ?thesis
      using eq ‹j ≤ w› ‹w ≤ length (?W ! nat-of-lit L)› ‹j ≤ j'› ‹w' − w = j' − j›
        ‹w ≤ w'› w' L-le-W
      unfolding j-j' j-le s n
      by (auto simp: min-def split: if-splits)
  qed

  have HHH: ‹X ≤ RES (R⁻¹ `` {S}) ⟹ X ≤ ⇓ R (RETURN S)› for X S R
    by (auto simp: RETURN-def conc-fun-RES)

  show ?thesis
    unfolding cut-watch-list-heur2-def cut-watch-list-heur-alt-def prod.case
    apply (rewrite at ‹let - = get-watched-wl-heur - in -› Let-def)
    unfolding n[symmetric]
    apply (rewrite at ‹let - = n in -› Let-def)
    apply (refine-vcg WHILEIT-rule-stronger-inv-RES[where R = ?R and
      I' = I' and Φ = ‹{((i, j, W'), W). (W'[nat-of-lit L := take i (W' ! nat-of-lit L)], W) ∈ Id ∧
        i ≤ length (W' ! nat-of-lit L) ∧ nat-of-lit L < length W' ∧
  n = length (W' ! nat-of-lit L)}⁻¹ `` -›] HHH)
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by (auto simp: )
    subgoal by auto
    subgoal by auto
    subgoal unfolding I'-def by (auto simp: n)
```

subgoal unfolding *I′-def* **by** (*auto simp*: *n*)
subgoal unfolding *I′-def* **by** (*auto simp*: *n*)
subgoal unfolding *I′-def* **by** *auto*
subgoal unfolding *I′-def* **by** *auto*
subgoal unfolding *I′-def* **by** (*auto simp*: *n*)
subgoal using *REC* **by** (*auto simp*: *n*)
subgoal unfolding *I′-def* **by** (*auto simp*: *n*)
subgoal for *s* **using** *step*[*of* ‹*s*›] **unfolding** *I′-def* **by** (*auto simp*: *n*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

lemma *vdom-m-cut-watch-list*:
‹*set xs* ⊆ *set* (*W L*) ⟹ *vdom-m* $\mathcal{A}$ (*W*(*L* := *xs*)) *d* ⊆ *vdom-m* $\mathcal{A}$ *W d*›
by (*cases* ‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›)
(*force simp*: *vdom-m-def split*: *if-splits*)+

The following order allows the rule to be used as a destruction rule, make it more useful for refinement proofs.

lemma *vdom-m-cut-watch-listD*:
‹*x* ∈ *vdom-m* $\mathcal{A}$ (*W*(*L* := *xs*)) *d* ⟹ *set xs* ⊆ *set* (*W L*) ⟹ *x* ∈ *vdom-m* $\mathcal{A}$ *W d*›
using *vdom-m-cut-watch-list*[*of xs W L*] **by** *auto*

lemma *cut-watch-list-heur-cut-watch-list-heur*:
‹(*uncurry3 cut-watch-list-heur*, *uncurry3 cut-watch-list*) ∈
[λ(((*j*, *w*), *L*), *S*). *True*]$_f$
nat-rel $\times_f$ *nat-rel* $\times_f$ *nat-lit-lit-rel* $\times_f$ *twl-st-heur″* $\mathcal{D}$ *r lcount* → ⟨*twl-st-heur″* $\mathcal{D}$ *r lcount*⟩*nres-rel*›
unfolding *cut-watch-list-heur-def cut-watch-list-def uncurry-def*
$\mathcal{L}_{all}$*-all-atms-all-lits*[*symmetric*]
apply (*intro frefI nres-relI*)
apply *refine-vcg*
subgoal
by (*auto simp*: *cut-watch-list-heur-def cut-watch-list-def twl-st-heur′-def*
twl-st-heur-def map-fun-rel-def2)
subgoal
by (*auto simp*: *cut-watch-list-heur-def cut-watch-list-def twl-st-heur′-def*
twl-st-heur-def map-fun-rel-def2)
subgoal
by (*auto simp*: *cut-watch-list-heur-def cut-watch-list-def twl-st-heur′-def*
twl-st-heur-def map-fun-rel-def2)
subgoal
by (*auto simp*: *cut-watch-list-heur-def cut-watch-list-def twl-st-heur′-def*
twl-st-heur-def map-fun-rel-def2)
subgoal
by (*auto simp*: *cut-watch-list-heur-def cut-watch-list-def twl-st-heur′-def*
twl-st-heur-def map-fun-rel-def2 vdom-m-cut-watch-list set-take-subset
set-drop-subset dest!: *vdom-m-cut-watch-listD*
dest!: *in-set-dropD in-set-takeD*)
done

lemma *unit-propagation-inner-loop-wl-D-heur-unit-propagation-inner-loop-wl-D*:
‹(*uncurry unit-propagation-inner-loop-wl-D-heur*, *uncurry unit-propagation-inner-loop-wl*) ∈
[λ(*L*, *S*). *length*(*watched-by S L*) ≤ *r*−*MIN-HEADER-SIZE*]$_f$
nat-lit-lit-rel $\times_f$ *twl-st-heur″* $\mathcal{D}$ *r lcount* → ⟨*twl-st-heur″* $\mathcal{D}$ *r lcount*⟩ *nres-rel*›

proof −
have *length-le*: ‹*length* (*watched-by x2b x1b*) ≤ *r* − *MIN-HEADER-SIZE*› **and**
length-eq: ‹*length* (*watched-by x2b x1b*) = *length* (*watched-by* (*snd y*) (*fst y*))› **and**
eq: ‹*x1b* = *fst y*›
if
‹*case y of* (*L*, *S*) ⇒ *length* (*watched-by S L*) ≤ *r*−*MIN-HEADER-SIZE*› **and**
‹(*x*, *y*) ∈ *nat-lit-lit-rel* ×*f* *twl-st-heur''* 𝒟 *r lcount*› **and**
‹*y* = (*x1*, *x2*)› **and**
‹*x* = (*x1a*, *x2a*)› **and**
‹(*x1*, *x2*) = (*x1b*, *x2b*)›
for *x y x1 x2 x1a x2a x1b x2b r*
using *that* **by** *auto*
show *?thesis*
unfolding *unit-propagation-inner-loop-wl-D-heur-def*
unit-propagation-inner-loop-wl-def uncurry-def
apply (*intro frefI nres-relI*)
apply (*refine-vcg cut-watch-list-heur-cut-watch-list-heur*[*of* 𝒟 *r*, *THEN fref-to-Down-curry3*]
unit-propagation-inner-loop-wl-loop-D-heur-unit-propagation-inner-loop-wl-loop-D[*of r - -* 𝒟 *lcount*,
THEN fref-to-Down-curry])

apply (*rule length-le*; *assumption*)
apply (*rule eq*; *assumption*)
apply (*rule length-eq*; *assumption*)
subgoal by *auto*
subgoal by (*auto simp*: *twl-st-heur'-def twl-st-heur-state-simp-watched*)
subgoal
by (*auto simp*: *twl-st-heur'-def twl-st-heur-state-simp-watched watched-by-alt-def*
simp flip: *all-lits-st-alt-def*[*symmetric*])
apply (*rule order.trans*)
apply (*rule cut-watch-list-heur2-cut-watch-list-heur*)
apply (*subst Down-id-eq*)
apply (*rule cut-watch-list-heur-cut-watch-list-heur*[*of* 𝒟, *THEN fref-to-Down-curry3*])
by *auto*
qed

lemma *select-and-remove-from-literals-to-update-wl-heur-select-and-remove-from-literals-to-update-wl*:
‹*literals-to-update-wl y* ≠ {#} ⟹
(*x*, *y*) ∈ *twl-st-heur''* 𝒟*1 r1 lcount* ⟹
select-and-remove-from-literals-to-update-wl-heur x
≤ ⇓{((*S*, *L*), (*S'*, *L'*)). ((*S*, *L*), (*S'*, *L'*)) ∈ *twl-st-heur''* 𝒟*1 r1 lcount* ×*f* *nat-lit-lit-rel* ∧
S' = *set-literals-to-update-wl* (*literals-to-update-wl y* − {#*L*#}) *y* ∧
get-clauses-wl-heur S = *get-clauses-wl-heur x*}
(*select-and-remove-from-literals-to-update-wl y*)›
supply *RETURN-as-SPEC-refine*[*refine2*]
unfolding *select-and-remove-from-literals-to-update-wl-heur-def*
select-and-remove-from-literals-to-update-wl-def
apply (*refine-vcg*)
subgoal
by (*subst trail-pol-same-length*[*of* ‹*get-trail-wl-heur x*› ‹*get-trail-wl y*› ‹*all-atms-st y*›])
(*auto simp*: *twl-st-heur-def twl-st-heur'-def RETURN-RES-refine-iff*)
subgoal
by (*auto simp*: *twl-st-heur-def twl-st-heur'-def RETURN-RES-refine-iff trail-pol-alt-def*)
subgoal
apply (*subst* (*asm*) *trail-pol-same-length*[*of* ‹*get-trail-wl-heur x*› ‹*get-trail-wl y*› ‹*all-atms-st y*›])
apply (*auto simp*: *twl-st-heur-def twl-st-heur'-def*; *fail*)[]
apply (*rule bind-refine-res*)

```
    prefer 2
    apply (rule isa-trail-nth-rev-trail-nth[THEN fref-to-Down-curry, unfolded comp-def RETURN-def,
      unfolded conc-fun-RES, of ‹get-trail-wl y› - - - ‹all-atms-st y›])
    apply ((auto simp: twl-st-heur-def twl-st-heur'-def; fail)+)[2]
    subgoal for z
      apply (cases x; cases y)
      by (auto simp add: Cons-nth-drop-Suc[symmetric] twl-st-heur-def twl-st-heur'-def
        RETURN-RES-refine-iff rev-trail-nth-def)
    done
  done

lemma outer-loop-length-watched-le-length-arena:
  assumes
    xa-x': ‹(xa, x') ∈ twl-st-heur'' 𝒟 r lcount› and
    prop-heur-inv: ‹unit-propagation-outer-loop-wl-D-heur-inv x xa› and
    prop-inv: ‹unit-propagation-outer-loop-wl-inv x'› and
    xb-x'a: ‹(xb, x'a) ∈ {((S, L), (S', L')). ((S, L), (S', L')) ∈ twl-st-heur'' 𝒟1 r lcount ×f nat-lit-lit-rel
∧
          S' = set-literals-to-update-wl (literals-to-update-wl x' − {#L#}) x' ∧
          get-clauses-wl-heur S = get-clauses-wl-heur xa}› and
    st: ‹x'a = (x1, x2)›
      ‹xb = (x1a, x2a)› and
    x2: ‹x2 ∈# all-lits-st x1› and
    st': ‹(x2, x1) = (x1b, x2b)›
  shows ‹length (watched-by x2b x1b) ≤ r−MIN-HEADER-SIZE›
proof −
  have ‹correct-watching x'›
    using prop-inv unfolding unit-propagation-outer-loop-wl-inv-def
      unit-propagation-outer-loop-wl-inv-def
    by auto
  moreover have ‹x2 ∈# all-lits-st x'›
    using x2 assms unfolding all-atms-def all-lits-def
    by (auto simp: 𝓛all-atm-of-all-lits-of-mm correct-watching.simps)
  ultimately have dist: ‹distinct-watched (watched-by x' x2)›
    using x2 xb-x'a unfolding all-atms-def all-lits-def
    by (cases x'; auto simp: 𝓛all-atm-of-all-lits-of-mm correct-watching.simps ac-simps)
  then have dist: ‹distinct-watched (watched-by x1 x2)›
    using xb-x'a unfolding st
    by (cases x'; auto simp: 𝓛all-atm-of-all-lits-of-mm correct-watching.simps)
  have dist-vdom: ‹distinct (get-vdom x1a)›
    using xb-x'a
    by (cases x')
      (auto simp: twl-st-heur-def twl-st-heur'-def st aivdom-inv-dec-alt-def)

  have
      valid: ‹valid-arena (get-clauses-wl-heur xa) (get-clauses-wl x1) (set (get-vdom x1a))›
    using xb-x'a unfolding all-atms-def all-lits-def st
    by (cases x')
     (auto simp: twl-st-heur'-def twl-st-heur-def)

  have ‹vdom-m (all-atms-st x1) (get-watched-wl x1) (get-clauses-wl x1) ⊆ set (get-vdom x1a)›
    using xb-x'a
    by (cases x')
      (auto simp: twl-st-heur-def twl-st-heur'-def st)
  then have subset: ‹set (map fst (watched-by x1 x2)) ⊆ set (get-vdom x1a)›
    using x2 unfolding vdom-m-def st all-lits-st-alt-def[symmetric]
```

```
    by (cases x1)
      (force simp: twl-st-heur'-def twl-st-heur-def
        dest!: multi-member-split)
  have watched-incl: ‹mset (map fst (watched-by x1 x2)) ⊆# mset (get-vdom x1a)›
    by (rule distinct-subseteq-iff[THEN iffD1])
      (use dist[unfolded distinct-watched-alt-def] dist-vdom subset in
        ‹simp-all flip: distinct-mset-mset-distinct›)
  have vdom-incl: ‹set (get-vdom x1a) ⊆ {MIN-HEADER-SIZE..< length (get-clauses-wl-heur xa)}›
    using valid-arena-in-vdom-le-arena[OF valid] arena-dom-status-iff[OF valid] by auto

  have ‹length (get-vdom x1a) ≤ length (get-clauses-wl-heur xa) − MIN-HEADER-SIZE›
    by (subst distinct-card[OF dist-vdom, symmetric])
      (use card-mono[OF - vdom-incl] in auto)
  then show ?thesis
    using size-mset-mono[OF watched-incl] xb-x'a st'
    by auto
qed

lemma unit-propagation-outer-loop-wl-D-heur-alt-def:
  ‹unit-propagation-outer-loop-wl-D-heur S_0 = do {
  S ← WHILE_T^unit-propagation-outer-loop-wl-D-heur-inv S_0
  (λS. literals-to-update-wl-heur S < isa-length-trail (get-trail-wl-heur S))
  (λS. do {
  ASSERT(literals-to-update-wl-heur S < isa-length-trail (get-trail-wl-heur S));
  (S', L) ← select-and-remove-from-literals-to-update-wl-heur S;
  ASSERT(length (get-clauses-wl-heur S') = length (get-clauses-wl-heur S));
  unit-propagation-inner-loop-wl-D-heur L S'
  })
  S_0;
  unit-propagation-update-statistics (of-nat (isa-length-trail (get-trail-wl-heur S_0)))
    (of-nat (isa-length-trail (get-trail-wl-heur S))) S
  } ›
  unfolding unit-propagation-outer-loop-wl-D-heur-def IsaSAT-Profile.start-def IsaSAT-Profile.stop-def
    Let-def
  by auto

lemma unit-propagation-outer-loop-wl-alt-def:
  ‹unit-propagation-outer-loop-wl S_0 = do {
    S ← WHILE_T^unit-propagation-outer-loop-wl-inv
      (λS. literals-to-update-wl S ≠ {#})
      (λS. do {
        ASSERT(literals-to-update-wl S ≠ {#});
        (S', L) ← select-and-remove-from-literals-to-update-wl S;
        ASSERT(L ∈# all-lits-st S');
        unit-propagation-inner-loop-wl L S'
      })
      (S_0 :: 'v twl-st-wl);
   RETURN S}
›
  unfolding unit-propagation-outer-loop-wl-def by auto

lemma unit-propagation-update-statistics-twl-st-heur'':
  ‹(S, x') ∈ twl-st-heur'' D r lcount ⟹
    unit-propagation-update-statistics a b S ≤ ⇓(twl-st-heur'' D r lcount) (RETURN x')›
  unfolding unit-propagation-update-statistics-def Let-def conc-fun-RETURN
```

apply (*refine-vcg trail-height-before-conflict*[**where** $\mathcal{A}$ = ‹*all-atms-st x'*›, *THEN fref-to-Down*, *of* - ‹*get-trail-wl x'*›, *THEN order-trans*])
subgoal
by (*auto simp*: *twl-st-heur-def twl-st-heur'-def*)
subgoal
by (*auto simp*: *twl-st-heur-def twl-st-heur'-def*)
subgoal
by (*auto simp*: *twl-st-heur-def twl-st-heur'-def*)
subgoal
by (*auto simp*: *twl-st-heur-def twl-st-heur'-def trail-height-before-conflict-spec-def*)
done

theorem *unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D'*:
‹(*unit-propagation-outer-loop-wl-D-heur*, *unit-propagation-outer-loop-wl*) ∈
twl-st-heur'' $\mathcal{D}$ *r lcount* →$_f$ ⟨*twl-st-heur''* $\mathcal{D}$ *r lcount*⟩ *nres-rel*›
unfolding *unit-propagation-outer-loop-wl-D-heur-alt-def*
unit-propagation-outer-loop-wl-alt-def all-lits-alt-def2[*symmetric*]
apply (*intro frefI nres-relI*)
apply (*refine-vcg*
unit-propagation-update-statistics-twl-st-heur''
unit-propagation-inner-loop-wl-D-heur-unit-propagation-inner-loop-wl-D[*of r* $\mathcal{D}$ *lcount*, *THEN fref-to-Down-curry*]
select-and-remove-from-literals-to-update-wl-heur-select-and-remove-from-literals-to-update-wl
[*of* - - $\mathcal{D}$ *r lcount*])
subgoal for *x y S T*
using *isa-length-trail-pre*[*of* ‹*get-trail-wl-heur S*› ‹*get-trail-wl T*› ‹*all-atms-st T*›] **apply** −
unfolding *unit-propagation-outer-loop-wl-D-heur-inv-def twl-st-heur'-def*
apply (*rule-tac x*=*y* **in** *exI*)
apply (*rule-tac x*=*T* **in** *exI*)
by (*auto 5 2 simp*: *twl-st-heur-def twl-st-heur'-def*)
subgoal for - - *x y*
by (*subst isa-length-trail-length-u*[*THEN fref-to-Down-unRET-Id*, *of* - ‹*get-trail-wl y*› ‹*all-atms-st y*›])
(*auto simp*: *twl-st-heur-def twl-st-heur'-def simp flip*: *all-lits-st-alt-def*)
subgoal by (*auto simp*: *twl-st-heur'-def*)
subgoal for *x y xa x' xb x'a x1 x2 x1a x2a x1b x2b*
by (*rule-tac x*=*x* **and** *xa*=*xa* **and** $\mathcal{D}$=$\mathcal{D}$ **in** *outer-loop-length-watched-le-length-arena*)
subgoal by (*auto simp*: *twl-st-heur'-def*)
done

lemma *twl-st-heur'D-twl-st-heurD*:
assumes *H*: ‹(⋀$\mathcal{D}$. *f* ∈ *twl-st-heur'* $\mathcal{D}$ →$_f$ ⟨*twl-st-heur'* $\mathcal{D}$⟩ *nres-rel*)›
shows ‹*f* ∈ *twl-st-heur* →$_f$ ⟨*twl-st-heur*⟩ *nres-rel*› (**is** ‹- ∈ *?A B*›)
proof −
obtain *f1 f2* **where** *f*: ‹*f* = (*f1*, *f2*)›
by (*cases f*) *auto*
show *?thesis*
using *assms* **unfolding** *f*
apply (*simp only*: *fref-def twl-st-heur'-def nres-rel-def in-pair-collect-simp*)
apply (*intro conjI impI allI*)
subgoal for *x y*
apply (*rule weaken-⇓'*[*of* - ‹*twl-st-heur'* (*dom-m* (*get-clauses-wl y*))›])
apply (*fastforce simp*: *twl-st-heur'-def*)+
done
done
qed

lemma *watched-by-app-watched-by-app-heur*:
‹(*uncurry2* (*RETURN ooo watched-by-app-heur*), *uncurry2* (*RETURN ooo watched-by-app*)) ∈
[λ((*S*, *L*), *K*). *L* ∈# *all-lits-st S* ∧ *K* < *length* (*get-watched-wl S L*)]$_f$
twl-st-heur ×$_f$ *Id* ×$_f$ *Id* → ⟨*Id*⟩ *nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *watched-by-app-heur-def watched-by-app-def twl-st-heur-def map-fun-rel-def2*)

lemma *update-clause-wl-heur-pre-le-sint64*:
assumes
‹*arena-is-valid-clause-idx-and-access* (*get-clauses-wl-heur S*) *bf baa*› **and**
‹*length* (*get-clauses-wl-heur S*)≤ *snat64-max*› **and**
‹*arena-lit-pre* (*get-clauses-wl-heur S*) (*bf* + *baa*)›
shows ‹*bf* + *baa* ≤ *snat64-max*›
‹*length* (*get-clauses-wl-heur S*) ≤ *snat64-max*›
using *assms*
by (*auto simp*: *arena-is-valid-clause-idx-and-access-def isasat-fast-def*
dest!: *arena-lifting*(*10*))

end
theory *IsaSAT-LBD-LLVM*
imports *IsaSAT-LBD IsaSAT-Setup0-LLVM*
begin

sepref-register *mark-lbd-from-clause-heur get-level-pol mark-lbd-from-list-heur*
mark-lbd-from-conflict mop-arena-status

sepref-def *mark-lbd-from-clause-heur-impl*
is ‹*uncurry3 mark-lbd-from-clause-heur*›
:: ‹*trail-pol-fast-assn*^{k} *∗*$_a$ *arena-fast-assn*^{k} *∗*$_a$ *sint64-nat-assn*^{k} *∗*$_a$ *lbd-assn*^{d} →$_a$ *lbd-assn*›
unfolding *mark-lbd-from-clause-heur-def nfoldli-upt-by-while*
apply (*rewrite at* ‹- = ⌑› *unat-const-fold*[**where** *'a*=*32*])
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *calculate-LBD-heur-st-impl*
is ‹*uncurry3 calculate-LBD-heur-st*›
:: ‹*trail-pol-fast-assn*^{k} *∗*$_a$ *arena-fast-assn*^{d} *∗*$_a$ *lbd-assn*^{d} *∗*$_a$ *sint64-nat-assn*^{k} →$_a$
arena-fast-assn ×$_a$ *lbd-assn*›
supply [[*goals-limit*=*1*]]
unfolding *calculate-LBD-heur-st-def isasat-bounded-assn-def*
fold-tuple-optimizations
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

sepref-def *mark-lbd-from-list-heur-impl*
is ‹*uncurry2 mark-lbd-from-list-heur*›
:: ‹*trail-pol-fast-assn*^{k} *∗*$_a$ *out-learned-assn*^{k} *∗*$_a$ *lbd-assn*^{d} →$_a$ *lbd-assn*›
supply [[*goals-limit*=*1*]]
unfolding *mark-lbd-from-list-heur-def nfoldli-upt-by-while*
apply (*rewrite at* ‹- = ⌑› *unat-const-fold*[**where** *'a*=*32*])
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *mark-lbd-from-conflict-alt-def*:
‹*mark-lbd-from-conflict* = (λ*S*. *do*{

```
    let (M, S) = extract-trail-wl-heur S;
    let (outl, S) = extract-outl-wl-heur S;
    let (lbd, S) = extract-lbd-wl-heur S;
    lbd ← mark-lbd-from-list-heur M outl lbd;
    RETURN (update-lbd-wl-heur lbd (update-trail-wl-heur M (update-outl-wl-heur outl S)))
   })›
  by (auto simp: state-extractors mark-lbd-from-conflict-def split: isasat-int-splits)

sepref-def mark-lbd-from-conflict-impl
  is ‹mark-lbd-from-conflict›
  :: ‹isasat-bounded-assn^d →_a isasat-bounded-assn›
  supply [[goals-limit=1]]
  unfolding mark-lbd-from-conflict-alt-def
  by sepref

lemma update-lbd-pre-arena-act-preD:
  ‹update-lbd-pre ((a, ba), b) ⟹
  arena-act-pre (update-lbd a ba b) a›
  unfolding update-lbd-pre-def arena-act-pre-def prod.simps
  by (auto simp: arena-is-valid-clause-idx-def intro!: valid-arena-update-lbd)

sepref-register update-lbd-and-mark-used
sepref-def update-lbd-and-mark-used-impl
  is ‹uncurry2 (RETURN ooo update-lbd-and-mark-used)›
    :: ‹[update-lbd-pre]_a sint64-nat-assn^k *_a uint32-nat-assn^k *_a arena-fast-assn^d → arena-fast-assn›
  unfolding update-lbd-and-mark-used-def LBD-SHIFT-def
  supply [dest] = update-lbd-pre-arena-act-preD
  apply (annot-unat-const ‹TYPE(32)›)
  by sepref

sepref-def update-lbd-shrunk-clause-impl
  is ‹uncurry update-lbd-shrunk-clause›
  :: ‹sint64-nat-assn^k *_a arena-fast-assn^d →_a arena-fast-assn›
  unfolding update-lbd-shrunk-clause-def
  apply (rewrite  at ‹If (¤ ≤ -)› annot-unat-snat-upcast[where 'l=64])
  apply (rewrite at ‹If (- ≤ (- − ¤))› snat-const-fold[where 'a=64])
  apply (rewrite at ‹If (- ≤ -) - (¤ − -)› annot-snat-unat-conv)
  apply (rewrite  at ‹If (- ≤ -) - (¤ − -)› annot-unat-unat-downcast[where 'l=32])
  apply (annot-unat-const ‹TYPE(32)›)
  by sepref

end
theory IsaSAT-Inner-Propagation-LLVM
  imports IsaSAT-Setup-LLVM
    IsaSAT-Inner-Propagation-Defs
    IsaSAT-VMTF-LLVM
    IsaSAT-LBD-LLVM
begin
hide-const (open) NEMonad.ASSERT NEMonad.RETURN
sepref-register isa-save-pos unit-propagation-update-statistics

lemma unit-propagation-update-statistics-alt-def:
  ‹unit-propagation-update-statistics p q S = do {
  let (stats, S) = extract-stats-wl-heur S;
  let (M, S) = extract-trail-wl-heur S;
  let pq = q − p;
```

let stats = incr-propagation-by pq stats;
let stats = (if get-conflict-wl-is-None-heur S then stats else incr-conflict stats);
let stats = (if count-decided-pol M = 0 then incr-units-since-last-GC-by pq (incr-uset-by pq stats) else stats);
height ← (if get-conflict-wl-is-None-heur S then RETURN q else do {j ← trail-height-before-conflict M; RETURN (of-nat j)});
let stats = set-no-conflict-until q stats;
RETURN (update-stats-wl-heur stats (update-trail-wl-heur M S))
}›
by (*auto simp*: *unit-propagation-update-statistics-def state-extractors Let-def get-conflict-wl-is-None-heur-def*
split: *isasat-int-splits intro*!: *ext*)

sepref-def *unit-propagation-update-statistics-impl*
is ‹*uncurry2* (*unit-propagation-update-statistics*)›
:: ‹*word64-assn*^{k} $*_a$ *word64-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\to_a$ *isasat-bounded-assn*›
supply [[*goals-limit=1*]] *of-nat-unat*[*sepref-import-param*]
unfolding *unit-propagation-update-statistics-alt-def*
apply (*annot-unat-const* ‹*TYPE* (*32*)›)
apply (*rewrite at* ‹*RETURN* (*word-of-nat* ¤)› *annot-unat-unat-upcast*[**where** *'l=64*])
by *sepref*

lemma *isa-save-pos-alt-def*:
‹*isa-save-pos C i* = (λS_0. *do* {
ASSERT(*arena-is-valid-clause-idx* (*get-clauses-wl-heur* S_0) *C*);
if arena-length (*get-clauses-wl-heur* S_0) *C* > *MAX-LENGTH-SHORT-CLAUSE then do* {
let (*N*, *S*) = *extract-arena-wl-heur* S_0;
ASSERT (*N* = *get-clauses-wl-heur* S_0);
ASSERT(*isa-update-pos-pre* ((*C*, *i*), *N*));
let N = *arena-update-pos C i N*;
RETURN (*update-arena-wl-heur N S*)
} *else RETURN* S_0
})
›
by (*auto simp*: *isa-save-pos-def state-extractors*
split: *isasat-int-splits intro*!: *ext*)

sepref-def *isa-save-pos-fast-code*
is ‹*uncurry2 isa-save-pos*›
:: ‹*sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\to_a$ *isasat-bounded-assn*›
supply
[[*goals-limit=1*]]
if-splits[*split*]
unfolding *isa-save-pos-alt-def PR-CONST-def access-length-heur-def*[*symmetric*]
by *sepref*

sepref-register *isa-find-unwatched-wl-st-heur isa-find-unwatched-between isa-find-unset-lit*
polarity-pol

sepref-register *0 1*

sepref-def *isa-find-unwatched-between-fast-code*

is ‹*uncurry4 isa-find-unset-lit*›
:: ‹[λ((((*M*, *N*), -), -), -). *length N* ≤ *snat64-max*]$_a$
trail-pol-fast-assn^k *∗$_a$ arena-fast-assn*^{k} *∗$_a$ sint64-nat-assn*^{k} *∗$_a$ sint64-nat-assn*^{k} *∗$_a$ sint64-nat-assn*^{k}
→
snat-option-assn′ TYPE(*64*)›
supply [[*goals-limit* = *3*]]
unfolding *isa-find-unset-lit-def isa-find-unwatched-between-def SET-FALSE-def*[*symmetric*]
PR-CONST-def
apply (*rewrite* **in** ‹*if* ¤ *then - else -*› *tri-bool-eq-def*[*symmetric*])
apply (*annot-snat-const* ‹*TYPE* (*64*)›)
by *sepref*

definition *isa-find-unset-lit-st* **where**
‹*isa-find-unset-lit-st S* = *isa-find-unset-lit* (*get-trail-wl-heur S*) (*get-clauses-wl-heur S*)›

definition *isasat-find-unset-lit-st-impl* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*isasat-find-unset-lit-st-impl* = (λ*N C D E*.
read-all-st-code
(λ*M N* - - - - - - - - - - - - - - - -. *isa-find-unwatched-between-fast-code M N C D E*) *N*)›

global-interpretation *find-unset-lit*: *read-trail-arena-param-adder2-threeargs* **where**
R = ‹*snat-rel′* (*TYPE*(*64*))› **and**
R′ = ‹*snat-rel′* (*TYPE*(*64*))› **and**
R″ = ‹*snat-rel′* (*TYPE*(*64*))› **and**
f = ‹λ*C C′ C″ M N*. *isa-find-unwatched-between-fast-code M N C C′ C″*› **and**
f′ = ‹λ*C C′ C″ M N*. *isa-find-unset-lit M N C C′ C″*› **and**
x-assn = ‹*snat-option-assn′ TYPE*(*64*)› **and**
P = ‹λ*C C′ C″ M N*. *length N* ≤ *snat64-max*›
rewrites
‹(λ*N C D E*.
read-all-st (λ*M N* - - - - - - - - - - - - - - - -. *isa-find-unset-lit M N C D E*) *N*) = *isa-find-unset-lit-st*›
and
‹(λ*N C D E*.
read-all-st-code
(λ*M N* - - - - - - - - - - - - - - - - -. *isa-find-unwatched-between-fast-code M N C D E*) *N*) = *isasat-find-unset-lit-st-impl*›
apply (*unfold-locales*)
apply (*subst* (*9*) *uncurry-def*)+
apply (*rule isa-find-unwatched-between-fast-code.refine*)
subgoal by (*auto simp*: *read-all-st-def isa-find-unset-lit-st-def intro*!: *ext split*: *isasat-int-splits*)
subgoal by (*auto simp*: *isasat-find-unset-lit-st-impl-def*)
done

lemmas [*sepref-fr-rules*] = *find-unset-lit.refine*
lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
isasat-find-unset-lit-st-impl-def[*unfolded read-all-st-code-def*]

sepref-def *swap-lits-impl* **is** ‹*uncurry3 mop-arena-swap*›
:: ‹*sint64-nat-assn*^{k} *∗$_a$ sint64-nat-assn*^{k} *∗$_a$ sint64-nat-assn*^{k} *∗$_a$ arena-fast-assn*^{d} →$_a$ *arena-fast-assn*›
unfolding *mop-arena-swap-def swap-lits-pre-def*
unfolding *gen-swap*
by *sepref*

sepref-register *isa-find-unset-lit-st*

lemma *case-tri-bool-If*:

‹(*case a of*
None ⇒ f1
| Some v ⇒
(if v then f2 else f3)) =
(let b = a in if b = UNSET
then f1
else if b = SET-TRUE then f2 else f3)›
by (*auto split*: *option.splits*)

sepref-def *find-unwatched-wl-st-heur-fast-code*
is ‹*uncurry isa-find-unwatched-wl-st-heur*›
:: ‹[λ(*S*, *C*). *length* (*get-clauses-wl-heur S*) ≤ *snat64-max*]$_a$
isasat-bounded-assn^k *$*_a$ *sint64-nat-assn*^{k} → *snat-option-assn*′ *TYPE*(*64*)›
supply [[*goals-limit* = *1*]] *isasat-fast-def*[*simp*]
unfolding *isa-find-unwatched-wl-st-heur-def PR-CONST-def*
fmap-rll-def[*symmetric*]
length-uint32-nat-def[*symmetric*] *isa-find-unwatched-def*
case-tri-bool-If
fmap-rll-u64-def[*symmetric*]
mop-arena-length-st-def[*symmetric*]
mop-arena-pos-st-def[*symmetric*]
apply (*subst isa-find-unset-lit-def*[*symmetric*])+
apply (*subst isa-find-unset-lit-st-def*[*symmetric*])+
apply (*annot-snat-const* ‹*TYPE* (*64*)›)
by *sepref*

lemma *other-watched-wl-heur-alt-def*:
‹*other-watched-wl-heur* = (λ*S*. *arena-other-watched* (*get-clauses-wl-heur S*))›
apply (*intro ext*)
unfolding *other-watched-wl-heur-def*
arena-other-watched-def
mop-access-lit-in-clauses-heur-def
by *auto argo*

definition *clause-not-marked-to-delete-heur-code* :: ‹*twl-st-wll-trail-fast2* ⇒ - ⇒ -› **where**
‹*clause-not-marked-to-delete-heur-code S C*′ = *read-arena-wl-heur-code* (λ*N*. *not-deleted-code N C*′) *S*›

sepref-def *other-watched-wl-heur-impl*
is ‹*uncurry3 other-watched-wl-heur*›
:: ‹*isasat-bounded-assn*^{k} *$*_a$ *unat-lit-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} →$_a$
unat-lit-assn›
supply [[*goals-limit=1*]]
unfolding *other-watched-wl-heur-alt-def*
arena-other-watched-def
mop-access-lit-in-clauses-heur-def[*symmetric*]
apply (*annot-snat-const* ‹*TYPE* (*64*)›)
by *sepref*

sepref-register *update-clause-wl-heur*
setup ‹*map-theory-claset* (*fn ctxt => ctxt delSWrapper split-all-tac*)›

lemma *arena-lit-pre-le2*: ‹
arena-lit-pre a i ⟹ *length a* ≤ *snat64-max* ⟹ *i* < *max-snat 64*›
using *arena-lifting*(*7*)[*of a - -*] **unfolding** *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*

snat64-max-def max-snat-def
by *fastforce*

lemma *snat64-max-le-max-snat64*: ‹*a* < *snat64-max* ⟹ *Suc a* < *max-snat 64*›
by (*auto simp*: *max-snat-def snat64-max-def*)

lemma *update-clause-wl-heur-alt-def*:
‹*update-clause-wl-heur* = (λ(*L*::*nat literal*) *L'* *C b j w i f* S_0. *do* {
let (*N*, *S*) = *extract-arena-wl-heur* S_0;
ASSERT (*N* = *get-clauses-wl-heur* S_0);
let (*W*, *S*) = *extract-watchlist-wl-heur S*;
ASSERT (*W* = *get-watched-wl-heur* S_0);
K' ← *mop-arena-lit2'* (*set* (*get-vdom S*)) *N C f*;
ASSERT(*w* < *length N*);
N' ← *mop-arena-swap C i f N*;
ASSERT(*nat-of-lit K'* < *length W*);
ASSERT(*length* (*W* ! (*nat-of-lit K'*)) < *length N*);
let W = *W*[*nat-of-lit K'*:= *W* ! (*nat-of-lit K'*) @ [(*C*, *L*, *b*)]];
let S = *update-arena-wl-heur N' S*;
let S = *update-watchlist-wl-heur W S*;
RETURN (*j*, *w*+*1*, *S*)
})›
by (*auto intro*!: *ext simp*: *state-extractors update-clause-wl-heur-def*
split: *isasat-int-splits*)

sepref-def *update-clause-wl-fast-code*
is ‹*uncurry8 update-clause-wl-heur*›
:: ‹[λ(((((((*L*, *L'*), *C*), *b*), *j*), *w*), *i*), *f*), *S*). *length* (*get-clauses-wl-heur S*) ≤ *snat64-max*]$_a$
unat-lit-assn^k *$*_a$ *unat-lit-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} *$*_a$ *bool1-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} *$*_a$
sint64-nat-assn^k
*$*_a$ *isasat-bounded-assn*^{d} → *sint64-nat-assn* ×$_a$ *sint64-nat-assn* ×$_a$ *isasat-bounded-assn*›
supply [[*goals-limit*=*1*]] *arena-lit-pre-le2*[*intro*] *swap-lits-pre-def*[*simp*]
snat64-max-le-max-snat64[*intro*]
unfolding *update-clause-wl-heur-alt-def*
fmap-rll-def[*symmetric*] *delete-index-and-swap-update-def*[*symmetric*]
delete-index-and-swap-ll-def[*symmetric*] *fmap-swap-ll-def*[*symmetric*]
append-ll-def[*symmetric*] *update-clause-wl-code-pre-def*
fmap-rll-u64-def[*symmetric*]
fmap-swap-ll-u64-def[*symmetric*]
fmap-swap-ll-def[*symmetric*]
PR-CONST-def mop-arena-lit2'-def
apply (*annot-snat-const* ‹*TYPE* (*64*)›)
by *sepref*

sepref-register *mop-arena-swap*

definition *propagate-lit-wl-heur-inner* :: ‹-› **where**
‹*propagate-lit-wl-heur-inner L' C i* = (λ*M N D j W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*. *do* {
ASSERT(*i* ≤ *1*);
M ← *cons-trail-Propagated-tr L' C M*;
N' ← *mop-arena-swap C 0* (*1* − *i*) *N*;
heur ← *mop-save-phase-heur* (*atm-of L'*) (*is-pos L'*) *heur*;
RETURN (*Tuple17 M N' D j W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs*)
})›

lemma *propagate-lit-wl-heur-propagate-lit-wl-heur-inner*:
  ‹*propagate-lit-wl-heur* = (λ*L'* *C* *i* (*S*$_0$::*isasat*).
  case-isasat-int (*propagate-lit-wl-heur-inner* *L'* *C* *i*)
   S$_0$)›
  by (*auto intro*!: *ext simp*: *state-extractors propagate-lit-wl-heur-def read-all-st-def*
    propagate-lit-wl-heur-inner-def
    split: *isasat-int-splits*)

sepref-def *propagate-lit-wl-fast-code*
  is ‹*uncurry3 propagate-lit-wl-heur*›
  :: ‹[λ(((*L*, *C*), *i*), *S*). *length* (*get-clauses-wl-heur* *S*) ≤ *snat64-max*]$_a$
    unat-lit-assn^k *$*_a$ *sint64-nat-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} *$*_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
  unfolding *PR-CONST-def propagate-lit-wl-heur-propagate-lit-wl-heur-inner*
    RETURN-case-tuple16-invers comp-def propagate-lit-wl-heur-inner-def
  unfolding
    fmap-swap-ll-def[*symmetric*]
    fmap-swap-ll-u64-def[*symmetric*]
    save-phase-def
  apply (*annot-snat-const* ‹*TYPE* (*64*)›)
  supply [[*goals-limit=1*]]
  by *sepref*
lemmas [*llvm-inline*] = *Mreturn-comp-IsaSAT-int*

sepref-register *incr-uset incr-units-since-last-GC*

lemma *propagate-lit-wl-bin-heur-alt2*:
  ‹*propagate-lit-wl-bin-heur* = (λ*L'* *C* (*S*$_0$::*isasat*).
  case-isasat-int (λ*M* *N* *D* *j* *W* *ivmtf* *icount* *ccach* *lbd* *outl* *stats* *heur* *aivdom* *clss* *opts* *arena* *occs*. *do* {
      M ← *cons-trail-Propagated-tr* *L'* *C* *M*;
      heur ← *mop-save-phase-heur* (*atm-of* *L'*) (*is-pos* *L'*) *heur*;
      RETURN (*Tuple17* *M* *N* *D* *j* *W* *ivmtf* *icount* *ccach* *lbd* *outl* *stats* *heur* *aivdom* *clss* *opts* *arena* *occs*)
  })
   S$_0$)›
  by (*auto intro*!: *ext simp*: *state-extractors propagate-lit-wl-bin-heur-def read-all-st-def*
    propagate-lit-wl-heur-inner-def
    split: *isasat-int-splits*)

lemma *propagate-lit-wl-bin-heur-alt-def*:
  ‹*propagate-lit-wl-bin-heur* = (λ*L'* *C* *S*$_0$. *do* {
      let (*M*, *S*) = *extract-trail-wl-heur* *S*$_0$;
      ASSERT (*M* = *get-trail-wl-heur* *S*$_0$);
      let (*heur*, *S*) = *extract-heur-wl-heur* *S*;
      ASSERT (*heur* = *get-heur* *S*$_0$);
      M ← *cons-trail-Propagated-tr* *L'* *C* *M*;
      heur ← *mop-save-phase-heur* (*atm-of* *L'*) (*is-pos* *L'*) *heur*;
      let *S* = *update-trail-wl-heur* *M* *S*;
      let *S* = *update-heur-wl-heur* *heur* *S*;
      RETURN *S*
  })›
  by (*auto intro*!: *ext simp*: *state-extractors propagate-lit-wl-bin-heur-def*
      split: *isasat-int-splits*)

sepref-def *propagate-lit-wl-bin-fast-code*
  is ‹*uncurry2 propagate-lit-wl-bin-heur*›
  :: ‹[λ((*L*, *C*), *S*). *length* (*get-clauses-wl-heur S*) ≤ *snat64-max*]$_a$
    unat-lit-assn^k *$*_a$ *sint64-nat-assn*^{k} *$*_a$ *isasat-bounded-assn*^{d} →
    isasat-bounded-assn›
  unfolding *PR-CONST-def propagate-lit-wl-bin-heur-alt2*
    RETURN-case-tuple16-invers comp-def
  supply [[*goals-limit=1*]] *length-ll-def*[*simp*]
  by *sepref*

lemma *update-blit-wl-heur-alt-def*:
  ‹*update-blit-wl-heur* = (λ(*L*::*nat literal*) *C b j w K S*$_0$. *do* {
    let (*W*, *S*) = *extract-watchlist-wl-heur S*$_0$;
    ASSERT (*W* = *get-watched-wl-heur S*$_0$);
    ASSERT(*nat-of-lit L* < *length W*);
    ASSERT(*j* < *length* (*W* ! *nat-of-lit L*));
    ASSERT(*j* < *length* (*get-clauses-wl-heur S*$_0$));
    ASSERT(*w* < *length* (*get-clauses-wl-heur S*$_0$));
    let W = *W*[*nat-of-lit L* := (*W*!*nat-of-lit L*)[*j*:= (*C*, *K*, *b*)]];
    RETURN (*j*+*1*, *w*+*1*, *update-watchlist-wl-heur W S*)
  })›
  by (*auto intro*!: *ext simp*: *state-extractors update-blit-wl-heur-def*
    split: *isasat-int-splits*)

sepref-def *update-blit-wl-heur-fast-code*
  is ‹*uncurry6 update-blit-wl-heur*›
  :: ‹[λ((((((-, -), -), -), *C*), *i*), *S*). *length* (*get-clauses-wl-heur S*) ≤ *snat64-max*]$_a$
    unat-lit-assn^k *$*_a$ *sint64-nat-assn*^{k} *$*_a$ *bool1-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} *$*_a$
    sint64-nat-assn^k *$*_a$ *unat-lit-assn*^{k} *$*_a$ *isasat-bounded-assn*^{d} →
    sint64-nat-assn ×$_a$ *sint64-nat-assn* ×$_a$ *isasat-bounded-assn*›
  supply [[*goals-limit=1*]] *snat64-max-le-max-snat64*[*intro*]
  unfolding *update-blit-wl-heur-alt-def append-ll-def*[*symmetric*]
    op-list-list-upd-alt-def[*symmetric*]
  apply (*annot-snat-const* ‹*TYPE* (*64*)›)
  by *sepref*

sepref-register *keep-watch-heur*

lemma *op-list-list-take-alt-def*: ‹*op-list-list-take xss i l* = *xss*[*i* := *take l* (*xss* ! *i*)]›
  unfolding *op-list-list-take-def* **by** *auto*

lemma *keep-watch-heur-alt-def*:
  ‹*keep-watch-heur* = (λ*L i j S*. *do* {
    let (*W*, *S*) = *extract-watchlist-wl-heur S*;
    ASSERT(*nat-of-lit L* < *length W*);
    ASSERT(*i* < *length* (*W* ! *nat-of-lit L*));
    ASSERT(*j* < *length* (*W* ! *nat-of-lit L*));
    let W = *W*[*nat-of-lit L* := (*W*!(*nat-of-lit L*))[*i* := *W* ! (*nat-of-lit L*) ! *j*]];
    RETURN (*update-watchlist-wl-heur W S*)
  })›
  by (*auto intro*!: *ext simp*: *state-extractors keep-watch-heur-def*
    split: *isasat-int-splits*)

sepref-def *keep-watch-heur-fast-code*
  is ‹*uncurry3 keep-watch-heur*›

:: ‹*unat-lit-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\to_a$ *isasat-bounded-assn*›
supply
[[*goals-limit=1*]]
unfolding *keep-watch-heur-alt-def PR-CONST-def*
unfolding *fmap-rll-def*[*symmetric*]
unfolding
op-list-list-upd-alt-def[*symmetric*]
nth-rll-def[*symmetric*]
SET-FALSE-def[*symmetric*] *SET-TRUE-def*[*symmetric*]
by *sepref*

sepref-register *unit-propagation-inner-loop-body-wl-heur*

sepref-register *isa-set-lookup-conflict-aa set-conflict-wl-heur mark-conflict-to-rescore*

lemma *mark-conflict-to-rescore-alt-def*:
‹*mark-conflict-to-rescore C* S_0 = *do* {
let (*M*, *S*) = *extract-trail-wl-heur* S_0;
let (*N*, *S*) = *extract-arena-wl-heur S*;
let (*vm*, *S*) = *extract-vmtf-wl-heur S*;
n ← *mop-arena-length N C*;
ASSERT (*n* ≤ *length* (*get-clauses-wl-heur* S_0));
(-, *vm*) ← *WHILE*$_T$ (λ(*i*, *vm*). *i* < *n*)
(λ(*i*, *vm*). *do*{
ASSERT (*i* < *n*);
L ← *mop-arena-lit2 N C i*;
vm ← *isa-vmtf-bump-to-rescore-also-reasons-cl M N C* (−*L*) *vm*;
RETURN (*i*+*1*, *vm*)
})
(*0*, *vm*);
let (*lbd*, *S*) = *extract-lbd-wl-heur S*;
(*N*, *lbd*) ← *calculate-LBD-heur-st M N lbd C*;
let S = *update-trail-wl-heur M S*;
let S = *update-arena-wl-heur N S*;
let S = *update-vmtf-wl-heur vm S*;
let S = *update-lbd-wl-heur lbd S*;
RETURN S }›
by (*auto intro*!: *ext simp*: *state-extractors mark-conflict-to-rescore-def Let-def*
split: *isasat-int-splits*)

sepref-register *isa-vmtf-bump-to-rescore-also-reasons-cl*
sepref-def *mark-conflict-to-rescore-impl*
is ‹*uncurry mark-conflict-to-rescore*›
:: ‹*sint64-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\to_a$ *isasat-bounded-assn*›
unfolding *mark-conflict-to-rescore-alt-def*
apply (*annot-snat-const* ‹*TYPE* (*64*)›)
by *sepref*

lemma *set-conflict-wl-heur-alt-def*:
‹*set-conflict-wl-heur* = (λ*C* S_0. *do* {
let n = *0*;
let (*M*, *S*) = *extract-trail-wl-heur* S_0;
let (*N*, *S*) = *extract-arena-wl-heur S*;
let (*D*, *S*) = *extract-conflict-wl-heur S*;

let (*outl*, *S*) = *extract-outl-wl-heur S*;
ASSERT(*curry5 isa-set-lookup-conflict-aa-pre M N C D n outl*);
(*D*, *clvls*, *outl*) ← *isa-set-lookup-conflict-aa M N C D n outl*;
j ← *mop-isa-length-trail M*;
let S = *update-conflict-wl-heur D S*;
let S = *update-outl-wl-heur outl S*;
let S = *update-clvls-wl-heur clvls S*;
let S = *update-literals-to-update-wl-heur j S*;
let S = *update-trail-wl-heur M S*;
let S = *update-arena-wl-heur N S*;
RETURN S})›
by (*auto intro*!: *ext bind-cong*[*OF refl*] *simp*: *state-extractors set-conflict-wl-heur-def Let-def*
split: *isasat-int-splits*)

sepref-def *set-conflict-wl-heur-fast-code*
is ‹*uncurry set-conflict-wl-heur*›
:: ‹[λ(*C*, *S*). *length* (*get-clauses-wl-heur S*) ≤ *snat64-max*]$_a$
sint64-nat-assn^k *$*_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
supply [[*goals-limit*=*1*]]
unfolding *set-conflict-wl-heur-alt-def*
apply (*annot-unat-const* ‹*TYPE* (*32*)›)
by *sepref*

sepref-register *update-blit-wl-heur clause-not-marked-to-delete-heur*

lemma *unit-propagation-inner-loop-wl-loop-D-heur-inv0D*:
‹*unit-propagation-inner-loop-wl-loop-D-heur-inv0 L* (*j*, *w*, *S0*) ⟹
j ≤ *length* (*get-clauses-wl-heur S0*) − *MIN-HEADER-SIZE* ∧
w ≤ *length* (*get-clauses-wl-heur S0*) − *MIN-HEADER-SIZE*›
unfolding *unit-propagation-inner-loop-wl-loop-D-heur-inv0-def prod.case*
unit-propagation-inner-loop-wl-loop-inv-def unit-propagation-inner-loop-l-inv-def
apply *normalize-goal*+
by (*simp only*: *twl-st-l twl-st twl-st-wl*
$\mathcal{L}_{all}$*-all-atms-all-lits*) *linarith*

sepref-def *pos-of-watched-heur-impl*
is ‹*uncurry2 pos-of-watched-heur*›
:: ‹*isasat-bounded-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} *$*_a$ *unat-lit-assn*^{k} →$_a$ *sint64-nat-assn*›
supply [[*goals-limit*=*1*]]
unfolding *pos-of-watched-heur-def*
apply (*annot-snat-const* ‹*TYPE* (*64*)›)
by *sepref*

sepref-def *unit-propagation-inner-loop-body-wl-fast-heur-code*
is ‹*uncurry3 unit-propagation-inner-loop-body-wl-heur*›
:: ‹[λ((*L*, *w*), *S*). *length* (*get-clauses-wl-heur S*) ≤ *snat64-max*]$_a$
unat-lit-assn^k *$*_a$ *sint64-nat-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} *$*_a$ *isasat-bounded-assn*^{d} →
sint64-nat-assn ×$_a$ *sint64-nat-assn* ×$_a$ *isasat-bounded-assn*›
supply [[*goals-limit*=*1*]]
if-splits[*split*] *snat64-max-le-max-snat64*[*intro*] *unit-propagation-inner-loop-wl-loop-D-heur-inv0D*[*dest*!]
unfolding *unit-propagation-inner-loop-body-wl-heur-def PR-CONST-def*
unfolding *fmap-rll-def*[*symmetric*]
unfolding *option.case-eq-if is-None-alt*[*symmetric*]

SET-FALSE-def[*symmetric*] *SET-TRUE-def*[*symmetric*] *tri-bool-eq-def*[*symmetric*]
apply (*annot-snat-const* ‹*TYPE* (*64*)›)
by *sepref*

lemmas [*llvm-inline*] =
other-watched-wl-heur-impl-def
pos-of-watched-heur-impl-def
propagate-lit-wl-heur-def
keep-watch-heur-fast-code-def
nat-of-lit-rel-impl-def

experiment begin

export-llvm
isa-save-pos-fast-code
watched-by-app-heur-fast-code
isa-find-unwatched-between-fast-code
find-unwatched-wl-st-heur-fast-code
update-clause-wl-fast-code
propagate-lit-wl-fast-code
propagate-lit-wl-bin-fast-code
status-neq-impl
update-blit-wl-heur-fast-code
keep-watch-heur-fast-code
set-conflict-wl-heur-fast-code
unit-propagation-inner-loop-body-wl-fast-heur-code

end

end
theory *IsaSAT-Proofs*
imports *IsaSAT-Setup*
begin

14.4 DRAT proof generation

We do not prove anything about the proof we generate. In particular, it could be incorrect, because some clauses is not printed.

definition *log-literal* :: ‹*nat literal* ⇒ *unit*› **where**
‹*log-literal* - = ()›

definition *log-start-new-clause* :: ‹*nat* ⇒ *unit*› **where**
‹*log-start-new-clause* - = ()›

definition *log-start-del-clause* :: ‹*nat* ⇒ *unit*› **where**
‹*log-start-del-clause* - = ()›

definition *log-end-clause* :: ‹*nat* ⇒ *unit*› **where**
‹*log-end-clause* - = ()›

definition *log-clause-heur* :: ‹*isasat* ⇒ *nat* ⇒ *unit nres*› **where**
‹*log-clause-heur S C* = *do* {
i ← *mop-arena-length-st S C*;
ASSERT (*i* ≤ *length* (*get-clauses-wl-heur S*));

```
    - ← WHILE_T (λj. j < i)
      (λj. do{ ASSERT (j<i);
            L ← mop-access-lit-in-clauses-heur S C j;
            let - = log-literal L;
            RETURN (j+1)
       })
      0;
    RETURN ()
}›

definition log-clause2 :: ‹nat twl-st-wl ⇒ nat ⇒ unit nres› where
  ‹log-clause2 S C = do {
     ASSERT (C ∈# dom-m (get-clauses-wl S));
     let i = length (get-clauses-wl S ∝ C);
     - ← WHILE_T (λj. j < i)
      (λj. do{
           ASSERT (j < i);
           let L = get-clauses-wl S ∝ C ! j;
           let - = log-literal L;
           RETURN (j+1)
       })
      0;
    RETURN ()
   }›

definition log-clause :: ‹'v twl-st-wl ⇒ nat ⇒ unit› where ‹log-clause S - = ()›

lemma log-clause2-log-clause:
  ‹(uncurry log-clause2, uncurry (RETURN oo log-clause)) ∈
  [λ(S,C). C ∈# dom-m (get-clauses-wl S)]_f Id ×_r nat-rel → ⟨unit-rel⟩nres-rel›
proof −
  show ?thesis
    unfolding log-clause-def log-clause2-def comp-def uncurry-def mop-arena-length-st-def
    apply (intro frefI nres-relI)
    subgoal for x y
   apply (refine-vcg WHILET-rule[where I = ‹λ-. True› and R = ‹measure (λi. length (get-clauses-wl
(fst y) ∝ snd y) − i)›])
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    done
  done
qed

lemma log-clause-heur-log-clause2:
  assumes ‹(S,T) ∈ twl-st-heur› ‹(C,C') ∈ nat-rel›
  shows ‹log-clause-heur S C ≤⇓unit-rel (log-clause2 T C')›
proof −
  have [refine0]: ‹(0,0)∈nat-rel›
    by auto
  have length: ‹⇓ nat-rel ((RETURN ∘ (λc. length (get-clauses-wl T ∝ c))) C') ≤ SPEC (λc. (c, length
(get-clauses-wl T ∝ C')) ∈ {(a,b). a=b ∧ a = length (get-clauses-wl T ∝ C)})›
    by (use assms in auto)
  show ?thesis
```

unfolding *log-clause-heur-def log-clause2-def comp-def uncurry-def mop-arena-length-st-def mop-access-lit-in-clauses-heur-def*
apply (*refine-vcg mop-arena-lit*[**where** *vdom* = ‹*set* (*get-vdom S*)› **and** *N* = ‹*get-clauses-wl T*›, *THEN order-trans*]
mop-arena-length[**where** *vdom* = ‹*set* (*get-vdom S*)›, *THEN fref-to-Down-curry*, *THEN order-trans*, *unfolded prod.simps*])
apply *assumption*
subgoal using *assms* **by** (*auto simp*: *twl-st-heur-def*)
apply (*rule length*)
subgoal by (*use assms* **in** ‹*auto simp*: *twl-st-heur-def dest*: *arena-lifting*(*10*)›)
subgoal by *auto*
subgoal using *assms* **by** (*auto simp*: *twl-st-heur-def*)
apply *assumption*
subgoal by (*use assms* **in** *auto*)
apply (*rule refl*)
subgoal by *auto*
by *auto*
qed

definition *log-new-clause-heur* :: ‹*isasat* ⇒ *nat* ⇒ *unit nres*› **where**
‹*log-new-clause-heur S C* = *do* {
let - = *log-start-new-clause 0*;
log-clause-heur S C;
let - = *log-end-clause 0*;
RETURN ()
}›

lemma *log-new-clause-heur-alt-def*:
‹*log-new-clause-heur* = *log-clause-heur*›
by (*auto intro*!: *ext simp*: *log-new-clause-heur-def log-clause-heur-def*)

definition *log-del-clause-heur* :: ‹*isasat* ⇒ *nat* ⇒ *unit nres*› **where**
‹*log-del-clause-heur S C* = *do* {
let - = *log-start-del-clause 0*;
log-clause-heur S C;
let - = *log-end-clause 0*;
RETURN ()
}›

lemma *log-del-clause-heur-alt-def*:
‹*log-del-clause-heur* = *log-clause-heur*›
by (*auto intro*!: *ext simp*: *log-del-clause-heur-def log-clause-heur-def*)

lemma *log-new-clause-heur-log-clause*:
assumes ‹(*S*,*T*) ∈ *twl-st-heur*› ‹(*C*,*C'*) ∈ *nat-rel*› ‹*C* ∈# *dom-m* (*get-clauses-wl T*)›
shows ‹*log-new-clause-heur S C* ≤ *SPEC* (λ*c*. (*c*, *log-clause T C'*) ∈ *unit-rel*)›
unfolding *log-new-clause-heur-alt-def*
apply (*rule log-clause-heur-log-clause2*[*THEN order-trans*, *OF assms*(*1*,*2*)])
apply (*rule order-trans*)
apply (*rule ref-two-step'*)
apply (*rule log-clause2-log-clause*[*THEN fref-to-Down-curry*])
using *assms* **by** *auto*

definition *log-unit-clause* **where**
‹*log-unit-clause L* =

```
    (let - = log-start-new-clause 0;
      - = log-literal L;
      - = log-end-clause 0 in
     ()
  )›

definition log-del-binary-clause where
  ‹log-del-binary-clause L L' =
    (let - = log-start-del-clause 0;
      - = log-literal L;
      - = log-literal L';
      - = log-end-clause 0 in
     ()
  )›
```

For removing unit literals, we cheat as usual: we signal to the C side which literals are in and flush the clause if need be (without effect on the Isabelle side, because we neither want nor care about the proofs).

```
definition mark-literal-for-unit-deletion :: ‹nat literal ⇒ unit› where
  ‹mark-literal-for-unit-deletion - = ()›

definition mark-clause-for-unit-as-unchanged :: ‹nat ⇒ unit› where
  ‹mark-clause-for-unit-as-unchanged - = ()›

definition mark-clause-for-unit-as-changed :: ‹nat ⇒ unit› where
  ‹mark-clause-for-unit-as-changed - = ()›

end
theory IsaSAT-Backtrack-Defs
  imports IsaSAT-Setup IsaSAT-VMTF IsaSAT-Rephase-State IsaSAT-LBD
    IsaSAT-Proofs
    IsaSAT-Bump-Heuristics
begin

hide-const (open) NEMonad.ASSERT NEMonad.RETURN
```

Chapter 15

Backtrack

The backtrack function is highly complicated and tricky to maintain.

15.1 Backtrack with direct extraction of literal if highest level

Empty conflict **definition** *empty-conflict-and-extract-clause-heur-inv* **where**
‹*empty-conflict-and-extract-clause-heur-inv M outl* =
(λ(*E*, *C*, *i*). *mset* (*take i C*) = *mset* (*take i outl*) ∧
length C = *length outl* ∧ *C* ! *0* = *outl* ! *0* ∧ *i* ≥ *1* ∧ *i* ≤ *length outl* ∧
(*1* < *length* (*take i C*) ⟶
highest-lit M (*mset* (*tl* (*take i C*)))
(*Some* (*C* ! *1*, *get-level M* (*C* ! *1*)))))›

definition *isa-empty-conflict-and-extract-clause-heur* ::
‹*trail-pol* ⇒ *lookup-clause-rel* ⇒ *nat literal list* ⇒ (- × *nat literal list* × *nat*) *nres*›
where
‹*isa-empty-conflict-and-extract-clause-heur M D outl* = *do* {
let C = *replicate* (*length outl*) (*outl*!*0*);
(*D*, *C*, -) ← *WHILE*$_T$
(λ(*D*, *C*, *i*). *i* < *length-uint32-nat outl*)
(λ(*D*, *C*, *i*). *do* {
ASSERT(*i* < *length outl*);
ASSERT(*i* < *length C*);
ASSERT(*lookup-conflict-remove1-pre* (*outl* ! *i*, *D*));
let D = *lookup-conflict-remove1* (*outl* ! *i*) *D*;
let C = *C*[*i* := *outl* ! *i*];
ASSERT(*get-level-pol-pre* (*M*, *C*!*i*));
ASSERT(*get-level-pol-pre* (*M*, *C*!*1*));
ASSERT(*1* < *length C*);
let C = (*if get-level-pol M* (*C*!*i*) > *get-level-pol M* (*C*!*1*) *then swap C 1 i else C*);
ASSERT(*i*+*1* ≤ *unat32-max*);
RETURN (*D*, *C*, *i*+*1*)
})
(*D*, *C*, *1*);
ASSERT(*length outl* ≠ *1* ⟶ *length C* > *1*);
ASSERT(*length outl* ≠ *1* ⟶ *get-level-pol-pre* (*M*, *C*!*1*));
RETURN ((*True*, *D*), *C*, *if length outl* = *1 then 0 else get-level-pol M* (*C*!*1*))
}›

definition *empty-cach-ref-set-inv* **where**

```
‹empty-cach-ref-set-inv cach0 support =
  (λ(i, cach). length cach = length cach0 ∧
      (∀ L ∈ set (drop i support). L < length cach) ∧
      (∀ L ∈ set (take i support).  cach ! L = SEEN-UNKNOWN) ∧
      (∀ L < length cach. cach ! L ≠ SEEN-UNKNOWN ⟶ L ∈ set (drop i support)))›

definition empty-cach-ref-set where
  ‹empty-cach-ref-set = (λ(cach0, support). do {
    let n = length support;
    ASSERT(n ≤ Suc (unat32-max div 2));
    (-, cach) ← WHILE_T^empty-cach-ref-set-inv cach0 support
      (λ(i, cach). i < length support)
      (λ(i, cach). do {
         ASSERT(i < length support);
         ASSERT(support ! i < length cach);
         RETURN(i+1, cach[support ! i := SEEN-UNKNOWN])
      })
     (0, cach0);
    RETURN (cach, emptied-list support)
  })›
```

Minimisation of the conflict

```
definition empty-cach-ref-pre where
  ‹empty-cach-ref-pre = (λ(cach :: minimize-status list, supp :: nat list).
      (∀ L∈set supp. L < length cach) ∧
      length supp ≤ Suc (unat32-max div 2) ∧
      (∀ L<length cach. cach ! L ≠ SEEN-UNKNOWN ⟶ L ∈ set supp))›

definition (in −) empty-cach-ref where
  ‹empty-cach-ref = (λ(cach, support). (replicate (length cach) SEEN-UNKNOWN, []))›

definition extract-shorter-conflict-list-heur-st
  :: ‹isasat ⇒ (isasat × - × -) nres›
  where
    ‹extract-shorter-conflict-list-heur-st = (λS. do {
     let M = get-trail-wl-heur S;
     let N = get-clauses-wl-heur S;
     let outl = get-outlearned-heur S;
     let vm = get-vmtf-heur S;
     let lbd = get-lbd S;
     let cach = get-conflict-cach S;
     let (-, D) = get-conflict-wl-heur S;
     lbd ← mark-lbd-from-list-heur M outl lbd;
     ASSERT(fst M ≠ []);
     let K = lit-of-last-trail-pol M;
     ASSERT(0 < length outl);
     ASSERT(lookup-conflict-remove1-pre (−K, D));
     let D = lookup-conflict-remove1 (−K) D;
     let outl = outl[0 := −K];
     vm ← isa-vmtf-mark-to-rescore-also-reasons M N outl (−K) vm;
     (D, cach, outl) ← isa-minimize-and-extract-highest-lookup-conflict M N D cach lbd outl;
     ASSERT(empty-cach-ref-pre cach);
     let cach = empty-cach-ref cach;
     ASSERT(outl ≠ [] ∧ length outl ≤ unat32-max);
     (D, C, n) ← isa-empty-conflict-and-extract-clause-heur M D outl;
     let S = set-outl-wl-heur (take 1 outl) S;
```

let S = set-conflict-wl-heur D S; let S = set-vmtf-wl-heur vm S;
let S = set-ccach-max-wl-heur cach S; let S = set-lbd-wl-heur lbd S;
RETURN (S, n, C)
})›

definition *update-propagation-heuristics-stats* **where**
‹*update-propagation-heuristics-stats = (λglue (fema, sema, res-info, wasted, phasing, reluctant, fullyproped, s).*
(ema-update glue fema, ema-update glue sema,
incr-conflict-count-since-last-restart res-info, wasted,phasing, reluctant, False, s))›

definition *update-propagation-heuristics* **where**
‹*update-propagation-heuristics glue = Restart-Heuristics o update-propagation-heuristics-stats glue o get-content*›

definition *propagate-bt-wl-D-heur*
:: ‹*nat literal ⇒ nat clause-l ⇒ isasat ⇒ isasat nres*› **where**
‹*propagate-bt-wl-D-heur = (λL C S₀. do {*
let M = get-trail-wl-heur S₀;
let vdom = get-aivdom S₀;
let N0 = get-clauses-wl-heur S₀;
let W0 = get-watched-wl-heur S₀;
let lcount = get-learned-count S₀;
let heur = get-heur S₀;
let stats = get-stats-heur S₀;
let lbd = get-lbd S₀;
let vm0 = get-vmtf-heur S₀;
ASSERT(length (get-vdom-aivdom vdom) ≤ length N0);
ASSERT(length (get-avdom-aivdom vdom) ≤ length N0);
ASSERT(nat-of-lit (C!1) < length W0 ∧ nat-of-lit (−L) < length W0);
ASSERT(length C > 1);
let L′ = C!1;
ASSERT(length C ≤ unat32-max div 2 + 1);
vm ← isa-bump-rescore C M vm0;
glue ← get-LBD lbd;
let b = False;
let b′ = (length C = 2);
ASSERT(isasat-fast S₀ ⟶ append-and-length-fast-code-pre ((b, C), N0));
ASSERT(isasat-fast S₀ ⟶ clss-size-lcount lcount < snat64-max);
(N, i) ← fm-add-new b C N0;
ASSERT(update-lbd-pre ((i, glue), N));
let N = update-lbd-and-mark-used i glue N;
ASSERT(isasat-fast S₀ ⟶ length-ll W0 (nat-of-lit (−L)) < snat64-max);
let W = W0[nat-of-lit (− L) := W0 ! nat-of-lit (− L) @ [(i, L′, b′)]];
ASSERT(isasat-fast S₀ ⟶ length-ll W (nat-of-lit L′) < snat64-max);
let W = W[nat-of-lit L′ := W!nat-of-lit L′ @ [(i, −L, b′)]];
lbd ← lbd-empty lbd;
j ← mop-isa-length-trail M;
ASSERT(i ≠ DECISION-REASON);
ASSERT(cons-trail-Propagated-tr-pre ((−L, i), M));
M ← cons-trail-Propagated-tr (− L) i M;
vm ← isa-bump-heur-flush M vm;
heur ← mop-save-phase-heur (atm-of L′) (is-neg L′) heur;
let S = set-watched-wl-heur W S₀;
let S = set-learned-count-wl-heur (clss-size-incr-lcount lcount) S;

```
    let S = set-aivdom-wl-heur (add-learned-clause-aivdom i vdom) S;
   let S = set-heur-wl-heur (unset-fully-propagated-heur (heuristic-reluctant-tick (update-propagation-heuristics
glue heur))) S;
    let S = set-stats-wl-heur (add-lbd (of-nat glue) stats) S;
    let S = set-trail-wl-heur M S;
    let S = set-clauses-wl-heur N S;
    let S = set-literals-to-update-wl-heur j S;
    let S = set-count-max-wl-heur 0 S;
    let S = set-vmtf-wl-heur vm S;
    let S = set-lbd-wl-heur lbd S;
    - ← log-new-clause-heur S i;
    S ← maybe-mark-added-clause-heur2 S i;
    RETURN (S)
  })›

definition propagate-unit-bt-wl-D-int
  :: ‹nat literal ⇒ isasat ⇒ isasat nres›
  where
   ‹propagate-unit-bt-wl-D-int = (λL S. do {
    let M = get-trail-wl-heur S;
    let vdom = get-aivdom S;
    let N = get-clauses-wl-heur S;
    let W0 = get-watched-wl-heur S;
    let lcount = get-learned-count S;
    let heur = get-heur S;
    let stats = get-stats-heur S;
    let lbd = get-lbd S;
    let vm0 = get-vmtf-heur S;
    vm ← isa-bump-heur-flush M vm0;
    glue ← get-LBD lbd;
    lbd ← lbd-empty lbd;
    j ← mop-isa-length-trail M;
    ASSERT(0 ≠ DECISION-REASON);
    ASSERT(cons-trail-Propagated-tr-pre ((− L, 0::nat), M));
    M ← cons-trail-Propagated-tr (− L) 0 M;
    let stats = incr-units-since-last-GC (incr-uset stats);
    let S = set-stats-wl-heur stats S;
    let S = set-trail-wl-heur M S;
    let S = set-vmtf-wl-heur vm S;
    let S = set-lbd-wl-heur lbd S;
    let S = set-literals-to-update-wl-heur j S;
   let S = set-heur-wl-heur (unset-fully-propagated-heur (heuristic-reluctant-tick (update-propagation-heuristics
glue heur))) S;
    let S = set-learned-count-wl-heur (clss-size-incr-lcountUEk lcount) S;
    RETURN S})›

Full function definition backtrack-wl-D-heur-inv where
  ‹backtrack-wl-D-heur-inv S ⟷ (∃ S'. (S, S') ∈ twl-st-heur-conflict-ana ∧ backtrack-wl-inv S')›

definition backtrack-wl-D-nlit-heur
  :: ‹isasat ⇒ isasat nres›
  where
   ‹backtrack-wl-D-nlit-heur S₀ =
   do {
    ASSERT(backtrack-wl-D-heur-inv S₀);
    ASSERT(fst (get-trail-wl-heur S₀) ≠ []);
```

```
    L ← lit-of-hd-trail-st-heur S₀;
    (S, n, C) ← extract-shorter-conflict-list-heur-st S₀;
    ASSERT(get-clauses-wl-heur S = get-clauses-wl-heur S₀ ∧
        get-learned-count S = get-learned-count S₀);
    S ← find-decomp-wl-st-int n S;

    ASSERT(get-clauses-wl-heur S = get-clauses-wl-heur S₀ ∧
        get-learned-count S = get-learned-count S₀);
    if size C > 1
    then do {
      S ← propagate-bt-wl-D-heur L C S;
      save-phase-st S
    }
    else do {
      propagate-unit-bt-wl-D-int L S
    }
  }›


lemma empty-cach-ref-set-empty-cach-ref:
  ‹(empty-cach-ref-set, RETURN o empty-cach-ref) ∈
    [λ(cach, supp). (∀ L ∈ set supp. L < length cach) ∧ length supp ≤ Suc (unat32-max div 2) ∧
      (∀ L < length cach. cach ! L ≠ SEEN-UNKNOWN ⟶ L ∈ set supp)]_f
    Id → ⟨Id⟩ nres-rel›
proof −
  have H: ‹WHILE_T^empty-cach-ref-set-inv cach0 support' (λ(i, cach). i < length support')
      (λ(i, cach).
          ASSERT (i < length support') ≫=
          (λ-. ASSERT (support' ! i < length cach) ≫=
          (λ-. RETURN (i + 1, cach[support' ! i := SEEN-UNKNOWN]))))
      (0, cach0) ≫=
    (λ(-, cach). RETURN (cach, emptied-list support'))
    ≤ ⇓ Id (RETURN (replicate (length cach0) SEEN-UNKNOWN, []))›
   if
    ‹∀ L∈set support'. L < length cach0› and
    ‹∀ L<length cach0. cach0 ! L ≠ SEEN-UNKNOWN ⟶ L ∈ set support'›
   for cach support cach0 support'
  proof −
    have init: ‹empty-cach-ref-set-inv cach0 support' (0, cach0)›
      using that unfolding empty-cach-ref-set-inv-def
      by auto
    have valid-length:
      ‹empty-cach-ref-set-inv cach0 support' s ⟹ case s of (i, cach) ⇒ i < length support' ⟹
          s = (cach', sup') ⟹ support' ! cach' < length sup'› for s cach' sup'
      using that unfolding empty-cach-ref-set-inv-def
      by auto
   have set-next: ‹empty-cach-ref-set-inv cach0 support' (i + 1, cach'[support' ! i := SEEN-UNKNOWN])›
      if
        inv: ‹empty-cach-ref-set-inv cach0 support' s› and
        cond: ‹case s of (i, cach) ⇒ i < length support'› and
        s: ‹s = (i, cach')› and
        valid[simp]: ‹support' ! i < length cach'›
      for s i cach'
    proof −
      have
        le-cach-cach0: ‹length cach' = length cach0› and
```

```
      le-length: ‹∀ L∈set (drop i support'). L < length cach'› and
      UNKNOWN: ‹∀ L∈set (take i support'). cach' ! L = SEEN-UNKNOWN› and
      support: ‹∀ L<length cach'. cach' ! L ≠ SEEN-UNKNOWN ⟶ L ∈ set (drop i support')› and
      [simp]: ‹i < length support'›
      using inv cond unfolding empty-cach-ref-set-inv-def s prod.case
      by auto

    show ?thesis
      unfolding empty-cach-ref-set-inv-def
      unfolding prod.case
      apply (intro conjI)
      subgoal by (simp add: le-cach-cach0)
      subgoal using le-length by (simp add: Cons-nth-drop-Suc[symmetric])
      subgoal using UNKNOWN by (auto simp add: take-Suc-conv-app-nth)
      subgoal using support by (auto simp add: Cons-nth-drop-Suc[symmetric])
      done
  qed
  have final: ‹((cach', emptied-list support'), replicate (length cach0) SEEN-UNKNOWN, []) ∈ Id›
    if
      inv: ‹empty-cach-ref-set-inv cach0 support' s› and
      cond: ‹¬ (case s of (i, cach) ⇒ i < length support')› and
      s: ‹s = (i, cach')›
    for s cach' i
  proof −
    have
      le-cach-cach0: ‹length cach' = length cach0› and
      le-length: ‹∀ L∈set (drop i support'). L < length cach'› and
      UNKNOWN: ‹∀ L∈set (take i support'). cach' ! L = SEEN-UNKNOWN› and
      support: ‹∀ L<length cach'. cach' ! L ≠ SEEN-UNKNOWN ⟶ L ∈ set (drop i support')› and
      i: ‹¬i < length support'›
      using inv cond unfolding empty-cach-ref-set-inv-def s prod.case
      by auto
    have ‹∀ L<length cach'. cach' ! L  = SEEN-UNKNOWN›
      using support i by auto
    then have [dest]: ‹L ∈ set cach' ⟹ L = SEEN-UNKNOWN› for L
      by (metis in-set-conv-nth)
    then have [dest]: ‹SEEN-UNKNOWN ∉ set cach' ⟹ cach0 = [] ∧ cach' = []›
      using le-cach-cach0 by (cases cach') auto
    show ?thesis
      by (auto simp: emptied-list-def list-eq-replicate-iff le-cach-cach0)
  qed
  show ?thesis
    unfolding conc-Id id-def
    apply (refine-vcg WHILEIT-rule[where R = ‹measure (λ(i, -). length support' − i)›])
    subgoal by auto
    subgoal by (rule init)
    subgoal by auto
    subgoal by (rule valid-length)
    subgoal by (rule set-next)
    subgoal by auto
    subgoal using final by simp
    done
qed
show ?thesis
  unfolding empty-cach-ref-set-def empty-cach-ref-def Let-def comp-def
  by (intro frefI nres-relI ASSERT-leI) (clarify intro!: H ASSERT-leI)
```

qed

end
theory *IsaSAT-Backtrack*
  imports *IsaSAT-Backtrack-Defs*
begin

hide-const (**open**) *NEMonad.ASSERT NEMonad.RETURN NEMonad.SPEC*

Chapter 16

Backtrack

The backtrack function is highly complicated and tricky to maintain.

16.1 Backtrack with direct extraction of literal if highest level

Empty conflict **definition** (**in** −) *empty-conflict-and-extract-clause*
:: ‹(*nat*,*nat*) *ann-lits* ⇒ *nat clause* ⇒ *nat clause-l* ⇒
(*nat clause option* × *nat clause-l* × *nat*) *nres*›
where
‹*empty-conflict-and-extract-clause M D outl* =
SPEC(λ(*D*, *C*, *n*). *D* = *None* ∧ *mset C* = *mset outl* ∧ *C*!*0* = *outl*!*0* ∧
(*length C* > *1* ⟶ *highest-lit M* (*mset* (*tl C*)) (*Some* (*C*!*1*, *get-level M* (*C*!*1*)))) ∧
(*length C* > *1* ⟶ *n* = *get-level M* (*C*!*1*)) ∧
(*length C* = *1* ⟶ *n* = *0*)
)›

definition *empty-conflict-and-extract-clause-heur-inv* **where**
‹*empty-conflict-and-extract-clause-heur-inv M outl* =
(λ(*E*, *C*, *i*). *mset* (*take i C*) = *mset* (*take i outl*) ∧
length C = *length outl* ∧ *C* ! *0* = *outl* ! *0* ∧ *i* ≥ *1* ∧ *i* ≤ *length outl* ∧
(*1* < *length* (*take i C*) ⟶
highest-lit M (*mset* (*tl* (*take i C*)))
(*Some* (*C* ! *1*, *get-level M* (*C* ! *1*)))))›

definition *empty-conflict-and-extract-clause-heur* ::
nat multiset ⇒ (*nat*, *nat*) *ann-lits*
⇒ *lookup-clause-rel*
⇒ *nat literal list* ⇒ (- × *nat literal list* × *nat*) *nres*
where
‹*empty-conflict-and-extract-clause-heur* 𝒜 *M D outl* = *do* {
let C = *replicate* (*length outl*) (*outl*!*0*);
(*D*, *C*, -) ← *WHILE*$_T$*empty-conflict-and-extract-clause-heur-inv M outl*
(λ(*D*, *C*, *i*). *i* < *length-uint32-nat outl*)
(λ(*D*, *C*, *i*). *do* {
ASSERT(*i* < *length outl*);
ASSERT(*i* < *length C*);
ASSERT(*lookup-conflict-remove1-pre* (*outl* ! *i*, *D*));
let D = *lookup-conflict-remove1* (*outl* ! *i*) *D*;
let C = *C*[*i* := *outl* ! *i*];
ASSERT(*C*!*i* ∈# $\mathcal{L}_{all}$ 𝒜 ∧ *C*!*1* ∈# $\mathcal{L}_{all}$ 𝒜 ∧ *1* < *length C*);
let C = (*if get-level M* (*C*!*i*) > *get-level M* (*C*!*1*) *then swap C 1 i else C*);

```
        ASSERT(i+1 ≤ unat32-max);
        RETURN (D, C, i+1)
      })
     (D, C, 1);
   ASSERT(length outl ≠ 1 ⟶ length C > 1);
   ASSERT(length outl ≠ 1 ⟶ C!1 ∈# \<L>_all \<A>);
   RETURN ((True, D), C, if length outl = 1 then 0 else get-level M (C!1))
 }›

lemma empty-conflict-and-extract-clause-heur-empty-conflict-and-extract-clause:
 assumes
   D: ‹D = mset (tl outl)› and
   outl: ‹outl ≠ []› and
   dist: ‹distinct outl› and
   lits: ‹literals-are-in-\<L>_in \<A> (mset outl)› and
   DD': ‹(D', D) ∈ lookup-clause-rel \<A>› and
   consistent: ‹¬ tautology (mset outl)› and
   bounded: ‹isasat-input-bounded \<A>›
 shows
   ‹empty-conflict-and-extract-clause-heur \<A> M D' outl ≤ ⇓ (option-lookup-clause-rel \<A> ×_r Id ×_r Id)
     (empty-conflict-and-extract-clause M D outl)›
proof −
 have size-out: ‹size (mset outl) ≤ 1 + unat32-max div 2›
   using simple-clss-size-upper-div2[OF bounded lits - consistent]
     ‹distinct outl› by auto
 have empty-conflict-and-extract-clause-alt-def:
   ‹empty-conflict-and-extract-clause M D outl = do {
     (D', outl') ← SPEC (λ(E, F). E = {#} ∧ mset F = D);
     SPEC
       (λ(D, C, n).
           D = None ∧
           mset C = mset outl ∧
           C ! 0 = outl ! 0 ∧
           (1 < length C ⟶
             highest-lit M (mset (tl C)) (Some (C ! 1, get-level M (C ! 1)))) ∧
           (1 < length C ⟶ n = get-level M (C ! 1)) ∧ (length C = 1 ⟶ n = 0))
   }› for M D outl
   unfolding empty-conflict-and-extract-clause-def RES-RES2-RETURN-RES
   by (auto simp: ex-mset)
 define I where
   ‹I ≡ λ(E, C, i). mset (take i C) = mset (take i outl) ∧
     (E, D − mset (take i outl)) ∈ lookup-clause-rel \<A> ∧
           length C = length outl ∧ C ! 0 = outl ! 0 ∧ i ≥ 1 ∧ i ≤ length outl ∧
           (1 < length (take i C) ⟶
                highest-lit M (mset (tl (take i C)))
                 (Some (C ! 1, get-level M (C ! 1))))›
 have I0: ‹I (D', replicate (length outl) (outl ! 0), 1)›
   using assms by (cases outl) (auto simp: I-def)

 have [simp]: ‹ba ≥ 1 ⟹ mset (tl outl) − mset (take ba outl) = mset ((drop ba outl))›
   for ba
   apply (subst append-take-drop-id[of ‹ba − 1›, symmetric])
   using dist
   unfolding mset-append
   by (cases outl; cases ba)
     (auto simp: take-tl drop-Suc[symmetric] remove-1-mset-id-iff-notin dest: in-set-dropD)
```

have *empty-conflict-and-extract-clause-heur-inv*:
 ‹*empty-conflict-and-extract-clause-heur-inv M outl*
 (*D'*, *replicate* (*length outl*) (*outl* ! *0*), *1*)›
 using *assms*
 unfolding *empty-conflict-and-extract-clause-heur-inv-def*
 by (*cases outl*) *auto*
have *I0*: ‹*I* (*D'*, *replicate* (*length outl*) (*outl* ! *0*), *1*)›
 using *assms*
 unfolding *I-def*
 by (*cases outl*) *auto*
have
 C1-L: ‹*aa*[*ba* := *outl* ! *ba*] ! *1* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› (**is** *?A1inL*) **and**
 ba-le: ‹*ba* + *1* ≤ *unat32-max*› (**is** *?ba-le*) **and**
 I-rec: ‹*I* (*lookup-conflict-remove1* (*outl* ! *ba*) *a*,
 if get-level M (*aa*[*ba* := *outl* ! *ba*] ! *1*)
 < *get-level M* (*aa*[*ba* := *outl* ! *ba*] ! *ba*)
 then swap (*aa*[*ba* := *outl* ! *ba*]) *1 ba*
 else aa[*ba* := *outl* ! *ba*],
 ba + *1*)› (**is** *?I*) **and**
 inv: ‹*empty-conflict-and-extract-clause-heur-inv M outl*
 (*lookup-conflict-remove1* (*outl* ! *ba*) *a*,
 if get-level M (*aa*[*ba* := *outl* ! *ba*] ! *1*)
 < *get-level M* (*aa*[*ba* := *outl* ! *ba*] ! *ba*)
 then swap (*aa*[*ba* := *outl* ! *ba*]) *1 ba*
 else aa[*ba* := *outl* ! *ba*],
 ba + *1*)› (**is** *?inv*)
 if
 ‹*empty-conflict-and-extract-clause-heur-inv M outl s*› **and**
 I: ‹*I s*› **and**
 ‹*case s of* (*D*, *C*, *i*) ⇒ *i* < *length-uint32-nat outl*› **and**
 st:
 ‹*s* = (*a*, *b*)›
 ‹*b* = (*aa*, *ba*)› **and**
 ba-le: ‹*ba* < *length outl*› **and**
 ‹*ba* < *length aa*› **and**
 ‹*lookup-conflict-remove1-pre* (*outl* ! *ba*, *a*)›
 for *s a b aa ba*
proof −
 have
 mset-aa: ‹*mset* (*take ba aa*) = *mset* (*take ba outl*)› **and**
 aD: ‹(*a*, *D* − *mset* (*take ba outl*)) ∈ *lookup-clause-rel* $\mathcal{A}$› **and**
 l-aa-outl: ‹*length aa* = *length outl*› **and**
 aa0: ‹*aa* ! *0* = *outl* ! *0*› **and**
 ba-ge1: ‹*1* ≤ *ba*› **and**
 ba-lt: ‹*ba* ≤ *length outl*› **and**
 highest: ‹*1* < *length* (*take ba aa*) ⟶
 highest-lit M (*mset* (*tl* (*take ba aa*)))
 (*Some* (*aa* ! *1*, *get-level M* (*aa* ! *1*)))›
 using *I* **unfolding** *st I-def prod.case*
 by *auto*
 have *set-aa-outl*: ‹*set* (*take ba aa*) = *set* (*take ba outl*)›
 using *mset-aa* **by** (*blast dest*: *mset-eq-setD*)
 show *?ba-le*
 using *ba-le assms size-out*
 by (*auto simp*: *unat32-max-def*)
 have *ba-ge1-aa-ge*: ‹*ba* > *1* ⟹ *aa* ! *1* ∈ *set* (*take ba aa*)›

using *ba-ge1 ba-le l-aa-outl*
by (*auto simp*: *in-set-take-conv-nth intro*!: *bex-lessI*[*of* - ‹*Suc 0*›])
then have ‹*aa*[*ba* := *outl* ! *ba*] ! *1* ∈ *set outl*›
using *ba-le l-aa-outl ba-ge1*
unfolding *mset-aa in-multiset-in-set*[*symmetric*]
by (*cases* ‹*ba* > *1*›)
(*auto simp*: *mset-aa dest*: *in-set-takeD*)
then show *?A1inL*
using *literals-are-in-*$\mathcal{L}_{in}$*-in-mset-*$\mathcal{L}_{all}$[*OF lits*, *of* ‹*aa*[*ba* := *outl* ! *ba*] ! *1*›] **by** *auto*

define *aa2* **where** ‹*aa2* ≡ *tl* (*tl* (*take ba aa*))›
have *tl-take-nth-con*: ‹*tl* (*take ba aa*) = *aa* ! *Suc 0* # *aa2*› **if** ‹*ba* > *Suc 0*›
using *ba-le ba-ge1 that l-aa-outl* **unfolding** *aa2-def*
by (*cases aa*; *cases* ‹*tl aa*›; *cases ba*; *cases* ‹*ba* − *1*›)
auto
have *no-tauto-nth*: ‹ *i* < *length outl* ⟹ − *outl* ! *ba* = *outl* ! *i* ⟹ *False*› **for** *i*
using *consistent ba-le nth-mem*
by (*force simp*: *tautology-decomp*′ *uminus-lit-swap*)
have *outl-ba--L*: ‹*outl* ! *ba* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
using *ba-le literals-are-in-*$\mathcal{L}_{in}$*-in-mset-*$\mathcal{L}_{all}$[*OF lits*, *of* ‹*outl* ! *ba*›] **by** *auto*
have ‹(*lookup-conflict-remove1* (*outl* ! *ba*) *a*,
remove1-mset (*outl* ! *ba*) (*D* −(*mset* (*take ba outl*)))) ∈ *lookup-clause-rel* $\mathcal{A}$›
by (*rule lookup-conflict-remove1*[*THEN fref-to-Down-unRET-uncurry*])
(*use ba-ge1 ba-le aD outl-ba--L* **in**
‹*auto simp*: *D in-set-drop-conv-nth image-image dest*: *no-tauto-nth*
intro!: *bex-geI*[*of* - *ba*]›)
then have ‹(*lookup-conflict-remove1* (*outl* ! *ba*) *a*,
D − *mset* (*take* (*Suc ba*) *outl*))
∈ *lookup-clause-rel* $\mathcal{A}$›
using *aD ba-le ba-ge1 ba-ge1-aa-ge aa0*
by (*auto simp*: *take-Suc-conv-app-nth*)
moreover have ‹*1* < *length*
(*take* (*ba* + *1*)
(*if get-level M* (*aa*[*ba* := *outl* ! *ba*] ! *1*)
< *get-level M* (*aa*[*ba* := *outl* ! *ba*] ! *ba*)
then swap (*aa*[*ba* := *outl* ! *ba*]) *1 ba*
else aa[*ba* := *outl* ! *ba*])) ⟶
highest-lit M
(*mset*
(*tl* (*take* (*ba* + *1*)
(*if get-level M* (*aa*[*ba* := *outl* ! *ba*] ! *1*)
< *get-level M* (*aa*[*ba* := *outl* ! *ba*] ! *ba*)
then swap (*aa*[*ba* := *outl* ! *ba*]) *1 ba*
else aa[*ba* := *outl* ! *ba*])))))
(*Some*
((*if get-level M* (*aa*[*ba* := *outl* ! *ba*] ! *1*)
< *get-level M* (*aa*[*ba* := *outl* ! *ba*] ! *ba*)
then swap (*aa*[*ba* := *outl* ! *ba*]) *1 ba*
else aa[*ba* := *outl* ! *ba*]) !
1,
get-level M
((*if get-level M* (*aa*[*ba* := *outl* ! *ba*] ! *1*)
< *get-level M* (*aa*[*ba* := *outl* ! *ba*] ! *ba*)
then swap (*aa*[*ba* := *outl* ! *ba*]) *1 ba*
else aa[*ba* := *outl* ! *ba*]) !
1)))›

```
    using highest ba-le ba-ge1
    by (cases ‹ba = Suc 0›)
      (auto simp: highest-lit-def take-Suc-conv-app-nth l-aa-outl
        get-maximum-level-add-mset swap-nth-relevant max-def take-update-swap
        swap-only-first-relevant tl-update-swap mset-update nth-tl
        get-maximum-level-remove-non-max-lvl tl-take-nth-con
        aa2-def[symmetric])
  moreover have ‹mset
    (take (ba + 1)
      (if get-level M (aa[ba := outl ! ba] ! 1)
          < get-level M (aa[ba := outl ! ba] ! ba)
        then swap (aa[ba := outl ! ba]) 1 ba
        else aa[ba := outl ! ba])) =
    mset (take (ba + 1) outl)›
    using ba-le ba-ge1 ba-ge1-aa-ge aa0
    unfolding mset-aa
    by (cases ‹ba = 1›)
      (auto simp: take-Suc-conv-app-nth l-aa-outl
        take-swap-relevant swap-only-first-relevant mset-aa set-aa-outl
        mset-update add-mset-remove-trivial-If)
  ultimately show ?I
    using ba-ge1 ba-le
    unfolding I-def prod.simps
    by (auto simp: l-aa-outl aa0)

  then show ?inv
    unfolding empty-conflict-and-extract-clause-heur-inv-def I-def
    by (auto simp: l-aa-outl aa0)
qed
have mset-tl-out: ‹mset (tl outl) - mset outl = {#}›
  by (cases outl) auto
have H1: ‹WHILE_T^empty-conflict-and-extract-clause-heur-inv M outl
  (λ(D, C, i). i < length-uint32-nat outl)
  (λ(D, C, i). do {
      - ← ASSERT (i < length outl);
      - ← ASSERT (i < length C);
      - ← ASSERT (lookup-conflict-remove1-pre (outl ! i, D));
      - ← ASSERT
          (C[i := outl ! i] ! i ∈# L_all A ∧
           C[i := outl ! i] ! 1 ∈# L_all A ∧
          1 < length (C[i := outl ! i]));
      - ← ASSERT (i + 1 ≤ unat32-max);
      RETURN
       (lookup-conflict-remove1 (outl ! i) D,
        if get-level M (C[i := outl ! i] ! 1)
          < get-level M (C[i := outl ! i] ! i)
        then swap (C[i := outl ! i]) 1 i
        else C[i := outl ! i],
        i + 1)
    })
  (D', replicate (length outl) (outl ! 0), 1)
  ≤ ⇓ {((E, C, n), (E', F')). (E, E') ∈ lookup-clause-rel A ∧ mset C = mset outl ∧
        C ! 0 = outl ! 0 ∧
       (1 < length C ⟶
         highest-lit M (mset (tl C)) (Some (C ! 1, get-level M (C ! 1)))) ∧
       n = length outl ∧
```

```
        I (E, C, n)}
      (SPEC (λ(E, F). E = {#} ∧ mset F = D))›
  unfolding conc-fun-RES
  apply (refine-vcg WHILEIT-rule-stronger-inv-RES[where R = ‹measure (λ(-, -, i). length outl −
i)› and
      I' = ‹I›])
  subgoal by auto
  subgoal by (rule empty-conflict-and-extract-clause-heur-inv)
  subgoal by (rule I0)
  subgoal using assms by (cases outl; auto)
  subgoal
    by (auto simp: I-def)
  subgoal for s a b aa ba
    using literals-are-in-ℒ_in-in-ℒ_all lits
    unfolding lookup-conflict-remove1-pre-def prod.simps
    by (auto simp: I-def empty-conflict-and-extract-clause-heur-inv-def
      lookup-clause-rel-def D atms-of-def)
  subgoal for s a b aa ba
    using literals-are-in-ℒ_in-in-ℒ_all lits
    unfolding lookup-conflict-remove1-pre-def prod.simps
    by (auto simp: I-def empty-conflict-and-extract-clause-heur-inv-def
      lookup-clause-rel-def D atms-of-def)
  subgoal for s a b aa ba
    by (rule C1-L)
  subgoal for s a b aa ba
    using literals-are-in-ℒ_in-in-ℒ_all lits
    unfolding lookup-conflict-remove1-pre-def prod.simps
    by (auto simp: I-def empty-conflict-and-extract-clause-heur-inv-def
      lookup-clause-rel-def D atms-of-def)
  subgoal for s a b aa ba
    by (rule ba-le)
  subgoal
    by (rule inv)
  subgoal
    by (rule I-rec)
  subgoal
    by auto
  subgoal for s
    unfolding prod.simps
    apply (cases s)
    apply clarsimp
    apply (intro conjI)
    subgoal
      by (rule ex-mset)
    subgoal
      using assms
      by (auto simp: empty-conflict-and-extract-clause-heur-inv-def I-def mset-tl-out)
    subgoal
      using assms
      by (auto simp: empty-conflict-and-extract-clause-heur-inv-def I-def mset-tl-out)
    subgoal
      using assms
      by (auto simp: empty-conflict-and-extract-clause-heur-inv-def I-def mset-tl-out)
    subgoal
      using assms
      by (auto simp: empty-conflict-and-extract-clause-heur-inv-def I-def mset-tl-out)
```

```
    subgoal
      using assms
      by (auto simp: empty-conflict-and-extract-clause-heur-inv-def I-def mset-tl-out)
    done
  done
have x1b-lall: ‹x1b ! 1 ∈# \mathcal{L}_{all} \mathcal{A}›
  if
    inv: ‹(x, x')
    ∈ {((E, C, n), E', F').
        (E, E') ∈ lookup-clause-rel \mathcal{A} ∧
        mset C = mset outl ∧
        C ! 0 = outl ! 0 ∧
        (1 < length C ⟶
        highest-lit M (mset (tl C)) (Some (C ! 1, get-level M (C ! 1)))) ∧
          n = length outl ∧
        I (E, C, n)}› and
    ‹x' ∈ {(E, F). E = {#} ∧ mset F = D}› and
    st:
    ‹x' = (x1, x2)›
    ‹x2a = (x1b, x2b)›
    ‹x = (x1a, x2a)› and
    ‹length outl ≠ 1 ⟶ 1 < length x1b› and
    ‹length outl ≠ 1›
  for x x' x1 x2 x1a x2a x1b x2b
proof −
  have
    ‹(x1a, x1) ∈ lookup-clause-rel \mathcal{A}› and
    ‹mset x1b = mset outl› and
    ‹x1b ! 0 = outl ! 0› and
    ‹Suc 0 < length x1b ⟶
    highest-lit M (mset (tl x1b))
      (Some (x1b ! Suc 0, get-level M (x1b ! Suc 0)))› and
    mset-aa: ‹mset (take x2b x1b) = mset (take x2b outl)› and
    ‹(x1a, D − mset (take x2b outl)) ∈ lookup-clause-rel \mathcal{A}› and
    l-aa-outl: ‹length x1b = length outl› and
    ‹x1b ! 0 = outl ! 0› and
    ba-ge1: ‹Suc 0 ≤ x2b› and
    ba-le: ‹x2b ≤ length outl› and
    ‹Suc 0 < length x1b ∧ Suc 0 < x2b ⟶
   highest-lit M (mset (tl (take x2b x1b)))
    (Some (x1b ! Suc 0, get-level M (x1b ! Suc 0)))›
    using inv unfolding st I-def prod.case st
    by auto

  have set-aa-outl: ‹set (take x2b x1b) = set (take x2b outl)›
    using mset-aa by (blast dest: mset-eq-setD)
  have ba-ge1-aa-ge: ‹x2b > 1 ⟹ x1b ! 1 ∈ set (take x2b x1b)›
    using ba-ge1 ba-le l-aa-outl
    by (auto simp: in-set-take-conv-nth intro!: bex-lessI[of - ‹Suc 0›])
  then have ‹x1b ! 1 ∈ set outl›
    using ba-le l-aa-outl ba-ge1 that
    unfolding mset-aa in-multiset-in-set[symmetric]
    by (cases ‹x2b > 1›)
      (auto simp: mset-aa dest: in-set-takeD)
  then show ?thesis
    using literals-are-in-\mathcal{L}_{in}-in-mset-\mathcal{L}_{all}[OF lits, of ‹x1b ! 1›] by auto
```

```
  qed

  show ?thesis
    unfolding empty-conflict-and-extract-clause-heur-def empty-conflict-and-extract-clause-alt-def
      Let-def I-def[symmetric]
    apply (subst empty-conflict-and-extract-clause-alt-def)
    unfolding conc-fun-RES
   apply (refine-vcg WHILEIT-rule-stronger-inv[where R = ‹measure (λ(-, -, i). length outl − i)› and
         I' = ‹I›] H1)
    subgoal using assms by (auto simp: I-def)
    subgoal by (rule x1b-lall)
    subgoal using assms
      by (auto intro!: RETURN-RES-refine simp: option-lookup-clause-rel-def I-def)
    done
qed

lemma isa-empty-conflict-and-extract-clause-heur-empty-conflict-and-extract-clause-heur:
 ‹(uncurry2 isa-empty-conflict-and-extract-clause-heur, uncurry2 (empty-conflict-and-extract-clause-heur
A)) ∈
    trail-pol A ×f Id ×f Id →f ⟨Id⟩nres-rel ›
proof −
  have [refine0]: ‹((x2b, replicate (length x2c) (x2c ! 0), 1), x2,
  replicate (length x2a) (x2a ! 0), 1)
 ∈ Id ×f Id ×f Id›
    if
      ‹(x, y) ∈ trail-pol A ×f Id ×f Id› and     ‹x1 = (x1a, x2)› and
      ‹y = (x1, x2a)› and
      ‹x1b = (x1c, x2b)› and
      ‹x = (x1b, x2c)›
    for x y x1 x1a x2 x2a x1b x1c x2b x2c
    using that by auto

  show ?thesis
    supply [[goals-limit=1]]
     unfolding isa-empty-conflict-and-extract-clause-heur-def empty-conflict-and-extract-clause-heur-def
uncurry-def
    apply (intro frefI nres-relI)
    apply (refine-rcg)
                  apply (assumption+)[5]
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal
      by (rule get-level-pol-pre) auto
    subgoal
      by (rule get-level-pol-pre) auto
    subgoal by auto
    subgoal by auto
    subgoal
      by (auto simp: get-level-get-level-pol[of - - A])
    subgoal by auto
    subgoal
      by (rule get-level-pol-pre) auto
    subgoal by (auto simp: get-level-get-level-pol[of - - A])
    done
```

qed

definition *extract-shorter-conflict-wl-nlit* **where**
‹*extract-shorter-conflict-wl-nlit K M NU D NE UE* =
SPEC(λ*D'*. *D'* ≠ *None* ∧ *the D'* ⊆# *the D* ∧ *K* ∈# *the D'* ∧
mset '# *ran-mf NU* + *NE* + *UE* ⊨*pm the D'*)›

definition *extract-shorter-conflict-wl-nlit-st*
:: ‹'*v twl-st-wl* ⇒ '*v twl-st-wl nres*›
where
‹*extract-shorter-conflict-wl-nlit-st* =
(λ(*M*, *N*, *D*, *NE*, *UE*, *WS*, *Q*). *do* {
let K = −*lit-of* (*hd M*);
D ← *extract-shorter-conflict-wl-nlit K M N D NE UE*;
RETURN (*M*, *N*, *D*, *NE*, *UE*, *WS*, *Q*)})›

definition *empty-lookup-conflict-and-highest*
:: ‹'*v twl-st-wl* ⇒ ('*v twl-st-wl* × *nat*) *nres*›
where
‹*empty-lookup-conflict-and-highest* =
(λ(*M*, *N*, *D*, *NE*, *UE*, *WS*, *Q*). *do* {
let K = −*lit-of* (*hd M*);
let n = *get-maximum-level M* (*remove1-mset K* (*the D*));
RETURN ((*M*, *N*, *D*, *NE*, *UE*, *WS*, *Q*), *n*)})›

definition *extract-shorter-conflict-heur* **where**
‹*extract-shorter-conflict-heur* = (λ*M NU NUE C outl*. *do* {
let K = *lit-of* (*hd M*);
let C = *Some* (*remove1-mset* (−*K*) (*the C*));
C ← *iterate-over-conflict* (−*K*) *M NU NUE* (*the C*);
RETURN (*Some* (*add-mset* (−*K*) *C*))
})›

definition (**in** −) *empty-cach* **where**
‹*empty-cach cach* = (λ-. *SEEN-UNKNOWN*)›

definition *empty-conflict-and-extract-clause-pre*
:: ‹(((*nat*,*nat*) *ann-lits* × *nat clause*) × *nat clause-l*) ⇒ *bool*› **where**
‹*empty-conflict-and-extract-clause-pre* =
(λ((*M*, *D*), *outl*). *D* = *mset* (*tl outl*) ∧ *outl* ≠ [] ∧ *distinct outl* ∧
¬*tautology* (*mset outl*) ∧ *length outl* ≤ *unat32-max*)›

lemma *empty-cach-ref-empty-cach*:
‹*isasat-input-bounded* $\mathcal{A}$ ⟹ (*RETURN o empty-cach-ref*, *RETURN o empty-cach*) ∈ *cach-refinement* $\mathcal{A}$ →$_f$ ⟨*cach-refinement* $\mathcal{A}$⟩ *nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *empty-cach-def empty-cach-ref-def cach-refinement-alt-def cach-refinement-list-def*
map-fun-rel-def intro: *bounded-included-le*)

Minimisation of the conflict **lemma** *the-option-lookup-clause-assn*:
‹(*RETURN o snd*, *RETURN o the*) ∈ [λ*D*. *D* ≠ *None*]$_f$ *option-lookup-clause-rel* $\mathcal{A}$ → ⟨*lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *option-lookup-clause-rel-def*)

lemma *heuristic-rel-stats-update-heuristics-stats*[*intro*!]:
‹*heuristic-rel-stats* $\mathcal{A}$ *heur* $\Longrightarrow$ *heuristic-rel-stats* $\mathcal{A}$ (*update-propagation-heuristics-stats glue heur*)›
by (*auto simp*: *heuristic-rel-stats-def phase-save-heur-rel-def phase-saving-def update-propagation-heuristics-stats-def*)

lemma *heuristic-rel-update-heuristics*[*intro*!]:
‹*heuristic-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *heuristic-rel* $\mathcal{A}$ (*update-propagation-heuristics glue heur*)›
by (*auto simp*: *heuristic-rel-def phase-save-heur-rel-def phase-saving-def update-propagation-heuristics-def*)

lemma *valid-arena-update-lbd-and-mark-used*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *i*: ‹*i* $\in\#$ *dom-m N*›
shows ‹*valid-arena* (*update-lbd-and-mark-used i lbd arena*) *N vdom*›
using *assms* **by** (*auto intro*!: *valid-arena-update-lbd valid-arena-mark-used valid-arena-mark-used2 simp*: *update-lbd-and-mark-used-def Let-def*)

definition *remove-last*
:: ‹*nat literal* $\Rightarrow$ *nat clause option* $\Rightarrow$ *nat clause option nres*›
where
‹*remove-last - -* = *SPEC*((=) *None*)›

Full function **lemma** *get-all-ann-decomposition-get-level*:
assumes
L': ‹*L'* = *lit-of* (*hd M'*)› **and**
nd: ‹*no-dup M'*› **and**
decomp: ‹(*Decided K # a, M2*) $\in$ *set* (*get-all-ann-decomposition M'*)› **and**
lev-K: ‹*get-level M' K* = *Suc* (*get-maximum-level M'* (*remove1-mset* (− *L'*) *y*))› **and**
L: ‹*L* $\in\#$ *remove1-mset* (− *lit-of* (*hd M'*)) *y*›
shows ‹*get-level a L* = *get-level M' L*›
proof −
obtain *M3* **where** *M3*: ‹*M'* = *M3* @ *M2* @ *Decided K # a*›
using *decomp* **by** *blast*
from *lev-K* **have** *lev-L*: ‹*get-level M' L* < *get-level M' K*›
using *get-maximum-level-ge-get-level*[*OF L, of M'*] **unfolding** *L'*[*symmetric*] **by** *auto*
have [*simp*]: ‹*get-level* (*M3* @ *M2* @ *Decided K # a*) *K* = *Suc* (*count-decided a*)›
using *nd* **unfolding** *M3* **by** *auto*
have *undef*:‹*undefined-lit* (*M3* @ *M2*) *L*›
using *lev-L get-level-skip-end*[*of* ‹*M3* @ *M2*› *L* ‹*Decided K # a*›] **unfolding** *M3*
by *auto*
then have ‹*atm-of L* $\neq$ *atm-of K*›
using *lev-L* **unfolding** *M3* **by** *auto*
then show *?thesis*
using *undef* **unfolding** *M3* **by** (*auto simp*: *get-level-cons-if*)
qed

definition *del-conflict-wl* :: ‹'*v twl-st-wl* $\Rightarrow$ '*v twl-st-wl*› **where**
‹*del-conflict-wl* = (λ(*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*). (*M*, *N*, *None*, *NE*, *UE*, *Q*, *W*))›

lemma [*simp*]:
‹*get-clauses-wl* (*del-conflict-wl S*) = *get-clauses-wl S*›
by (*cases S*) (*auto simp*: *del-conflict-wl-def*)

lemma *lcount-add-clause*[*simp*]: ‹*i* $\notin\#$ *dom-m N* $\Longrightarrow$
size (*learned-clss-l* (*fmupd i* (*C*, *False*) *N*)) = *Suc* (*size* (*learned-clss-l N*))›
by (*simp add*: *learned-clss-l-mapsto-upd-notin*)

lemma *length-watched-le*:
  assumes
    prop-inv: ‹*correct-watching x1*› **and**
    xb-x'a: ‹(*x1a*, *x1*) ∈ *twl-st-heur-conflict-ana*› **and**
    x2: ‹*x2* ∈# $\mathcal{L}_{all}$ (*all-atms-st x1*)›
  shows ‹*length* (*watched-by x1 x2*) ≤ *length* (*get-clauses-wl-heur x1a*) − *MIN-HEADER-SIZE*›
proof −
  have ‹*correct-watching x1*›
    using *prop-inv* **unfolding** *unit-propagation-outer-loop-wl-inv-def*
      unit-propagation-outer-loop-wl-inv-def
    by *auto*
  then have *dist*: ‹*distinct-watched* (*watched-by x1 x2*)›
    using *x2* **unfolding** *all-atms-def*[*symmetric*] *all-lits-alt-def*[*symmetric*]
    by (*cases x1*; *auto simp*: $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm correct-watching.simps*
        $\mathcal{L}_{all}$*-all-atms-all-lits*
      simp flip: *all-lits-alt-def2 all-lits-def all-atms-def all-lits-st-alt-def*)
  then have *dist*: ‹*distinct-watched* (*watched-by x1 x2*)›
    using *xb-x'a*
    by (*cases x1*; *auto simp*: $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm correct-watching.simps*)
  have *dist-vdom*: ‹*distinct* (*get-vdom x1a*)›
    using *xb-x'a*
    by (*cases x1*)
      (*auto simp*: *twl-st-heur-conflict-ana-def twl-st-heur'-def aivdom-inv-dec-alt-def*)
  have *x2*: ‹*x2* ∈# $\mathcal{L}_{all}$ (*all-atms-st x1*)›
    using *x2 xb-x'a* **unfolding** *all-atms-def*
    by *auto*

  have
      valid: ‹*valid-arena* (*get-clauses-wl-heur x1a*) (*get-clauses-wl x1*) (*set* (*get-vdom x1a*))›
    using *xb-x'a* **unfolding** *all-atms-def all-lits-def*
    by (*cases x1*)
     (*auto simp*: *twl-st-heur'-def twl-st-heur-conflict-ana-def*)

  have ‹*vdom-m* (*all-atms-st x1*) (*get-watched-wl x1*) (*get-clauses-wl x1*) ⊆ *set* (*get-vdom x1a*)›
    using *xb-x'a*
    by (*cases x1*)
      (*auto simp*: *twl-st-heur-conflict-ana-def twl-st-heur'-def all-atms-def*[*symmetric*])
  then have *subset*: ‹*set* (*map fst* (*watched-by x1 x2*)) ⊆ *set* (*get-vdom x1a*)›
    using *x2* **unfolding** *vdom-m-def*
    by (*cases x1*)
      (*force simp*: *twl-st-heur'-def twl-st-heur-def simp flip*: *all-atms-def*
        dest!: *multi-member-split*)
  have *watched-incl*: ‹*mset* (*map fst* (*watched-by x1 x2*)) ⊆# *mset* (*get-vdom x1a*)›
    by (*rule distinct-subseteq-iff*[*THEN iffD1*])
      (*use dist*[*unfolded distinct-watched-alt-def*] *dist-vdom subset* **in**
        ‹*simp-all flip*: *distinct-mset-mset-distinct*›)
  have *vdom-incl*: ‹*set* (*get-vdom x1a*) ⊆ {*MIN-HEADER-SIZE*..< *length* (*get-clauses-wl-heur x1a*)}›
    using *valid-arena-in-vdom-le-arena*[*OF valid*] *arena-dom-status-iff*[*OF valid*] **by** *auto*

  have ‹*length* (*get-vdom x1a*) ≤ *length* (*get-clauses-wl-heur x1a*) − *MIN-HEADER-SIZE*›
    by (*subst distinct-card*[*OF dist-vdom*, *symmetric*])
      (*use card-mono*[*OF - vdom-incl*] **in** *auto*)
  then show *?thesis*
    using *size-mset-mono*[*OF watched-incl*] *xb-x'a*
    by (*auto intro*!: *order-trans*[*of* ‹*length* (*watched-by x1 x2*)› ‹*length* (*get-vdom x1a*)›])
qed

definition *single-of-mset* **where**
‹*single-of-mset D = SPEC*(λ*L*. *D = mset* [*L*])›

lemma *backtrack-wl-D-nlit-backtrack-wl-D*:
‹(*backtrack-wl-D-nlit-heur*, *backtrack-wl*) ∈
{(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-conflict-ana* ∧ *length* (*get-clauses-wl-heur S*) = *r* ∧
learned-clss-count S ≤ *u*} →*f*
⟨{(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur* ∧ *length* (*get-clauses-wl-heur S*) ≤ *MAX-HEADER-SIZE+1* + *r* + *unat32-max div 2* ∧
learned-clss-count S ≤ *Suc u*}⟩*nres-rel*›
(**is** ‹- ∈ *?R* →*f* ⟨*?S*⟩*nres-rel*›)
proof −
have *backtrack-wl-D-nlit-heur-alt-def*: ‹*backtrack-wl-D-nlit-heur S*$_0$ =
do {
ASSERT(*backtrack-wl-D-heur-inv S*$_0$);
ASSERT(*fst* (*get-trail-wl-heur S*$_0$) ≠ []);
L ← *lit-of-hd-trail-st-heur S*$_0$;
(*S*, *n*, *C*) ← *extract-shorter-conflict-list-heur-st S*$_0$;
ASSERT(*get-clauses-wl-heur S* = *get-clauses-wl-heur S*$_0$ ∧
get-learned-count S = *get-learned-count S*$_0$);
S ← *find-decomp-wl-st-int n S*;
ASSERT(*get-clauses-wl-heur S* = *get-clauses-wl-heur S*$_0$ ∧
get-learned-count S = *get-learned-count S*$_0$);

if size C > *1*
then do {
let - = *C* ! *1*;
S ← *propagate-bt-wl-D-heur L C S*;
save-phase-st S
}
else do {
propagate-unit-bt-wl-D-int L S
}
}› **for** *S*$_0$
unfolding *backtrack-wl-D-nlit-heur-def Let-def*
by *auto*
have *inv*: ‹*backtrack-wl-D-heur-inv S′*›
if
‹*backtrack-wl-inv S*› **and**
‹(*S′*, *S*) ∈ *?R*›
for *S S′*
using *that* **unfolding** *backtrack-wl-D-heur-inv-def*
by (*cases S*; *cases S′*) (*blast intro*: *exI*[*of* - *S′*])
have *shorter*:
‹*extract-shorter-conflict-list-heur-st S′*
≤ ⇓ {((*T′*, *n*, *C*), *T*). (*T′*, *del-conflict-wl T*) ∈ *twl-st-heur-bt* ∧
n = *get-maximum-level* (*get-trail-wl T*)
(*remove1-mset* (−*lit-of*(*hd* (*get-trail-wl T*))) (*the* (*get-conflict-wl T*))) ∧
mset C = *the* (*get-conflict-wl T*) ∧
get-conflict-wl T ≠ *None*∧
equality-except-conflict-wl T S ∧
get-clauses-wl-heur T′ = *get-clauses-wl-heur S′* ∧
get-learned-count T′ = *get-learned-count S′* ∧
(*1* < *length C* ⟶
highest-lit (*get-trail-wl T*) (*mset* (*tl C*))

```
        (Some (C ! 1, get-level (get-trail-wl T) (C ! 1)))) ∧
       C ≠ [] ∧ hd C = −lit-of(hd (get-trail-wl T)) ∧
       mset C ⊆# the (get-conflict-wl S) ∧
       distinct-mset (the (get-conflict-wl S)) ∧
       literals-are-in-ℒ_in (all-atms-st S) (the (get-conflict-wl S)) ∧
       literals-are-in-ℒ_in-trail (all-atms-st T) (get-trail-wl T) ∧
       get-conflict-wl S ≠ None ∧
       − lit-of (hd (get-trail-wl S)) ∈# ℒ_all (all-atms-st S) ∧
       literals-are-ℒ_in (all-atms-st T) T ∧
       n < count-decided (get-trail-wl T) ∧
      get-trail-wl T ≠ [] ∧
       ¬ tautology (mset C) ∧
       correct-watching S ∧ length (get-clauses-wl-heur T') = length (get-clauses-wl-heur S')}
     (extract-shorter-conflict-wl S)›
 (is ‹- ≤ ⇓ ?shorter -›)
 if
  inv: ‹backtrack-wl-inv S› and
  S'-S: ‹(S', S) ∈ ?R›
 for S S'
proof −
 obtain M N D NE UE NEk UEk NS US N0 U0 Q W where
  S: ‹S = (M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, Q, W)›
  by (cases S)
 let ?M' = ‹get-trail-wl-heur S'›
 let ?arena = ‹get-clauses-wl-heur S'›
 let ?bD' = ‹get-conflict-wl-heur S'›
 let ?W' = ‹get-watched-wl-heur S'›
 let ?vm = ‹get-vmtf-heur S'›
 let ?clvls = ‹get-count-max-lvls-heur S'›
 let ?cach = ‹get-conflict-cach S'›
 let ?outl = ‹get-outlearned-heur S'›
 let ?lcount = ‹get-learned-count S'›
 let ?aivdom = ‹get-aivdom S'›
 let ?b = ‹fst ?bD'›
 let ?D' = ‹snd ?bD'›

 let ?vdom = ‹set (get-vdom-aivdom ?aivdom)›
 have
  M'-M: ‹(?M', M) ∈ trail-pol (all-atms-st S)› and
  ‹(?W', W) ∈ ⟨Id⟩map-fun-rel (D_0 (all-atms-st S))› and
  vm: ‹?vm ∈ bump-heur (all-atms-st S) M› and
  n-d: ‹no-dup M› and
  ‹?clvls ∈ counts-maximum-level M D› and
  cach-empty: ‹cach-refinement-empty (all-atms-st S) ?cach› and
  outl: ‹out-learned M D ?outl› and
  lcount: ‹clss-size-corr N NE UE NEk UEk NS US N0 U0 ?lcount› and
  ‹vdom-m (all-atms-st S) W N ⊆ ?vdom› and
  D': ‹(?bD', D) ∈ option-lookup-clause-rel (all-atms-st S)› and
  arena: ‹valid-arena ?arena N ?vdom› and
  bounded: ‹isasat-input-bounded (all-atms-st S)› and
  aivdom: ‹aivdom-inv-dec ?aivdom (dom-m N)›
  using S'-S unfolding S twl-st-heur-conflict-ana-def
  by (auto simp: S all-atms-def[symmetric])
 obtain T U where
  ℒ_in :‹literals-are-ℒ_in (all-atms-st S) S› and
  S-T: ‹(S, T) ∈ state-wl-l None› and
```

corr: ‹*correct-watching S*› **and**
T-U: ‹(*T*, *U*) ∈ *twl-st-l None*› **and**
trail-nempty: ‹*get-trail-l T* ≠ []› **and**
not-none: ‹*get-conflict-l T* ≠ *None*› **and**
struct-invs: ‹*twl-struct-invs U*› **and**
stgy-invs: ‹*twl-stgy-invs U*› **and**
list-invs: ‹*twl-list-invs T*› **and**
not-empty: ‹*get-conflict-l T* ≠ *Some* {#}› **and**
uL-D: ‹− *lit-of* (*hd* (*get-trail-wl S*)) ∈# *the* (*get-conflict-wl S*)› **and**
nss: ‹*no-step cdcl*$_W$*-restart-mset.skip* (*state*$_W$*-of U*)› **and**
nsr: ‹*no-step cdcl*$_W$*-restart-mset.resolve* (*state*$_W$*-of U*)›
using *inv* **unfolding** *backtrack-wl-inv-def backtrack-wl-inv-def backtrack-l-inv-def backtrack-inv-def*
apply −
apply *normalize-goal*+ **by** *simp*
have *D-none*: ‹*D* ≠ *None*›
using *S-T not-none* **by** (*auto simp*: *S*)
have *b*: ‹¬ *?b*›
using *D′ not-none S-T* **by** (*auto simp*: *option-lookup-clause-rel-def S state-wl-l-def*)
have ‹*get-conflict U* ≠ *Some* {#}›
using *struct-invs S-T T-U uL-D* **by** *auto*
then have ‹*get-learned-clauses0 U* = {#}›
‹*get-init-clauses0 U* = {#}›
using *struct-invs*
by (*cases U*; *auto simp*: *twl-struct-invs-def pcdcl-all-struct-invs-def*
clauses0-inv-def)+
then have *clss0*: ‹*get-learned-clauses0-wl S* = {#}›
‹*get-init-clauses0-wl S* = {#}›
using *S-T T-U* **by** *auto*
have *all-struct*:
‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*state*$_W$*-of U*)›
using *struct-invs* **unfolding** *twl-struct-invs-def pcdcl-all-struct-invs-def*
by *auto*
have
‹*cdcl*$_W$*-restart-mset.no-strange-atm* (*state*$_W$*-of U*)› **and**
lev-inv: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-M-level-inv* (*state*$_W$*-of U*)› **and**
‹∀ *s*∈#*learned-clss* (*state*$_W$*-of U*). ¬ *tautology s*› **and**
dist: ‹*cdcl*$_W$*-restart-mset.distinct-cdcl*$_W$*-state* (*state*$_W$*-of U*)› **and**
confl: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-conflicting* (*state*$_W$*-of U*)› **and**
‹*all-decomposition-implies-m* (*cdcl*$_W$*-restart-mset.clauses* (*state*$_W$*-of U*))
(*get-all-ann-decomposition* (*trail* (*state*$_W$*-of U*)))› **and**
learned: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-learned-clause* (*state*$_W$*-of U*)›
using *all-struct* **unfolding** *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*
by *fast*+
have *n-d*: ‹*no-dup M*›
using *lev-inv S-T T-U* **unfolding** *cdcl*$_W$*-restart-mset.cdcl*$_W$*-M-level-inv-def*
by (*auto simp*: *twl-st S*)
have *M-*$\mathcal{L}_{in}$: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* (*all-atms-st S*) (*get-trail-wl S*)›
apply (*rule literals-are-*$\mathcal{L}_{in}$*-literals-are-*$\mathcal{L}_{in}$*-trail*[*OF S-T struct-invs T-U* $\mathcal{L}_{in}$]) **.**
have *dist-D*: ‹*distinct-mset* (*the* (*get-conflict-wl S*))›
using *dist not-none S-T T-U* **unfolding** *cdcl*$_W$*-restart-mset.distinct-cdcl*$_W$*-state-def S*
by (*auto simp*: *twl-st*)
have ‹*the* (*conflicting* (*state*$_W$*-of U*)) =
add-mset (− *lit-of* (*cdcl*$_W$*-restart-mset.hd-trail* (*state*$_W$*-of U*)))
{#*L* ∈# *the* (*conflicting* (*state*$_W$*-of U*)). *get-level* (*trail* (*state*$_W$*-of U*)) *L*
< *backtrack-lvl* (*state*$_W$*-of U*)#}›
apply (*rule cdcl*$_W$*-restart-mset.no-skip-no-resolve-single-highest-level*)

subgoal using *nss* **unfolding** *S* **by** *simp*
subgoal using *nsr* **unfolding** *S* **by** *simp*
subgoal using *struct-invs* **unfolding** *twl-struct-invs-def*
pcdcl-all-struct-invs-def state$_W$-of-def[*symmetric*] *S* **by** *simp*
subgoal using *stgy-invs* **unfolding** *twl-stgy-invs-def S* **by** *simp*
subgoal using *not-none S-T T-U* **by** (*simp add*: *twl-st*)
subgoal using *not-empty not-none S-T T-U* **by** (*auto simp add*: *twl-st*)
done
then have *D-filter*: ‹*the D* = *add-mset* (− *lit-of* (*hd M*)) {#*L* ∈# *the D*. *get-level M L* < *count-decided M*#}›
using *trail-nempty S-T T-U* **by** (*simp add*: *twl-st S*)
have *tl-outl-D*: ‹*mset* (*tl* (*?outl*[0 := − *lit-of* (*hd M*)])) = *remove1-mset* (*?outl*[0 := − *lit-of* (*hd M*)] ! 0) (*the D*)›
using *outl S-T T-U not-none*
apply (*subst D-filter*)
by (*cases ?outl*) (*auto simp*: *out-learned-def S*)
let *?D* = ‹*remove1-mset* (− *lit-of* (*hd M*)) (*the D*)›
have $\mathcal{L}_{in}$*-S*: ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st S*) (*the* (*get-conflict-wl S*))›
apply (*rule literals-are-in-*$\mathcal{L}_{in}$*-literals-are-in-*$\mathcal{L}_{in}$*-conflict*[*OF S-T - T-U*])
using $\mathcal{L}_{in}$ *not-none struct-invs not-none S-T T-U* **by** (*auto simp*: *S*)
then have $\mathcal{L}_{in}$*-D*: ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st S*) *?D*›
unfolding *S* **by** (*auto intro*: *literals-are-in-*$\mathcal{L}_{in}$*-mono*)
have $\mathcal{L}_{in}$*-NU*: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* (*all-atms-st S*) (*mset* ‘# *ran-mf* (*get-clauses-wl S*))›

by (*auto simp*: *all-atms-def all-lits-def literals-are-in-*$\mathcal{L}_{in}$*-mm-def*
$\mathcal{L}_{all}$*-atm-of-all-lits-of-mm all-lits-st-def simp flip*: *all-lits-st-alt-def*)
(*simp add*: *all-lits-of-mm-union*)
have *tauto-confl*: ‹¬ *tautology* (*the* (*get-conflict-wl S*))›
apply (*rule conflict-not-tautology*[*OF S-T - T-U*])
using *struct-invs not-none S-T T-U* **by** (*auto simp*: *twl-st*)
from *not-tautology-mono*[*OF - this*, *of ?D*] **have** *tauto-D*: ‹¬ *tautology ?D*›
by (*auto simp*: *S*)
have *entailed*:
‹*mset* ‘# *ran-mf* (*get-clauses-wl S*) + (*get-unit-learned-clss-wl S* + *get-unit-init-clss-wl S*) +
(*get-subsumed-init-clauses-wl S* + *get-subsumed-learned-clauses-wl S*) +
(*get-init-clauses0-wl S* + *get-learned-clauses0-wl S*)⊨*pm*
add-mset (− *lit-of* (*hd* (*get-trail-wl S*)))
(*remove1-mset* (− *lit-of* (*hd* (*get-trail-wl S*))) (*the* (*get-conflict-wl S*)))›
using *uL-D learned not-none S-T T-U* **unfolding** *cdcl$_W$-restart-mset.cdcl$_W$-learned-clause-alt-def*
by (*auto simp*: *ac-simps twl-st get-unit-clauses-wl-alt-def*)
define *cach′* **where** ‹*cach′* = (λ-::*nat*. *SEEN-UNKNOWN*)›
have *mini*: ‹*minimize-and-extract-highest-lookup-conflict* (*all-atms-st S*) (*get-trail-wl S*) (*get-clauses-wl S*)
?D cach′ lbd (*?outl*[0 := − *lit-of* (*hd M*)])
≤ ⇓ {((*E*, *s*, *outl*), *E′*). *E* = *E′* ∧ *mset* (*tl outl*) = *E* ∧
outl ! 0 = − *lit-of* (*hd M*) ∧ *E′* ⊆# *remove1-mset* (− *lit-of* (*hd M*)) (*the D*) ∧
outl ≠ []}
(*iterate-over-conflict* (− *lit-of* (*hd M*)) (*get-trail-wl S*)
(*mset* ‘# *ran-mf* (*get-clauses-wl S*))
((*get-unit-learned-clss-wl S* + *get-unit-init-clss-wl S*) +
(*get-subsumed-learned-clauses-wl S* + *get-subsumed-init-clauses-wl S*) +
(*get-learned-clauses0-wl S* + *get-init-clauses0-wl S*))
?D)› **for** *lbd*
apply (*rule minimize-and-extract-highest-lookup-conflict-iterate-over-conflict*[*of S T U*
‹*?outl* [0 := − *lit-of* (*hd M*)]›
‹*remove1-mset* - (*the D*)› ‹*all-atms-st S*› *cach′* ‹−*lit-of* (*hd M*)› *lbd*])

subgoal using *S-T* **.**
subgoal using *T-U* **.**
subgoal using ‹*out-learned M D ?outl*› *tl-outl-D*
by (*auto simp*: *out-learned-def*)
subgoal using *confl not-none S-T T-U* **unfolding** $cdcl_W$*-restart-mset.*$cdcl_W$*-conflicting-def*
by (*auto simp*: *true-annot-CNot-diff twl-st S*)
subgoal
using *dist not-none S-T T-U* **unfolding** $cdcl_W$*-restart-mset.distinct-*$cdcl_W$*-state-def*
by (*auto simp*: *twl-st S*)
subgoal using *tauto-D* **.**
subgoal using *M-*$\mathcal{L}_{in}$ **unfolding** *S* **by** *simp*
subgoal using *struct-invs* **unfolding** *S* **by** *simp*
subgoal using *list-invs* **unfolding** *S* **by** *simp*
subgoal using *M-*$\mathcal{L}_{in}$ *cach-empty S-T T-U*
unfolding *cach-refinement-empty-def conflict-min-analysis-inv-def*
by (*auto dest*: *literals-are-in-*$\mathcal{L}_{in}$*-trail-in-lits-of-l-atms simp*: *cach′-def twl-st S*)
subgoal using *entailed* **unfolding** *S* **by** (*simp add*: *ac-simps*)
subgoal using $\mathcal{L}_{in}$*-D* **.**
subgoal using $\mathcal{L}_{in}$*-NU* **.**
subgoal using ‹*out-learned M D ?outl*› *tl-outl-D*
by (*auto simp*: *out-learned-def*)
subgoal using ‹*out-learned M D ?outl*› *tl-outl-D*
by (*auto simp*: *out-learned-def*)
subgoal using *bounded* **unfolding** *all-atms-def* **by** (*simp add*: *S*)
done
then have *mini*: ‹*minimize-and-extract-highest-lookup-conflict* (*all-atms-st S*) *M N*
?D cach′ lbd (*?outl*[*0* := − *lit-of* (*hd M*)])
≤ ⇓ {((*E*, *s*, *outl*), *E′*). *E* = *E′* ∧ *mset* (*tl outl*) = *E* ∧
outl ! *0* = − *lit-of* (*hd M*) ∧ *E′* ⊆# *remove1-mset* (− *lit-of* (*hd M*)) (*the D*) ∧
outl ≠ []}
(*iterate-over-conflict* (− *lit-of* (*hd M*)) (*get-trail-wl S*)
(*mset* ‘# *ran-mf N*)
(*get-unit-learned-clss-wl S* + *get-unit-init-clss-wl S* +
(*get-subsumed-learned-clauses-wl S* + *get-subsumed-init-clauses-wl S*) +
(*get-learned-clauses0-wl S* + *get-init-clauses0-wl S*)) *?D*)› **for** *lbd*
unfolding *S* **by** *auto*
have *mini*: ‹*minimize-and-extract-highest-lookup-conflict* (*all-atms-st S*) *M N*
?D cach′ lbd (*?outl*[*0* := − *lit-of* (*hd M*)])
≤ ⇓ {((*E*, *s*, *outl*), *E′*). *E* = *E′* ∧ *mset* (*tl outl*) = *E* ∧
outl ! *0* = − *lit-of* (*hd M*) ∧ *E′* ⊆# *remove1-mset* (− *lit-of* (*hd M*)) (*the D*) ∧
outl ≠ []}
(*SPEC* (λ*D′*. *D′* ⊆# *?D* ∧ *mset* ‘# *ran-mf N* +
(*get-unit-learned-clss-wl S* + *get-unit-init-clss-wl S* +
(*get-subsumed-learned-clauses-wl S* + *get-subsumed-init-clauses-wl S*) +
(*get-learned-clauses0-wl S* + *get-init-clauses0-wl S*)) ⊨*pm add-mset* (− *lit-of* (*hd M*))
D′))›
for *lbd*
apply (*rule order.trans*)
apply (*rule mini*)
apply (*rule ref-two-step′*)
apply (*rule order.trans*)
apply (*rule iterate-over-conflict-spec*)
subgoal using *entailed* **by** (*auto simp*: *S ac-simps*)
subgoal
using *dist not-none S-T T-U* **unfolding** *S* $cdcl_W$*-restart-mset.distinct-*$cdcl_W$*-state-def*
by (*auto simp*: *twl-st*)

subgoal by (*auto simp*: *ac-simps*)
done
have *uM-*$\mathcal{L}_{all}$: ‹− *lit-of* (*hd M*) ∈# $\mathcal{L}_{all}$ (*all-atms-st S*)›
using *M-*$\mathcal{L}_{in}$ *trail-nempty S-T T-U* **by** (*cases M*)
(*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-trail-Cons uminus-*$\mathcal{A}_{in}$*-iff twl-st S*)

have *L-D*: ‹*lit-of* (*hd M*) ∉# *the D*› **and**
tauto-confl': ‹¬*tautology* (*remove1-mset* (− *lit-of* (*hd M*)) (*the D*))›
using *uL-D tauto-confl*
by (*auto dest*!: *multi-member-split simp*: *S add-mset-eq-add-mset tautology-add-mset*)
then have *pre1*: ‹*D* ≠ *None* ∧ *delete-from-lookup-conflict-pre* (*all-atms-st S*) (− *lit-of* (*hd M*), *the D*)›
using *not-none uL-D uM-*$\mathcal{L}_{all}$ *S-T T-U* **unfolding** *delete-from-lookup-conflict-pre-def*
by (*auto simp*: *twl-st S*)
have *pre2*: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* (*all-atms-st S*) *M* ∧ *literals-are-in-*$\mathcal{L}_{in}$*-mm* (*all-atms-st S*) (*mset* ‘# *ran-mf N*) ≡ *True*›
and *lits-N*: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* (*all-atms-st S*) (*mset* ‘# *ran-mf N*)›
using *M-*$\mathcal{L}_{in}$ *S-T T-U not-none* $\mathcal{L}_{in}$
unfolding *is-*$\mathcal{L}_{all}$*-def literals-are-in-*$\mathcal{L}_{in}$*-mm-def literals-are-*$\mathcal{L}_{in}$*-def all-atms-def all-lits-def*
all-lits-st-alt-def[*symmetric*] *all-lits-st-def*
by (*auto simp*: *twl-st S all-lits-of-mm-union*)
have ‹*0* < *length ?outl*›
using ‹*out-learned M D ?outl*›
by (*auto simp*: *out-learned-def*)
have *trail-nempty*: ‹*M* ≠ []›
using *trail-nempty S-T T-U*
by (*auto simp*: *twl-st S*)

have *lookup-conflict-remove1-pre*: ‹*lookup-conflict-remove1-pre* (−*lit-of* (*hd M*), *?D'*)›
using *D' not-none not-empty S-T uM-*$\mathcal{L}_{all}$
unfolding *lookup-conflict-remove1-pre-def*
by (*cases* ‹*the D*›)
(*auto simp*: *option-lookup-clause-rel-def lookup-clause-rel-def S*
state-wl-l-def atms-of-def)
then have *lookup-conflict-remove1-pre*: ‹*lookup-conflict-remove1-pre* (−*lit-of-last-trail-pol ?M'*, *?D'*)›
by (*subst lit-of-last-trail-pol-lit-of-last-trail*[*THEN fref-to-Down-unRET-Id*, *of M ?M'*])
(*use M'-M trail-nempty* **in** ‹*auto simp*: *lit-of-hd-trail-def*›)

have ‹− *lit-of* (*hd M*) ∈# (*the D*)›
using *uL-D* **by** (*auto simp*: *S*)
then have *extract-shorter-conflict-wl-alt-def*:
‹*extract-shorter-conflict-wl* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*) = *do* {
- :: *bool list* ← *SPEC* (λ-. *True*);
let K = *lit-of* (*hd M*);
let D = (*remove1-mset* (−*K*) (*the D*));
- ← *RETURN* (); ~~////////////~~
E' ← (*SPEC*
(λ(*E'*). *E'* ⊆# *add-mset* (−*K*) *D* ∧ − *lit-of* (*hd M*) :# *E'* ∧
mset ‘# *ran-mf N* +
(*get-unit-learned-clss-wl S* + *get-unit-init-clss-wl S* +
(*get-subsumed-learned-clauses-wl S* + *get-subsumed-init-clauses-wl S*)) ⊨*pm E'*));
D ← *RETURN* (*Some E'*);
RETURN (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*)
}›
unfolding *extract-shorter-conflict-wl-def*
by (*auto simp*: *RES-RETURN-RES image-iff mset-take-mset-drop-mset' S union-assoc*

Un-commute Let-def Un-assoc sup-left-commute)

have *lookup-clause-rel-unique*: ‹$(D', a) \in$ *lookup-clause-rel* $\mathcal{A} \Longrightarrow (D', b) \in$ *lookup-clause-rel* $\mathcal{A} \Longrightarrow a = b$›
for a b $\mathcal{A}$
by (*auto simp*: *lookup-clause-rel-def mset-as-position-right-unique*)
have *isa-minimize-and-extract-highest-lookup-conflict*:
‹*isa-minimize-and-extract-highest-lookup-conflict ?M' ?arena*
(*lookup-conflict-remove1* ($-$*lit-of* (*hd M*)) *?D'*) *?cach lbd* (*?outl*[0 := $-$ *lit-of* (*hd M*)])
$\leq \Downarrow$ {((*E*, *s*, *outl*), *E'*).
(*E*, *mset* (*tl outl*)) $\in$ *lookup-clause-rel* (*all-atms-st S*) $\wedge$
mset outl = *E'* $\wedge$
outl ! 0 = $-$ *lit-of* (*hd M*) $\wedge$
E' $\subseteq\#$ *the D* $\wedge$ *outl* $\neq$ [] $\wedge$ *distinct outl* $\wedge$ *literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st S*) (*mset outl*) $\wedge$
$\neg$*tautology* (*mset outl*) $\wedge$
($\exists$ *cach'*. (*s*, *cach'*) $\in$ *cach-refinement* (*all-atms-st S*))}
(*SPEC* (λ*E'*. *E'* $\subseteq\#$ *add-mset* ($-$ *lit-of* (*hd M*)) (*remove1-mset* ($-$ *lit-of* (*hd M*)) (*the D*)) $\wedge$
$-$ *lit-of* (*hd M*) $\in\#$ *E'* $\wedge$
mset '# *ran-mf N* +
(*get-unit-learned-clss-wl S* + *get-unit-init-clss-wl S* +
(*get-subsumed-learned-clauses-wl S* + *get-subsumed-init-clauses-wl S*) +
(*get-learned-clauses0-wl S* + *get-init-clauses0-wl S*)) $\models$*pm*
E'))›
(**is** ‹- $\leq \Downarrow$ *?minimize* (*RES ?E*)›) **for** *lbd*
apply (*rule order-trans*)
apply (*rule*
isa-minimize-and-extract-highest-lookup-conflict-minimize-and-extract-highest-lookup-conflict
[*THEN fref-to-Down-curry5*,
of ‹*all-atms-st S*› *M N* ‹*remove1-mset* ($-$*lit-of* (*hd M*)) (*the D*)› *cach'* *lbd* ‹*?outl*[0 := $-$ *lit-of* (*hd M*)]›
- - - - - - ‹*?vdom*›])
subgoal using *bounded* **by** (*auto simp*: *S all-atms-def*)
subgoal using *tauto-confl' pre2* **by** *auto*
subgoal using *D' not-none arena S-T uL-D uM-*$\mathcal{L}_{all}$ *not-empty D' L-D b cach-empty M'-M* **unfolding** *all-atms-def*
by (*auto simp*: *option-lookup-clause-rel-def S state-wl-l-def image-image cach-refinement-empty-def cach'-def*
intro!: *lookup-conflict-remove1*[*THEN fref-to-Down-unRET-uncurry*]
dest: *multi-member-split lookup-clause-rel-unique*)
apply (*rule order-trans*)
apply (*rule mini*[*THEN ref-two-step'*])
subgoal
using *uL-D dist-D tauto-D* $\mathcal{L}_{in}$*-S* $\mathcal{L}_{in}$*-D tauto-D L-D*
by (*auto 5 3 simp*: *conc-fun-chain conc-fun-RES image-iff S union-assoc insert-subset-eq-iff*
neq-Nil-conv literals-are-in-$\mathcal{L}_{in}$*-add-mset tautology-add-mset*
intro: *literals-are-in-*$\mathcal{L}_{in}$*-mono*
dest: *distinct-mset-mono not-tautology-mono*
dest!: *multi-member-split*)
done

have *empty-conflict-and-extract-clause-heur*: ‹*isa-empty-conflict-and-extract-clause-heur ?M' x1 x2a*
$\leq \Downarrow$ ({((*E*, *outl*, *n*), *E'*).
(*E*, *None*) $\in$ *option-lookup-clause-rel* (*all-atms-st S*) $\wedge$
mset outl = *the E'* $\wedge$
outl ! 0 = $-$ *lit-of* (*hd M*) $\wedge$
the E' $\subseteq\#$ *the D* $\wedge$ *outl* $\neq$ [] $\wedge$ *E'* $\neq$ *None* $\wedge$

```
        (1 < length outl ⟶
          highest-lit M (mset (tl outl)) (Some (outl ! 1, get-level M (outl ! 1)))) ∧
        (1 < length outl ⟶ n = get-level M (outl ! 1)) ∧ (length outl = 1 ⟶ n = 0)}) (RETURN
(Some E'))›
     (is ‹- ≤ ⇓ ?empty-conflict -›)
     if
       ‹M ≠ []› and
       ‹− lit-of (hd M) ∈# 𝓛all (all-atms-st S)› and
       ‹0 < length ?outl› and
       ‹lookup-conflict-remove1-pre (− lit-of (hd M), ?D')› and
       ‹(x, E') ∈ ?minimize› and
       ‹E' ∈ ?E› and
       ‹x2 = (x1a, x2a)› and
       ‹x = (x1, x2)›
     for x :: ‹(nat × bool option list) × (minimize-status list × nat list) × nat literal list› and
       E' :: ‹nat literal multiset› and
       x1 :: ‹nat × bool option list› and
       x2 :: ‹(minimize-status list × nat list) × nat literal list› and
       x1a :: ‹minimize-status list × nat list› and
       x2a :: ‹nat literal list›
   proof −
     show ?thesis
       apply (rule order-trans)
        apply (rule isa-empty-conflict-and-extract-clause-heur-empty-conflict-and-extract-clause-heur
          [THEN fref-to-Down-curry2, of - - - M x1 x2a ‹all-atms-st S›])
         apply fast
       subgoal using M'-M by auto
       apply (subst Down-id-eq)
       apply (rule order.trans)
        apply (rule empty-conflict-and-extract-clause-heur-empty-conflict-and-extract-clause[of ‹mset (tl
x2a)›])
       subgoal by auto
       subgoal using that by auto
       subgoal using that by auto
       subgoal using that by auto
       subgoal using that by auto
       subgoal using that by auto
       subgoal using bounded unfolding S all-atms-def by simp
       subgoal unfolding empty-conflict-and-extract-clause-def
         using that
         by (auto simp: conc-fun-RES RETURN-def)
       done
   qed

   have final: ‹((set-lbd-wl-heur lbd (set-ccach-max-wl-heur (empty-cach-ref x1a) (set-vmtf-wl-heur vm'
(set-conflict-wl-heur x1b (set-outl-wl-heur (take 1 x2a) S')))), x2c, x1c),
        (M, N, Da, NE, UE, NEk, UEk, NS, US, N0, U0, Q, W))
        ∈ ?shorter› (is ‹((?updated-state, -, -), -) ∈ -›)
     if
       ‹M ≠ []› and
       ‹− lit-of (hd M) ∈# 𝓛all (all-atms-st S)› and
       ‹0 < length ?outl› and
       ‹lookup-conflict-remove1-pre (− lit-of (hd M), ?D')› and
       mini: ‹(x, E') ∈ ?minimize› and
       ‹E' ∈ ?E› and
       ‹(xa, Da) ∈ ?empty-conflict› and
```

st[*simp*]:
‹*x2b* = (*x1c*, *x2c*)›
‹*x2* = (*x1a*, *x2a*)›
‹*x* = (*x1*, *x2*)›
‹*xa* = (*x1b*, *x2b*)› **and**
vm': ‹(*vm'*, *uu*) ∈ {(*c*, *uu*). *c* ∈ *bump-heur* (*all-atms-st S*) *M*}›
for *x E' x1 x2 x1a x2a xa Da x1b x2b x1c x2c vm' uu lbd*
proof −
have *x1b-None*: ‹(*x1b*, *None*) ∈ *option-lookup-clause-rel* (*all-atms-st S*)›
using *that* **apply** (*auto simp*: *S simp flip*: *all-atms-def*)
done
have *cach*[*simp*]: ‹*cach-refinement-empty* (*all-atms-st S*) (*empty-cach-ref x1a*)›
using *empty-cach-ref-empty-cach*[*of* ‹*all-atms-st S*›, *THEN fref-to-Down-unRET*, *of x1a*]
mini bounded
by (*auto simp add*: *cach-refinement-empty-def empty-cach-def cach'-def S*
simp flip: *all-atms-def*)

have
out: ‹*out-learned M None* (*take* (*Suc 0*) *x2a*)› **and**
x1c-Da: ‹*mset x1c* = *the Da*› **and**
Da-None: ‹*Da* ≠ *None*› **and**
Da-D: ‹*the Da* ⊆# *the D*› **and**
x1c-D: ‹*mset x1c* ⊆# *the D*› **and**
x1c: ‹*x1c* ≠ []› **and**
hd-x1c: ‹*hd x1c* = − *lit-of* (*hd M*)› **and**
highest: ‹*Suc 0* < *length x1c* ⟹ *x2c* = *get-level M* (*x1c* ! *1*) ∧
highest-lit M (*mset* (*tl x1c*))
(*Some* (*x1c* ! *Suc 0*, *get-level M* (*x1c* ! *Suc 0*)))› **and**
highest2: ‹*length x1c* = *Suc 0* ⟹ *x2c* = *0*› **and**
‹*E'* = *mset x2a*› **and**
‹− *lit-of* (*M* ! *0*) ∈ *set x2a*› **and**
‹(λ*x*. *mset* (*fst x*)) ‘ *set-mset* (*ran-m N*) ∪
(*set-mset* (*get-unit-learned-clss-wl S*) ∪ *set-mset* (*get-unit-init-clss-wl S*)) ∪
(*set-mset* (*get-subsumed-learned-clauses-wl S*) ∪ *set-mset* (*get-subsumed-init-clauses-wl S*) ∪
(*set-mset* (*get-learned-clauses0-wl S*) ∪ *set-mset* (*get-init-clauses0-wl S*))) ⊨*p*
mset x2a› **and**
‹*x2a* ! *0* = − *lit-of* (*M* ! *0*)› **and**
‹*x1c* ! *0* = − *lit-of* (*M* ! *0*)› **and**
‹*mset x2a* ⊆# *the D*› **and**
‹*mset x1c* ⊆# *the D*› **and**
‹*x2a* ≠ []› **and**
x1c-nempty: ‹*x1c* ≠ []› **and**
‹*distinct x2a*› **and**
Da: ‹*Da* = *Some* (*mset x1c*)› **and**
‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st S*) (*mset x2a*)› **and**
‹¬ *tautology* (*mset x2a*)›
using *that*
unfolding *out-learned-def*
by (*auto simp add*: *hd-conv-nth S ac-simps simp flip*: *all-atms-def*)
have *Da-D'*: ‹*remove1-mset* (− *lit-of* (*hd M*)) (*the Da*) ⊆# *remove1-mset* (− *lit-of* (*hd M*)) (*the D*)›
using *Da-D mset-le-subtract* **by** *blast*

have *K*: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-stgy-invariant* (*state*$_W$*-of U*)›
using *stgy-invs* **unfolding** *twl-stgy-invs-def* **by** *fast*
have ‹*get-maximum-level M* {#*L* ∈# *the D*. *get-level M L* < *count-decided M*#}

$< count\text{-}decided\ M\rangle$
using $cdcl_W\text{-}restart\text{-}mset.no\text{-}skip\text{-}no\text{-}resolve\text{-}level\text{-}get\text{-}maximum\text{-}lvl\text{-}le[OF\ nss\ nsr\ all\text{-}struct\ K]$
$not\text{-}none\ not\text{-}empty\ confl\ trail\text{-}nempty\ S\text{-}T\ T\text{-}U$
unfolding $get\text{-}maximum\text{-}level\text{-}def$ **by** $(auto\ simp:\ twl\text{-}st\ S)$
then have
$\langle get\text{-}maximum\text{-}level\ M\ (remove1\text{-}mset\ (-\ lit\text{-}of\ (hd\ M))\ (the\ D)) < count\text{-}decided\ M\rangle$
by $(subst\ D\text{-}filter)\ auto$
then have $max\text{-}lvl\text{-}le$:
$\langle get\text{-}maximum\text{-}level\ M\ (remove1\text{-}mset\ (-\ lit\text{-}of\ (hd\ M))\ (the\ Da)) < count\text{-}decided\ M\rangle$
using $get\text{-}maximum\text{-}level\text{-}mono[OF\ Da\text{-}D',\ of\ M]$ **by** $auto$
have $\langle(?updated\text{-}state,\ del\text{-}conflict\text{-}wl\ (M,\ N,\ Da,\ NE,\ UE,\ NEk,\ UEk,\ NS,\ US,\ N0,\ U0,\ Q,\ W))$
$\in twl\text{-}st\text{-}heur\text{-}bt\rangle$
using $S'\text{-}S\ x1b\text{-}None\ cach\ out\ vm'\ cach$ **unfolding** $twl\text{-}st\text{-}heur\text{-}bt\text{-}def\ st$
by $(auto\ simp:\ twl\text{-}st\text{-}heur\text{-}def\ del\text{-}conflict\text{-}wl\text{-}def\ S\ twl\text{-}st\text{-}heur\text{-}bt\text{-}def$
$twl\text{-}st\text{-}heur\text{-}conflict\text{-}ana\text{-}def\ all\text{-}atms\text{-}st\text{-}def\ simp\ del:\ all\text{-}atms\text{-}st\text{-}def[symmetric])$

moreover have $x2c$: $\langle x2c = get\text{-}maximum\text{-}level\ M\ (remove1\text{-}mset\ (-\ lit\text{-}of\ (hd\ M))\ (the\ Da))\rangle$
using $highest\ highest2\ x1c\text{-}nempty\ hd\text{-}x1c$
by $(cases\ \langle length\ x1c = Suc\ 0\rangle;\ cases\ x1c)$
$(auto\ simp:\ highest\text{-}lit\text{-}def\ Da\ mset\text{-}tl)$
moreover have $\langle literals\text{-}are\text{-}\mathcal{L}_{in}\ (all\text{-}atms\text{-}st\ S)\ (M,\ N,\ Some\ (mset\ x1c),\ NE,\ UE,\ NEk,\ UEk,$
$NS,\ US,\ N0,\ U0,\ Q,\ W)\rangle$
using $\mathcal{L}_{in}$
by $(auto\ simp:\ S\ x2c\ literals\text{-}are\text{-}\mathcal{L}_{in}\text{-}def\ blits\text{-}in\text{-}\mathcal{L}_{in}\text{-}def\ simp\ flip:\ all\text{-}atms\text{-}def)$
moreover have $\langle\neg tautology\ (mset\ x1c)\rangle$
using $tauto\text{-}confl\ \ not\text{-}tautology\text{-}mono[OF\ x1c\text{-}D]$
by $(auto\ simp:\ S\ x2c)$
ultimately show $?thesis$
using $\mathcal{L}_{in}\text{-}S\ x1c\text{-}Da\ Da\text{-}None\ dist\text{-}D\ D\text{-}none\ x1c\text{-}D\ x1c\ hd\text{-}x1c\ highest\ uM\text{-}\mathcal{L}_{all}\ vm'\ M\text{-}\mathcal{L}_{in}$
$max\text{-}lvl\text{-}le\ corr\ trail\text{-}nempty$ **unfolding** $literals\text{-}are\text{-}\mathcal{L}_{in}\text{-}def\ all\text{-}lits\text{-}st\text{-}alt\text{-}def[symmetric]$
by $(simp\ add:\ \ S\ x2c\ is\text{-}\mathcal{L}_{all}\text{-}def\ all\text{-}lits\text{-}st\text{-}alt\text{-}def[symmetric],$
$simp\ add:\ all\text{-}atms\text{-}st\text{-}def)$
qed
have $hd\text{-}M'\text{-}M$: $\langle lit\text{-}of\text{-}last\text{-}trail\text{-}pol\ ?M' = lit\text{-}of\ (hd\ M)\rangle$
by $(subst\ lit\text{-}of\text{-}last\text{-}trail\text{-}pol\text{-}lit\text{-}of\text{-}last\text{-}trail[THEN\ fref\text{-}to\text{-}Down\text{-}unRET\text{-}Id,\ of\ M\ ?M'])$
$(use\ M'\text{-}M\ trail\text{-}nempty$ **in** $\langle auto\ simp:\ lit\text{-}of\text{-}hd\text{-}trail\text{-}def\rangle)$

have $outl\text{-}hd\text{-}tl$: $\langle ?outl[0 := -\ lit\text{-}of\ (hd\ M)] = -\ lit\text{-}of\ (hd\ M)\ \#\ tl\ (?outl[0 := -\ lit\text{-}of\ (hd\ M)])\rangle$
and
$[simp]$: $\langle ?outl \neq []\rangle$
using $outl$ **unfolding** $out\text{-}learned\text{-}def$
by $(cases\ ?outl;\ auto;\ fail)+$
have $uM\text{-}D$: $\langle -\ lit\text{-}of\ (hd\ M) \in\#\ the\ D\rangle$
by $(subst\ D\text{-}filter)\ auto$
have $mset\text{-}outl\text{-}D$: $\langle mset\ (?outl[0 := -\ lit\text{-}of\ (hd\ M)]) = (the\ D)\rangle$
by $(subst\ outl\text{-}hd\text{-}tl,\ subst\ mset.simps,\ subst\ tl\text{-}outl\text{-}D,\ subst\ D\text{-}filter)$
$(use\ uM\text{-}D\ D\text{-}filter[symmetric]$ **in** $auto)$
from $arg\text{-}cong[OF\ this,\ of\ set\text{-}mset]$ **have** $set\text{-}outl\text{-}D$: $\langle set\ (?outl[0 := -\ lit\text{-}of\ (hd\ M)]) = set\text{-}mset$
$(the\ D)\rangle$
by $auto$
have $outl\text{-}Lall$: $\langle\forall L \in set\ (?outl[0 := -\ lit\text{-}of\ (hd\ M)]).\ L \in\#\ \mathcal{L}_{all}\ (all\text{-}atms\text{-}st\ S)\rangle$
using $\mathcal{L}_{in}\text{-}S$ **unfolding** $set\text{-}outl\text{-}D$
by $(auto\ simp:\ S\ all\text{-}lits\text{-}of\text{-}m\text{-}add\text{-}mset$
$all\text{-}atms\text{-}def\ literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\text{-}def\ literals\text{-}are\text{-}in\text{-}\mathcal{L}_{in}\text{-}in\text{-}mset\text{-}\mathcal{L}_{all}$
$dest:\ multi\text{-}member\text{-}split)$

have *vmtf-mark-to-rescore-also-reasons*:
‹*isa-vmtf-mark-to-rescore-also-reasons ?M′ ?arena* (*?outl*[*0* := − *lit-of* (*hd M*)]) *K ?vm*
≤ *SPEC* (λ*c*. (*c*, ()) ∈ {(*c*, -). *c* ∈ *bump-heur* (*all-atms-st S*) *M*})›
if
‹*M* ≠ []› **and**
‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* (*all-atms-st S*) *M*› **and**
‹− *lit-of* (*hd M*) ∈# $\mathcal{L}_{all}$ (*all-atms-st S*)› **and**
‹*0* < *length ?outl*› **and**
‹*lookup-conflict-remove1-pre* (− *lit-of* (*hd M*), *?D′*)›
for *K*
proof −
have *outl-Lall*: ‹∀ *L*∈*set* (*?outl*[*0* := − *lit-of* (*hd M*)]). *L* ∈# $\mathcal{L}_{all}$ (*all-atms-st S*)›
using $\mathcal{L}_{in}$*-S* **unfolding** *set-outl-D*
by (*auto simp*: *S all-lits-of-m-add-mset*
all-atms-def literals-are-in-$\mathcal{L}_{in}$*-def literals-are-in-*$\mathcal{L}_{in}$*-in-mset-*$\mathcal{L}_{all}$
dest: *multi-member-split*)
have ‹*distinct* (*?outl*[*0* := − *lit-of* (*hd M*)])› **using** *dist-D* **by**(*auto simp*: *S mset-outl-D*[*symmetric*])
then have *length-outl*: ‹*length ?outl* ≤ *unat32-max*›
using *bounded tauto-confl* $\mathcal{L}_{in}$*-S simple-clss-size-upper-div2*[*OF bounded*, *of* ‹*mset* (*?outl*[*0* := − *lit-of* (*hd M*)])›]
by (*auto simp*: *out-learned-def S mset-outl-D*[*symmetric*] *unat32-max-def simp flip*: *all-atms-def*)
have *lit-annots*: ‹∀ *L*∈*set* (*?outl*[*0* := − *lit-of* (*hd M*)]).
∀ *C*. *Propagated* (− *L*) *C* ∈ *set M* ⟶
C ≠ *0* ⟶
C ∈# *dom-m N* ∧
(∀ *C*∈*set* [*C*..<*C* + *arena-length ?arena C*]. *arena-lit ?arena C* ∈# $\mathcal{L}_{all}$ (*all-atms-st S*))›
unfolding *set-outl-D*
apply (*intro ballI allI impI conjI*)
subgoal
using *list-invs S-T* **unfolding** *twl-list-invs-def*
by (*auto simp*: *S*)
subgoal for *L C i*
using *list-invs S-T arena lits-N literals-are-in-*$\mathcal{L}_{in}$*-mm-in-*$\mathcal{L}_{all}$[*of* ‹(*all-atms-st S*)› *N C* ‹*i* − *C*›]
unfolding *twl-list-invs-def*
by (*auto simp*: *S arena-lifting all-atms-def*[*symmetric*])
done
show *?thesis*
apply (*cases ?vm*)
apply (*rule order.trans*,
rule isa-vmtf-mark-to-rescore-also-reasons-vmtf-mark-to-rescore-also-reasons[*of* ‹*all-atms-st S*›,
THEN fref-to-Down-curry4,
of - - - K ?vm M ?arena ‹*?outl*[*0* := − *lit-of* (*hd M*)]› *K ?vm*])
subgoal using *bounded S* **by** (*auto simp*: *all-atms-def*)
subgoal using *vm arena M′-M* **by** (*auto simp*:)[]
apply (*rule order.trans*, *rule ref-two-step′*)
apply (*rule vmtf-mark-to-rescore-also-reasons-spec*[*OF - arena - - outl-Lall lit-annots*])
subgoal using *vm* **by** *auto*
subgoal using *length-outl* **by** *auto*
subgoal using *bounded* **by** *auto*
subgoal by (*auto simp*: *conc-fun-RES S all-atms-def*)
done
qed

have ‹*get-conflict U* ≠ *Some* {#}›

```
  using struct-invs confl S-T T-U uL-D by auto
 then have ‹get-learned-clauses0 U = {#}›
  ‹get-init-clauses0 U = {#}›
  using struct-invs
  by (cases U; auto simp: twl-struct-invs-def pcdcl-all-struct-invs-def
    clauses0-inv-def)+
 then have clss0: ‹get-learned-clauses0-wl S = {#}›
  ‹get-init-clauses0-wl S = {#}›
  using S-T T-U by auto
 have GG[refine0]:‹⇓ {((E, s, outl), E').
   (E, mset (tl outl)) ∈ lookup-clause-rel (all-atms-st S) ∧
   mset outl = E' ∧
   outl ! 0 = − lit-of (hd M) ∧
   E' ⊆# the D ∧
   outl ≠ [] ∧
   distinct outl ∧
   literals-are-in-ℒ_in (all-atms-st S) (mset outl) ∧
   ¬ tautology (mset outl) ∧ (∃ cach'. (s, cach') ∈ cach-refinement (all-atms-st S))}
  (SPEC
   (λE'. E' ⊆# add-mset (− lit-of (hd M)) (remove1-mset (− lit-of (hd M)) (the D)) ∧
        − lit-of (hd M) ∈# E' ∧
        mset `# ran-mf N +
        (get-unit-learned-clss-wl S + get-unit-init-clss-wl S +
         (get-subsumed-learned-clauses-wl S + get-subsumed-init-clauses-wl S) +
         (get-learned-clauses0-wl S + get-init-clauses0-wl S)) ⊨pm
        E'))
 ≤ ⇓ {((E, s, outl), E').
   (E, mset (tl outl)) ∈ lookup-clause-rel (all-atms-st S) ∧
   mset outl = E' ∧
   outl ! 0 = − lit-of (hd M) ∧
   E' ⊆# the D ∧
   outl ≠ [] ∧
   distinct outl ∧
   literals-are-in-ℒ_in (all-atms-st S) (mset outl) ∧
   ¬ tautology (mset outl) ∧ (∃ cach'. (s, cach') ∈ cach-refinement (all-atms-st S))}
   (SPEC
    (λE'. E' ⊆# add-mset (− lit-of (hd M)) (remove1-mset (− lit-of (hd M)) (the D)) ∧
         − lit-of (hd M) ∈# E' ∧
         mset `# ran-mf N +
         (get-unit-learned-clss-wl S + get-unit-init-clss-wl S +
          (get-subsumed-learned-clauses-wl S + get-subsumed-init-clauses-wl S)) ⊨pm
E'))›
   by (rule ref-two-step') (use clss0 in auto)
 show ?thesis
   supply [[goals-limit=1]]
   unfolding extract-shorter-conflict-list-heur-st-def
    empty-conflict-and-extract-clause-def S prod.simps
   apply (rewrite at ‹let - = list-update - - - in -› Let-def)
   apply (rewrite at ‹let - = get-trail-wl-heur S' in -› Let-def)
   apply (rewrite at ‹let - = get-clauses-wl-heur S' in -› Let-def)
   apply (rewrite at ‹let - = get-outlearned-heur S' in -› Let-def)
   apply (rewrite at ‹let - = get-vmtf-heur S' in -› Let-def)
   apply (rewrite at ‹let - = get-lbd S' in -› Let-def)
   apply (rewrite at ‹let - = get-conflict-wl-heur S' in -› Let-def)
   apply (rewrite at ‹let - = get-conflict-cach S' in -› Let-def)
   apply (rewrite at ‹let - = empty-cach-ref - in - › Let-def)
```

unfolding *hd-M′-M*
apply (*subst case-prod-beta*)
apply (*subst extract-shorter-conflict-wl-alt-def*)
apply (*refine-vcg isa-minimize-and-extract-highest-lookup-conflict*[*THEN order-trans*]
empty-conflict-and-extract-clause-heur)
subgoal
apply (*subst* (*2*) *Down-id-eq*[*symmetric*], *rule mark-lbd-from-list-heur-correctness*[*of - M*
‹(*all-atms-st S*)›])
apply (*use outl-Lall* **in** ‹*auto simp*: *M′-M literals-are-in-*$\mathcal{L}_{in}$*-def*
in-all-lits-of-m-ain-atms-of-iff in-$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$›)
by (*cases ?outl*) *auto*
subgoal using *trail-nempty* **using** *M′-M* **by** (*auto simp*: *trail-pol-def ann-lits-split-reasons-def*)
subgoal using ‹*0 < length ?outl*› **.**
subgoal unfolding *hd-M′-M*[*symmetric*] **by** (*rule lookup-conflict-remove1-pre*)
apply (*rule vmtf-mark-to-rescore-also-reasons*; *assumption?*)
subgoal using *trail-nempty* **.**
subgoal using *pre2* **by** (*auto simp*: *S all-atms-def*)
subgoal using *uM-*$\mathcal{L}_{all}$ **by** (*auto simp*: *S all-atms-def*)
subgoal premises *p*
using *bounded p* **by** (*auto simp*: *S empty-cach-ref-pre-def cach-refinement-alt-def*
intro!: *IsaSAT-Lookup-Conflict.bounded-included-le simp*: *all-atms-def simp del*: *isasat-input-bounded-def*
intro: *true-clss-cls-subsetI*)
subgoal by *auto*
subgoal using *bounded pre2*
by (*auto dest*!: *simple-clss-size-upper-div2 simp*: *unat32-max-def S all-atms-def*[*symmetric*]
simp del: *isasat-input-bounded-def*)
subgoal using *trail-nempty* **by** *fast*
subgoal using *uM-*$\mathcal{L}_{all}$ **.**
apply (*assumption*)+
subgoal
using *trail-nempty uM-*$\mathcal{L}_{all}$
unfolding *S*[*symmetric*]
by (*auto dest*!: *simp*: *clss0*)
apply *assumption*+
subgoal for *lbd uu vm uua x E′ x1 x2 x1a x2a xa Da a b aa ba*
using *trail-nempty uM-*$\mathcal{L}_{all}$ **apply** −
unfolding *S*[*symmetric*] *all-lits-alt-def*[*symmetric*]
by (*rule final*[*unfolded clss0 Multiset.empty-neutral*])
done
qed

have *find-decomp-wl-nlit*: ‹*find-decomp-wl-st-int n T*
≤ ⇓ {(*U*, *U′*). (*U*, *U′*) ∈ *twl-st-heur-bt* ∧ *equality-except-trail-wl U′′ T′* ∧
(∃ *K M2*. (*Decided K* # (*get-trail-wl U′*), *M2*) ∈ *set* (*get-all-ann-decomposition* (*get-trail-wl T′*))
∧
get-level (*get-trail-wl T′*) *K* = *get-maximum-level* (*get-trail-wl T′*) (*the* (*get-conflict-wl T′*) −
{#−*lit-of* (*hd* (*get-trail-wl T′*))#}) + *1* ∧
get-clauses-wl-heur U = *get-clauses-wl-heur S* ∧
get-learned-count U = *get-learned-count S*) ∧
(*get-trail-wl U′′*, *get-vmtf-heur U*) ∈ (*Id* ×$_f$ *Id*)$^{-1}$ ‘‘
(*Collect* (*find-decomp-w-ns-prop* (*all-atms-st T′*) (*get-trail-wl T′*) *n* (*get-vmtf-heur T*)))}
(*find-decomp-wl LK′ T′*)›
(**is** ‹- ≤ ⇓ *?find-decomp* -›)
if
‹(*S*, *S′*) ∈ *?R*› **and**
‹*backtrack-wl-inv S′*› **and**

```
    ‹backtrack-wl-D-heur-inv S› and
    TT': ‹(TnC, T') ∈ ?shorter S' S› and
    [simp]: ‹nC = (n, C)› and
    [simp]: ‹TnC = (T, nC)› and
    KK': ‹(LK, LK') ∈ {(L, L'). L = L' ∧ L = lit-of (hd (get-trail-wl S'))}›
  for S S' TnC T' T nC n C LK LK'
proof −
  obtain M N D NE UE NEk UEk NS US N0 U0 Q W where
    T': ‹T' = (M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, Q, W)›
    by (cases T')

  let ?M' = ‹get-trail-wl-heur T›
  let ?arena = ‹get-clauses-wl-heur T›
  let ?D' = ‹get-conflict-wl-heur T›
  let ?W' = ‹get-watched-wl-heur T›
  let ?vm = ‹get-vmtf-heur T›
  let ?clvls = ‹get-count-max-lvls-heur T›
  let ?cach = ‹get-conflict-cach T›
  let ?outl = ‹get-outlearned-heur T›
  let ?lcount = ‹get-learned-count T›
  let ?aivdom = ‹get-aivdom T›

  let ?vdom = ‹set (get-vdom-aivdom ?aivdom)›

  have
    vm: ‹?vm ∈ bump-heur (all-atms-st T') M› and
    M'M: ‹(?M', M) ∈ trail-pol (all-atms-st T')› and
    lits-trail: ‹literals-are-in-ℒ_in-trail (all-atms-st T') (get-trail-wl T')›
    using TT' by (auto simp: twl-st-heur-bt-def del-conflict-wl-def all-atms-st-def
        all-atms-def[symmetric] T' all-lits-st-alt-def[symmetric])

  have [simp]:
     ‹LK' = lit-of (hd (get-trail-wl T'))›
     ‹LK = LK'›
     using KK' TT' by (auto simp: equality-except-conflict-wl-get-trail-wl)

  have n: ‹n = get-maximum-level M (remove1-mset (− lit-of (hd M)) (mset C))› and
    eq: ‹equality-except-conflict-wl T' S'› and
    ‹the D = mset C› ‹D ≠ None› and
    clss-eq: ‹get-clauses-wl-heur S = ?arena› and
    n: ‹n < count-decided (get-trail-wl T')› and
    bounded: ‹isasat-input-bounded (all-atms-st T')› and
    T-T': ‹(T, del-conflict-wl T') ∈ twl-st-heur-bt› and
    n2: ‹n = get-maximum-level M (remove1-mset (− lit-of (hd M)) (the D))› and
    lcount: ‹get-learned-count T = get-learned-count S›
    using TT' KK' by (auto simp: T' twl-st-heur-bt-def del-conflict-wl-def all-atms-st-def
        T' all-lits-st-alt-def[symmetric] simp flip: all-atms-def
        simp del: isasat-input-bounded-def)
  have [simp]: ‹get-trail-wl S' = M›
    using eq ‹the D = mset C› ‹D ≠ None› by (cases S'; auto simp: T')
  have [simp]: ‹get-clauses-wl-heur S = ?arena›
    using TT' by (auto simp: T')

  have n-d: ‹no-dup M›
    using M'M unfolding trail-pol-def by auto
```

```
  have [simp]: ‹NO-MATCH [] M ⟹ out-learned M None ai ⟷ out-learned [] None ai› for M ai
    by (auto simp: out-learned-def)

  show ?thesis
    unfolding T' find-decomp-wl-st-int-def prod.case Let-def
    apply (rule bind-refine-res)
     prefer 2
     apply (rule order.trans)
      apply (rule isa-find-decomp-wl-imp-find-decomp-wl-imp[THEN fref-to-Down-curry2, of M n ?vm
          - - - ‹all-atms-st T'›])
    subgoal using n by (auto simp: T')
    subgoal using M'M vm by auto
     apply (rule order.trans)
      apply (rule ref-two-step')
      apply (rule find-decomp-wl-imp-le-find-decomp-wl')
    subgoal using vm .
    subgoal using lits-trail by (auto simp: T')
    subgoal using n by (auto simp: T')
    subgoal using n-d .
    subgoal using bounded .
    subgoal using n by (auto simp: T')
    unfolding find-decomp-w-ns-def conc-fun-RES
     apply (rule order.refl)
    using T-T' n-d lcount
    apply (cases ‹get-vmtf-heur T›)
    apply (auto simp: find-decomp-wl-def twl-st-heur-bt-def T' del-conflict-wl-def
        dest: no-dup-appendD
        simp flip: all-atms-def n2
        intro!: RETURN-RES-refine
        intro: )
    by (auto dest: no-dup-appendD intro:  simp: T' all-atms-st-def)
qed

have fst-find-lit-of-max-level-wl: ‹RETURN (C ! 1)
   ≤ ⇓ Id
       (find-lit-of-max-level-wl U' LK')›
  if
    ‹(S, S') ∈ ?R› and
    ‹backtrack-wl-inv S'› and
    ‹backtrack-wl-D-heur-inv S› and
    TT': ‹(TnC, T') ∈ ?shorter S' S› and
    [simp]: ‹nC = (n, C)› and
    [simp]: ‹TnC = (T, nC)› and
    find-decomp: ‹(U, U') ∈ ?find-decomp S T' n› and
    size-C: ‹1 < length C› and
    size-conflict-U': ‹1 < size (the (get-conflict-wl U'))› and
     KK': ‹(LK, LK') ∈ {(L, L'). L = L' ∧ L = lit-of (hd (get-trail-wl S'))}›
  for S S' TnC T' T nC n C U U' LK LK'
proof −
  obtain M N NE UE Q W NEk UEk NS US N0 U0 where
    T': ‹T' = (M, N, Some (mset C), NE, UE, NEk, UEk, NS, US, N0, U0, Q, W)› and
    ‹C ≠ []›
    using ‹(TnC, T') ∈ ?shorter S' S› ‹1 < length C› find-decomp
    apply (cases U'; cases T'; cases S')
    by (auto simp: find-lit-of-max-level-wl-def)
```

obtain M' K $M2$ **where**
U': ‹$U' = (M', N, Some\ (mset\ C), NE, UE, NEk, UEk, NS, US, N0, U0, Q, W)$› **and**
$decomp$: ‹$(Decided\ K\ \#\ M', M2) \in set\ (get\text{-}all\text{-}ann\text{-}decomposition\ M)$› **and**
$lev\text{-}K$: ‹$get\text{-}level\ M\ K = Suc\ (get\text{-}maximum\text{-}level\ M\ (remove1\text{-}mset\ (-\ lit\text{-}of\ (hd\ M))\ (the\ (Some\ (mset\ C)))))$›
using ‹$(TnC, T') \in ?shorter\ S'\ S$› ‹$1 < length\ C$› $find\text{-}decomp$
by ($cases\ U'$; $cases\ S'$)
($auto\ simp$: $find\text{-}lit\text{-}of\text{-}max\text{-}level\text{-}wl\text{-}def\ T'\ all\text{-}atms\text{-}st\text{-}def$)

have [$simp$]:
‹$LK' = lit\text{-}of\ (hd\ (get\text{-}trail\text{-}wl\ T'))$›
‹$LK = LK'$›
using $KK'\ TT'$ **by** ($auto\ simp$: $equality\text{-}except\text{-}conflict\text{-}wl\text{-}get\text{-}trail\text{-}wl$)

have $n\text{-}d$: ‹$no\text{-}dup\ (get\text{-}trail\text{-}wl\ S')$›
using ‹$(S, S') \in ?R$›
by ($auto\ simp$: $twl\text{-}st\text{-}heur\text{-}conflict\text{-}ana\text{-}def\ trail\text{-}pol\text{-}def$)

have [$simp$]: ‹$get\text{-}trail\text{-}wl\ S' = get\text{-}trail\text{-}wl\ T'$›
using ‹$(TnC, T') \in ?shorter\ S'\ S$› ‹$1 < length\ C$› $find\text{-}decomp$
by ($cases\ T'$; $cases\ S'$; $auto\ simp$: $find\text{-}lit\text{-}of\text{-}max\text{-}level\text{-}wl\text{-}def\ U'$; $fail$)+
have [$simp$]: ‹$remove1\text{-}mset\ (-\ lit\text{-}of\ (hd\ M))\ (mset\ C) = mset\ (tl\ C)$›
apply ($subst\ mset\text{-}tl$)
using ‹$(TnC, T') \in ?shorter\ S'\ S$›
by ($auto\ simp$: $find\text{-}lit\text{-}of\text{-}max\text{-}level\text{-}wl\text{-}def\ U'\ highest\text{-}lit\text{-}def\ T'$)

have $n\text{-}d$: ‹$no\text{-}dup\ M$›
using ‹$(TnC, T') \in ?shorter\ S'\ S$› $n\text{-}d$ **unfolding** T'
by ($cases\ S'$) $auto$

have $nempty[iff]$: ‹$remove1\text{-}mset\ (-\ lit\text{-}of\ (hd\ M))\ (the\ (Some(mset\ C))) \neq \{\#\}$›
using $U'\ T'\ find\text{-}decomp\ size\text{-}C$ **by** ($cases\ C$) ($auto\ simp$: $remove1\text{-}mset\text{-}empty\text{-}iff$)
have $H[simp]$: ‹$aa \in\# remove1\text{-}mset\ (-\ lit\text{-}of\ (hd\ M))\ (the\ (Some(mset\ C))) \Longrightarrow get\text{-}level\ M'\ aa = get\text{-}level\ M\ aa$› **for** aa
apply ($rule\ get\text{-}all\text{-}ann\text{-}decomposition\text{-}get\text{-}level[of$ ‹$lit\text{-}of\ (hd\ M)$› - K - $M2$ ‹$the\ (Some(mset\ C))$›$])$
subgoal ..
subgoal by ($rule\ n\text{-}d$)
subgoal by ($rule\ decomp$)
subgoal by ($rule\ lev\text{-}K$)
subgoal by $simp$
done

have ‹$get\text{-}maximum\text{-}level\ M\ (remove1\text{-}mset\ (-\ lit\text{-}of\ (hd\ M))\ (mset\ C)) = get\text{-}maximum\text{-}level\ M'\ (remove1\text{-}mset\ (-\ lit\text{-}of\ (hd\ M))\ (mset\ C))$›
by ($rule\ get\text{-}maximum\text{-}level\text{-}cong$) $auto$
then show $?thesis$
using ‹$(TnC, T') \in ?shorter\ S'\ S$› ‹$1 < length\ C$› $hd\text{-}conv\text{-}nth[OF$ ‹$C \neq []$›, $symmetric]$
by ($auto\ simp$: $find\text{-}lit\text{-}of\text{-}max\text{-}level\text{-}wl\text{-}def\ U'\ highest\text{-}lit\text{-}def\ T'$)
qed

have $propagate\text{-}bt\text{-}wl\text{-}D\text{-}heur$: ‹$propagate\text{-}bt\text{-}wl\text{-}D\text{-}heur\ LK\ C\ U \leq \Downarrow\ ?S\ (propagate\text{-}bt\text{-}wl\ LK'\ L'\ U')$›
if
SS': ‹$(S, S') \in ?R$› **and**
‹$backtrack\text{-}wl\text{-}inv\ S'$› **and**
‹$backtrack\text{-}wl\text{-}D\text{-}heur\text{-}inv\ S$› **and**

```
    ‹(TnC, T') ∈ ?shorter S' S› and
    [simp]: ‹nC = (n, C)› and
    [simp]: ‹TnC = (T, nC)› and
    find-decomp: ‹(U, U') ∈ ?find-decomp S T' n› and
    le-C: ‹1 < length C› and
    ‹1 < size (the (get-conflict-wl U'))› and
    C-L': ‹(C ! 1, L') ∈ nat-lit-lit-rel› and
    KK': ‹(LK, LK') ∈ {(L, L'). L = L' ∧ L = lit-of (hd (get-trail-wl S'))}›
  for S S' TnC T' T nC n C U U' L' LK LK'
proof −

  have
    TT': ‹(T, del-conflict-wl T') ∈ twl-st-heur-bt› and
    n: ‹n = get-maximum-level (get-trail-wl T')
       (remove1-mset (− lit-of (hd (get-trail-wl T'))) (mset C))› and
    T-C: ‹get-conflict-wl T' = Some (mset C)› and
    T'S': ‹equality-except-conflict-wl T' S'› and
    C-nempty: ‹C ≠ []› and
    hd-C: ‹hd C = − lit-of (hd (get-trail-wl T'))› and
    highest: ‹highest-lit (get-trail-wl T') (mset (tl C))
      (Some (C ! Suc 0, get-level (get-trail-wl T') (C ! Suc 0)))› and
    incl: ‹mset C ⊆# the (get-conflict-wl S')› and
    dist-S': ‹distinct-mset (the (get-conflict-wl S'))› and
    list-confl-S': ‹literals-are-in-ℒ_in (all-atms-st S') (the (get-conflict-wl S'))› and
    ‹get-conflict-wl S' ≠ None› and
    uM-ℒ_all: ‹−lit-of (hd (get-trail-wl S')) ∈# ℒ_all (all-atms-st S')› and
    lits: ‹literals-are-ℒ_in (all-atms-st T') T'› and
    tr-nempty: ‹get-trail-wl T' ≠ []› and
    r: ‹length (get-clauses-wl-heur S) = r› ‹length (get-clauses-wl-heur T) = r›
     ‹get-learned-count T = get-learned-count S› ‹learned-clss-count S ≤ u› and
    corr: ‹correct-watching S'›
    using ‹(TnC, T') ∈ ?shorter S' S›  ‹1 < length C› ‹(S, S') ∈ ?R›
    by auto

  obtain K M2 where
    UU': ‹(U, U') ∈ twl-st-heur-bt› and
    U'U': ‹equality-except-trail-wl U' T'› and
    lev-K: ‹get-level (get-trail-wl T') K = Suc (get-maximum-level (get-trail-wl T')
        (remove1-mset (− lit-of (hd (get-trail-wl T')))
          (the (get-conflict-wl T'))))› and
     decomp: ‹(Decided K # get-trail-wl U', M2) ∈ set (get-all-ann-decomposition (get-trail-wl T'))›
and
    r': ‹length (get-clauses-wl-heur U) = r›
     ‹get-learned-count U = get-learned-count T›
     ‹learned-clss-count U ≤ u› and
    S-arena: ‹get-clauses-wl-heur U = get-clauses-wl-heur S›
    using find-decomp r get-learned-count-learned-clss-countD2[of U T]
     get-learned-count-learned-clss-countD2[of T S]
    by (auto dest: )

  obtain M N NE UE NEk UEk Q NS US N0 U0 W where
    T': ‹T' = (M, N, Some (mset C), NE, UE, NEk, UEk, NS, US, N0, U0, Q, W)› and
    ‹C ≠ []›
    using TT' T-C ‹1 < length C›
    apply (cases T'; cases S')
    by (auto simp: find-lit-of-max-level-wl-def)
```

obtain D **where**
S': ‹$S' = (M,\ N,\ D,\ NE,\ UE,\ NEk,\ UEk,\ NS,\ US,\ N0,\ U0,\ Q,\ W)$›
using $T'S'$ ‹$1 < \textit{length}\ C$›
apply (*cases* S')
by (*auto simp*: *find-lit-of-max-level-wl-def* T' *del-conflict-wl-def*)

obtain *M1* **where**
U': ‹$U' = (M1,\ N,\ \textit{Some}\ (\textit{mset}\ C),\ NE,\ UE,\ NEk,\ UEk,\ NS,\ US,\ N0,\ U0,\ Q,\ W)$› **and**
lits-confl: ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st* S') (*mset* C)› **and**
‹*mset* $C \subseteq\#$ *the* (*get-conflict-wl* S')› **and**
tauto: ‹¬ *tautology* (*mset* C)›
using ‹$(TnC,\ T') \in$ *?shorter* $S'\ S$› ‹$1 <$ *length* C› *find-decomp*
apply (*cases* U')
by (*auto simp*: *find-lit-of-max-level-wl-def* T' *intro*: *literals-are-in-*$\mathcal{L}_{in}$*-mono*)

let *?M1'* = ‹*get-trail-wl-heur* U›
let *?arena* = ‹*get-clauses-wl-heur* U›
let *?D'* = ‹*get-conflict-wl-heur* U›
let *?W'* = ‹*get-watched-wl-heur* U›
let *?vm'* = ‹*get-vmtf-heur* U›
let *?clvls* = ‹*get-count-max-lvls-heur* U›
let *?cach* = ‹*get-conflict-cach* U›
let *?outl* = ‹*get-outlearned-heur* U›
let *?lcount* = ‹*get-learned-count* U›
let *?heur* = ‹*get-heur* U›
let *?lbd* = ‹*get-lbd* U›
let *?aivdom* = ‹*get-aivdom* U›

let *?vdom* = ‹*set* (*get-vdom-aivdom ?aivdom*)›

have *old*: ‹*get-old-arena* $U = []$›
using UU' *find-decomp* **by** (*cases* U) (*auto simp*: $U'\ T'$ *twl-st-heur-bt-def all-atms-def*[*symmetric*])
have [*simp*]:
‹$LK' =$ *lit-of* (*hd* M)›
‹$LK = LK'$›
using $KK'\ SS'$ **by** (*auto simp*: *equality-except-conflict-wl-get-trail-wl* S')
have
M1'-M1: ‹(*?M1'*, *M1*) ∈ *trail-pol* (*all-atms-st* U')› **and**
$W'W$: ‹(*?W'*, W) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 (*all-atms-st* U'))› **and**
vmtf: ‹*?vm'* ∈ *bump-heur* (*all-atms-st* U') *M1*› **and**
n-d-M1: ‹*no-dup M1*› **and**
empty-cach: ‹*cach-refinement-empty* (*all-atms-st* U') *?cach*› **and**
‹*length ?outl = Suc 0*› **and**
outl: ‹*out-learned M1 None ?outl*› **and**
vdom: ‹*vdom-m* (*all-atms-st* U') $W\ N \subseteq$ *?vdom*› **and**
lcount: ‹*clss-size-corr N NE UE NEk UEk NS US N0 U0 ?lcount*› **and**
vdom-m: ‹*vdom-m* (*all-atms-st* U') $W\ N \subseteq$ *?vdom*› **and**
D': ‹(*?D'*, *None*) ∈ *option-lookup-clause-rel* (*all-atms-st* U')› **and**
valid: ‹*valid-arena ?arena N ?vdom*› **and**
aivdom: ‹*aivdom-inv-dec ?aivdom* (*dom-m N*)› **and**
bounded: ‹*isasat-input-bounded* (*all-atms-st* U')› **and**
nempty: ‹*isasat-input-nempty* (*all-atms-st* U')› **and**
dist-vdom: ‹*distinct* (*get-vdom-aivdom ?aivdom*)› **and**
heur: ‹*heuristic-rel* (*all-atms-st* U') *?heur*› **and**
occs: ‹(*get-occs* U, *empty-occs-list* (*all-atms-st* U')) ∈ *occurrence-list-ref*›
using UU' **by** (*auto simp*: *out-learned-def twl-st-heur-bt-def* U' *all-atms-def*[*symmetric*]

```
    aivdom-inv-dec-alt-def)
have [simp]: ‹C ! 1 = L'› ‹C ! 0 = - lit-of (hd M)› and
  n-d: ‹no-dup M›
  using SS' C-L' hd-C ‹C ≠ []› by (auto simp: S' U' T' twl-st-heur-conflict-ana-def hd-conv-nth)
have undef: ‹undefined-lit M1 (lit-of (hd M))›
  using decomp n-d
  by (auto dest!: get-all-ann-decomposition-exists-prepend simp: T' hd-append U' neq-Nil-conv
      split: if-splits)
have C-1-neq-hd: ‹C ! Suc 0 ≠ - lit-of (hd M)›
  using distinct-mset-mono[OF incl dist-S'] C-L' ‹1 < length C› ‹C ! 0 = - lit-of (hd M)›
  by (cases C; cases ‹tl C›) (auto simp del: ‹C ! 0 = - lit-of (hd M)›)
have H: ‹(RES ((λi. (fmupd i (C, False) N, i)) ` {i. 0 < i ∧ i ∉# dom-m N}) ⤜
            (λ(N, i). ASSERT (i ∈# dom-m N) ⤜ (λ-. f N i))) =
      (RES ((λi. (fmupd i (C, False) N, i)) ` {i. 0 < i ∧ i ∉# dom-m N}) ⤜
            (λ(N, i). f N i))› (is ‹?A = ?B›) for f C N
proof -
  have ‹?B ≤ ?A›
    by (force intro: ext complete-lattice-class.Sup-subset-mono
      simp: intro-spec-iff bind-RES)
  moreover have ‹?A ≤ ?B›
    by (force intro: ext complete-lattice-class.Sup-subset-mono
      simp: intro-spec-iff bind-RES)
  ultimately show ?thesis by auto
qed

have propagate-bt-wl-D-heur-alt-def:
  ‹propagate-bt-wl-D-heur = (λL C S. do {
      let M = get-trail-wl-heur S;
      let vdom = get-aivdom S;
      let N0 = get-clauses-wl-heur S;
      let W0 = get-watched-wl-heur S;
      let lcount = get-learned-count S;
      let heur = get-heur S;
      let stats = get-stats-heur S;
      let lbd = get-lbd S;
      let vm0 = get-vmtf-heur S;
      ASSERT(length (get-vdom-aivdom vdom) ≤ length N0);
      ASSERT(length (get-avdom-aivdom vdom) ≤ length N0);
      ASSERT(nat-of-lit (C!1) < length W0 ∧ nat-of-lit (-L) < length W0);
      ASSERT(length C > 1);
      let L' = C!1;
      ASSERT (length C ≤ unat32-max div 2 + 1);
      vm ← isa-bump-rescore C M vm0;
      glue ← get-LBD lbd;
      let - = C;
      let b = False;
      ASSERT(isasat-fast S ⟶ append-and-length-fast-code-pre ((b, C), N0));
      ASSERT(isasat-fast S ⟶ clss-size-lcount lcount < snat64-max);
      (N, i) ← fm-add-new b C N0;
      ASSERT(update-lbd-pre ((i, glue), N));
      let N = update-lbd-and-mark-used i glue N;
      ASSERT(isasat-fast S ⟶ length-ll W0 (nat-of-lit (-L)) < snat64-max);
      let W = W0[nat-of-lit (- L) := W0 ! nat-of-lit (- L) @ [(i, L', length C = 2)]];
      ASSERT(isasat-fast S ⟶ length-ll W (nat-of-lit L') < snat64-max);
      let W = W[nat-of-lit L' := W!nat-of-lit L' @ [(i, -L, length C = 2)]];
      lbd ← lbd-empty lbd;
```

```
        j ← mop-isa-length-trail M;
        ASSERT(i ≠ DECISION-REASON);
        ASSERT(cons-trail-Propagated-tr-pre ((−L, i), M));
        M ← cons-trail-Propagated-tr (− L) i M;
        vm ← isa-bump-heur-flush M vm;
        heur ← mop-save-phase-heur (atm-of L') (is-neg L') heur;
       let
          S = set-watched-wl-heur W S;
          S = set-learned-count-wl-heur (clss-size-incr-lcount lcount) S;
          S = set-aivdom-wl-heur (add-learned-clause-aivdom i vdom) S;
      S = set-heur-wl-heur (unset-fully-propagated-heur (heuristic-reluctant-tick (update-propagation-heuristics
glue heur))) S;
          S = set-stats-wl-heur (add-lbd (word-of-nat glue) stats) S; S = set-trail-wl-heur M S;
          S = set-clauses-wl-heur N S; S = set-literals-to-update-wl-heur j S;
          S = set-count-max-wl-heur 0 S; S = set-vmtf-wl-heur vm S;
          S = set-lbd-wl-heur lbd S in
         do {- ← log-new-clause-heur S i;
         S ← maybe-mark-added-clause-heur2 S i;
         RETURN S}
    })›
    unfolding propagate-bt-wl-D-heur-def Let-def
    by (auto intro!: ext bind-cong[OF refl])
  have find-new-alt: ‹SPEC
            (λ(N', i). N' = fmupd i (D'', False) N ∧ 0 < i ∧
                i ∉# dom-m N ∧
                (∀ L∈#all-lits-of-mm (mset ‘# ran-mf N + (NE + UE) + (NEk + UEk)  + (NS +
US) + (N0 + U0)).
                    i ∉ fst ‘ set (W L))) = do {

        i ← SPEC
            (λi. 0 < i ∧
                i ∉# dom-m N ∧
                (∀ L∈#all-lits-of-mm (mset ‘# ran-mf N + (NE + UE)  + (NEk + UEk) + (NS +
US) + (N0 + U0)).
                    i ∉ fst ‘ set (W L)));
        N' ← RETURN (fmupd i (D'', False) N);
        RETURN (N', i)
    }› for D''
    by (auto simp: RETURN-def RES-RES-RETURN-RES2
     RES-RES-RETURN-RES)
  have propagate-bt-wl-D-alt-def:
    ‹propagate-bt-wl LK' L' U' = do {
        ASSERT (propagate-bt-wl-pre LK' L' (M1, N, Some (mset C), NE, UE, NEk, UEk, NS, US,
N0, U0, Q, W));
        - ← RETURN (); ////////////
        - ← RETURN (); ////
        D'' ←
          list-of-mset2 (− LK') L'
           (the (Some (mset C)));
        (N, i) ← SPEC
            (λ(N', i). N' = fmupd i (D'', False) N ∧ 0 < i ∧
                i ∉# dom-m N ∧
                (∀ L∈#all-lits-of-mm (mset ‘# ran-mf N + (NE + UE) + (NEk + UEk) + (NS +
US) + (N0 + U0)).
                    i ∉ fst ‘ set (W L)));
        - ← RETURN (); ////////
```

```
      - ← RETURN (); ////bd//empty//
  M2 ← cons-trail-propagate-l (− LK′) i M1;
      - ← RETURN (); ////////flush//
      - ← RETURN (); ////
      - ← RETURN (log-clause (M2,
          N, None, NE, UE, NEk, UEk, NS, US, N0, U0, {#LK′#},
          W(− LK′ := W (− LK′) @ [(i, L′, length D′′ = 2)],
            L′ := W L′ @ [(i, − LK′, length D′′ = 2)])) i);
      let S = (M2,
          N, None, NE, UE, NEk, UEk, NS, US, N0, U0, {#LK′#},
          W(− LK′ := W (− LK′) @ [(i, L′, length D′′ = 2)],
            L′ := W L′ @ [(i, − LK′, length D′′ = 2)]));
      RETURN S
    }›
  unfolding propagate-bt-wl-def Let-def find-new-alt nres-monad3
    U′ H get-fresh-index-wl-def prod.case
    propagate-bt-wl-def Let-def rescore-clause-def
  by (auto simp: U′ RES-RES2-RETURN-RES RES-RETURN-RES uminus-A_in-iff
      uncurry-def RES-RES-RETURN-RES length-list-ge2 C-1-neq-hd
      get-fresh-index-def RES-RETURN-RES2 RES-RES-RETURN-RES2 list-of-mset2-def
      cons-trail-propagate-l-def ac-simps
      intro!: bind-cong[OF refl]
      simp flip: all-lits-alt-def2 all-lits-alt-def all-lits-def)

have [refine0]: ‹SPEC (λ(vm′). vm′ ∈ bump-heur A M1)
  ≤ ⇓{((vm′), ()). vm′ ∈ bump-heur A M1 } (RETURN ())› for A
  by (auto intro!: RES-refine simp: RETURN-def)

have [refine0]:
  ‹isa-bump-rescore C ?M1′ ?vm′ ≤ SPEC (λc. (c, ()) ∈ {((vm), -).
     vm ∈ bump-heur (all-atms-st U′) M1})›
  apply (rule order.trans)
  apply (rule isa-vmtf-rescore[of ‹all-atms-st U′›, THEN fref-to-Down-curry2, of - - - C M1 ?vm′])
  subgoal using bounded by auto
  subgoal using M1′-M1 vmtf by auto
  apply (rule order.trans)
  apply (rule ref-two-step′)
  apply (rule vmtf-rescore-score-clause[THEN fref-to-Down-curry2, of ‹all-atms-st U′› C M1 ?vm′])
  subgoal using vmtf lits-confl bounded by (auto simp: S′ U′ all-atms-st-def)
  subgoal using vmtf M1′-M1 by auto
  subgoal by (auto simp: rescore-clause-def conc-fun-RES isa-rescore-clause-def)
  done

have [refine0]: ‹isa-bump-heur-flush Ma vm ≤
    SPEC(λc. (c, ()) ∈ {(vm′, -). vm′ ∈ bump-heur (all-atms-st U′) M2})›
  if vm: ‹vm ∈ bump-heur (all-atms-st U′) M1› and
   Ma: ‹(Ma, M2)
   ∈ {(M0, M0′′).
     (M0, M0′′) ∈ trail-pol (all-atms-st U′) ∧
     M0′′ = Propagated (− L) i # M1 ∧
     no-dup M0′′}›
  for vm i L Ma M2
proof −
  let ?M1′ = ‹cons-trail-Propagated-tr L i ?M1′›
  let ?M1 = ‹Propagated (−L) i # M1›
```

```
    have M1'-M1: ‹(Ma, M2) ∈ trail-pol (all-atms-st U')›
      using Ma by auto

    have vm: ‹vm ∈ bump-heur (all-atms-st U') ?M1›
      using vm by (auto simp: dest: isa-vmtf-consD)
    show ?thesis
     apply (rule order.trans)
      apply (rule isa-bump-heur-flush-isa-bump-flush[THEN fref-to-Down-curry, of ‹all-atms-st U'› ?M1 vm])
      subgoal by (use M1'-M1 Ma bounded vm nempty in auto)
      subgoal by (use M1'-M1 Ma bounded vm nempty in auto)
      subgoal using Ma by (auto simp: isa-bump-flush-def)
     done
  qed

  have [refine0]: ‹(mop-isa-length-trail ?M1') ≤ ⇓ {(j, -). j = length M1} (RETURN ())›
   by (rule order-trans[OF mop-isa-length-trail-length-u[THEN fref-to-Down-Id-keep, OF - M1'-M1]])
      (auto simp: conc-fun-RES RETURN-def)
  have [refine0]: ‹get-LBD ?lbd ≤ ⇓ {(-, -). True}(RETURN ())›
   unfolding get-LBD-def by (auto intro!: RES-refine simp: RETURN-def)
  have [refine0]: ‹RETURN C
    ≤ ⇓ Id
      (list-of-mset2 (− LK') L'
        (the (Some (mset C))))›
    using that
    by (auto simp: list-of-mset2-def S')
  have [simp]: ‹0 < header-size D''› for D''
    by (auto simp: header-size-def)
  have [simp]: ‹length ?arena + header-size D'' ∉ ?vdom›
   ‹length ?arena + header-size D'' ∉ vdom-m (all-atms-st U') W N›
   ‹length ?arena + header-size D'' ∉# dom-m N› for D''
   using valid-arena-in-vdom-le-arena(1)[OF valid] vdom
   by (auto 5 1 simp: vdom-m-def)
  have add-new-alt-def: ‹(SPEC
        (λ(N', i).
            N' = fmupd i (D'', False) N ∧
            0 < i ∧
            i ∉# dom-m N ∧
            (∀L∈#all-lits-of-mm (mset ‘# ran-mf N + (NE + UE)  + (NEk + UEk) + (NS + US) + (N0 + U0)).
                i ∉ fst ‘ set (W L)))) =
      (SPEC
        (λ(N', i).
            N' = fmupd i (D'', False) N ∧
            0 < i ∧
            i ∉ vdom-m (all-atms-st U') W N))› for D''
    using lits
    by (auto simp: T' vdom-m-def literals-are-$\mathcal{L}_{in}$-def is-$\mathcal{L}_{all}$-def U' all-atms-def
      all-lits-st-def all-lits-def ac-simps)
  have [refine0]: ‹fm-add-new False C ?arena
    ≤ ⇓ {((arena', i), (N', i')). valid-arena arena' N' (insert i ?vdom) ∧ i = i' ∧
        i ∉# dom-m N ∧ i ∉ ?vdom ∧ length arena' = length ?arena + header-size D'' + length D''}
      (SPEC
        (λ(N', i).
            N' = fmupd i (D'', False) N ∧
            0 < i ∧
```

i ∉# *dom-m N* ∧
(∀ *L*∈#*all-lits-of-mm* (*mset* ‘# *ran-mf N* + (*NE* + *UE*) + (*NEk* + *UEk*) + (*NS* + *US*) + (*N0* + *U0*)).
i ∉ *fst* ‘ *set* (*W L*))))›
if ‹(*C*, *D''*) ∈ *Id*› **for** *D''*
apply (*subst add-new-alt-def*)
apply (*rule order-trans*)
apply (*rule fm-add-new-append-clause*)
using *that valid le-C vdom*
by (*auto simp*: *intro*!: *RETURN-RES-refine valid-arena-append-clause*)
have [*refine0*]:
‹*lbd-empty ?lbd* ≤ *SPEC* (λ*c*. (*c*, ()) ∈ {(*c*, -). *c* = *replicate* (*length ?lbd*) *False*})›
by (*auto simp*: *lbd-empty-def*)

have ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st S'*) (*mset C*)›
using *incl list-confl-S' literals-are-in-*$\mathcal{L}_{in}$*-mono* **by** *blast*
then have *C-Suc1-in*: ‹*C* ! *Suc 0* ∈# $\mathcal{L}_{all}$ (*all-atms-st S'*)›
using ‹*1* < *length C*›
by (*cases C*; *cases* ‹*tl C*›) (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-add-mset*)
then have ‹*nat-of-lit* (*C* ! *Suc 0*) < *length ?W'*› ‹*nat-of-lit* (− *lit-of* (*hd* (*get-trail-wl S'*))) < *length ?W'*› **and**
W'-eq: ‹*?W'* ! (*nat-of-lit* (*C* ! *Suc 0*)) = *W* (*C*! *Suc 0*)›
‹*?W'* ! (*nat-of-lit* (− *lit-of* (*hd* (*get-trail-wl S'*)))) = *W* (− *lit-of* (*hd* (*get-trail-wl S'*)))›
using *uM-*$\mathcal{L}_{all}$ *W'W* **unfolding** *map-fun-rel-def* **by** (*auto simp*: *image-image S' U' all-atms-st-def*)
have *le-C-ge*: ‹*length C* ≤ *unat32-max div 2* + *1*›
using *clss-size-unat32-max*[*OF bounded*, *of* ‹*mset C*›] ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st S'*) (*mset C*)› *list-confl-S'*
dist-S' incl size-mset-mono[*OF incl*] *distinct-mset-mono*[*OF incl*]
simple-clss-size-upper-div2[*OF bounded - - tauto*]
by (*auto simp*: *unat32-max-def S' U' all-atms-def*[*symmetric*] *simp*: *all-atms-st-def*)
have *tr-SS'*: ‹(*get-trail-wl-heur S*, *M*) ∈ *trail-pol* (*all-atms-st S'*)›
using ‹(*S*, *S'*) ∈ *?R*› **unfolding** *twl-st-heur-conflict-ana-def*
by (*auto simp*: *all-atms-def S'*)
let *?NUE-after* = ‹*NE* + *NEk* + *UE* + *UEk* + (*NS* + *US*) + *N0* + *U0*›
let *?NUE-before* = ‹(*NE* + *NEk* + *UE* + *UEk* + (*NS* + *US*) + *N0* + *U0*)›

have *All-atms-rew*: ‹*set-mset* (*all-atms* (*fmupd x'* (*C'*, *b*) *N*) *?NUE-before*) =
set-mset (*all-atms N ?NUE-after*)› (**is** *?A*)
‹*trail-pol* (*all-atms* (*fmupd x'* (*C'*, *b*) *N*) *?NUE-before*) =
trail-pol (*all-atms N ?NUE-after*)› (**is** *?B*)
‹*bump-heur* (*all-atms* (*fmupd x'* (*C'*, *b*) *N*) *?NUE-before*) =
bump-heur (*all-atms N ?NUE-after*)› (**is** *?C*)
‹*option-lookup-clause-rel* (*all-atms* (*fmupd x'* (*C'*, *b*) *N*) *?NUE-before*) =
option-lookup-clause-rel (*all-atms N ?NUE-after*)› (**is** *?D*)
‹⟨*Id*⟩*map-fun-rel* (*D*$_0$ (*all-atms* (*fmupd x'* (*C'*, *b*) *N*) *?NUE-before*)) =
⟨*Id*⟩*map-fun-rel* (*D*$_0$ (*all-atms N ?NUE-after*))› (**is** *?E*)
‹*set-mset* ($\mathcal{L}_{all}$ (*all-atms* (*fmupd x'* (*C'*, *b*) *N*) *?NUE-before*)) =
set-mset ($\mathcal{L}_{all}$ (*all-atms N ?NUE-after*))›
‹*phase-saving* ((*all-atms* (*fmupd x'* (*C'*, *b*) *N*) *?NUE-before*)) =
phase-saving ((*all-atms N ?NUE-after*))› (**is** *?F*)
‹*cach-refinement-empty* ((*all-atms* (*fmupd x'* (*C'*, *b*) *N*) *?NUE-before*)) =
cach-refinement-empty ((*all-atms N ?NUE-after*))› (**is** *?G*)
‹*cach-refinement-nonull* ((*all-atms* (*fmupd x'* (*C'*, *b*) *N*) *?NUE-before*)) =
cach-refinement-nonull ((*all-atms N ?NUE-after*))› (**is** *?G2*)
‹*vdom-m* ((*all-atms* (*fmupd x'* (*C'*, *b*) *N*) *?NUE-before*)) =
vdom-m ((*all-atms N ?NUE-after*))› (**is** *?H*)

‹*isasat-input-bounded* ((*all-atms* (*fmupd x′* (*C′*, *b*) *N*) *?NUE-before*)) =
isasat-input-bounded ((*all-atms N ?NUE-after*))› (**is** *?I*)
‹*isasat-input-nempty* ((*all-atms* (*fmupd x′* (*C′*, *b*) *N*) *?NUE-before*)) =
isasat-input-nempty ((*all-atms N ?NUE-after*))› (**is** *?J*)
‹*vdom-m* (*all-atms N ?NUE-before*) *W* (*fmupd x′* (*C′*, *b*) *N*) =
insert x′ (*vdom-m* (*all-atms N ?NUE-after*) *W N*)› (**is** *?K*)
‹*heuristic-rel* ((*all-atms* (*fmupd x′* (*C′*, *b*) *N*) *?NUE-before*)) =
heuristic-rel (*all-atms N ?NUE-after*)› (**is** *?L*)
‹*empty-occs-list* ((*all-atms* (*fmupd x′* (*C′*, *b*) *N*) *?NUE-before*)) =
empty-occs-list (*all-atms N ?NUE-after*)› (**is** *?M*)
if ‹*x′* ∉# *dom-m N*› **and** *C*: ‹*C′* = *C*› **for** *b x′ C′*
proof −
show *A*: *?A*
using ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st S′*) (*mset C*)› *that*
by (*auto simp*: *all-atms-def all-lits-def ran-m-mapsto-upd-notin all-lits-of-mm-add-mset*
U′ S′ in-$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *literals-are-in-*$\mathcal{L}_{in}$*-def ac-simps all-atms-st-def*)
have *A2*: ‹*set-mset* ($\mathcal{L}_{all}$ (*all-atms* (*fmupd x′* (*C*, *b*) *N*) *?NUE-before*)) =
set-mset ($\mathcal{L}_{all}$ (*all-atms N ?NUE-after*))›
using *A* **unfolding** $\mathcal{L}_{all}$*-def C* **by** (*auto simp*: *A ac-simps*)
then show ‹*set-mset* ($\mathcal{L}_{all}$ (*all-atms* (*fmupd x′* (*C′*, *b*) *N*) *?NUE-before*)) =
set-mset ($\mathcal{L}_{all}$ (*all-atms N ?NUE-after*))›
using *A* **unfolding** $\mathcal{L}_{all}$*-def C* **by** (*auto simp*: *A*)
have *A3*: ‹*set-mset* (*all-atms* (*fmupd x′* (*C*, *b*) *N*) *?NUE-before*) =
set-mset (*all-atms N ?NUE-after*)›
using *A* **unfolding** $\mathcal{L}_{all}$*-def C* **by** (*auto simp*: *A*)

show *?B* **and** *?C* **and** *?D* **and** *?E* **and** *?F* **and** *?G* **and** *?G2* **and** *?H* **and** *?I* **and** *?J* **and** *?L* **and** *?M*
unfolding *trail-pol-def A A2 ann-lits-split-reasons-def isasat-input-bounded-def*
vmtf-def distinct-atoms-rel-def vmtf-$\mathcal{L}_{all}$*-def atms-of-def*
distinct-hash-atoms-rel-def
atoms-hash-rel-def A A2 A3 C option-lookup-clause-rel-def
lookup-clause-rel-def phase-saving-def cach-refinement-empty-def
cach-refinement-def heuristic-rel-def
cach-refinement-list-def vdom-m-def
isasat-input-bounded-def
isasat-input-nempty-def cach-refinement-nonull-def
heuristic-rel-def phase-save-heur-rel-def heuristic-rel-stats-def empty-occs-list-def
isa-vmtf-cong′[*OF A*, *unfolded C*]
unfolding *trail-pol-def*[*symmetric*] *ann-lits-split-reasons-def*[*symmetric*]
isasat-input-bounded-def[*symmetric*]
vmtf-def[*symmetric*]
distinct-atoms-rel-def[*symmetric*]
vmtf-$\mathcal{L}_{all}$*-def*[*symmetric*] *atms-of-def*[*symmetric*]
distinct-hash-atoms-rel-def[*symmetric*]
atoms-hash-rel-def[*symmetric*]
option-lookup-clause-rel-def[*symmetric*]
lookup-clause-rel-def[*symmetric*]
phase-saving-def[*symmetric*] *cach-refinement-empty-def*[*symmetric*]
cach-refinement-def[*symmetric*] *cach-refinement-nonull-def*[*symmetric*]
cach-refinement-list-def[*symmetric*]
vdom-m-def[*symmetric*]
isasat-input-bounded-def[*symmetric*]
isasat-input-nempty-def[*symmetric*]
heuristic-rel-def[*symmetric*] *empty-occs-list-def*[*symmetric*]
heuristic-rel-def[*symmetric*] *phase-save-heur-rel-def*[*symmetric*] *heuristic-rel-stats-def*[*symmetric*]

apply *auto*
done
show *?K*
using *that*
by (*auto simp*: *vdom-m-simps5 vdom-m-def ac-simps*)
qed

have [*refine0*]: ‹*mop-save-phase-heur* (*atm-of* (*C* ! *1*)) (*is-neg* (*C* ! *1*)) *?heur*
≤ *SPEC*
(λ*c*. (*c*, ())
∈ {(*c*, -). *heuristic-rel* (*all-atms-st U′*) *c*})›
using *heur uM-*$\mathcal{L}_{all}$ *lits-confl le-C*
literals-are-in-$\mathcal{L}_{in}$*-in-mset-*$\mathcal{L}_{all}$[*of* ‹*all-atms-st S′*› ‹*mset C*› ‹*C*!*1*›]
unfolding *mop-save-phase-heur-def*
by (*auto intro*!: *ASSERT-leI save-phase-heur-preI simp*: *U′ S′ all-atms-st-def*)
have *stuff*: ‹*?NUE-before* = *?NUE-after*›
by *auto*
have *arena-le*: ‹*length ?arena* + *header-size C* + *length C* ≤ *MAX-HEADER-SIZE*+*1* + *r* + *unat32-max div 2*›
using *r r′ le-C-ge* **by** (*auto simp*: *unat32-max-def header-size-def S′*)
have *avdom*: ‹*mset* (*get-avdom-aivdom ?aivdom*) ⊆# *mset* (*get-vdom-aivdom ?aivdom*)› **and**
ivdom: ‹*mset* (*get-ivdom-aivdom ?aivdom*) ⊆# *mset* (*get-vdom-aivdom ?aivdom*)›
using *aivdom* **unfolding** *aivdom-inv-dec-alt-def* **by** *auto*
have *vm*: ‹*vm* ∈ *bump-heur* (*all-atms N* (*NE* + *UE*)) *M1* ⟹
vm ∈ *bump-heur* (*all-atms N* (*NE* + *UE*)) (*Propagated* (− *lit-of* (*hd M*)) *x2a* # *M1*)› **for** *x2a vm*
by (*cases vm*)
(*auto intro*!: *isa-vmtf-consD simp*:)
then show *?thesis*
supply [[*goals-limit=1*]]
using *empty-cach n-d-M1 C-L′ W′W outl vmtf undef* ‹*1* < *length C*› *lits*
uM-$\mathcal{L}_{all}$ *vdom lcount vdom-m dist-vdom heur*
apply (*subst propagate-bt-wl-D-alt-def*)
unfolding *U′ H get-fresh-index-wl-def prod.case*
propagate-bt-wl-D-heur-alt-def rescore-clause-def
apply (*rewrite* **in** ‹*let* - = -!*1 in* -› *Let-def*)
apply (*rewrite* **in** ‹*let* - = *update-lbd-and-mark-used* - - - *in* -› *Let-def*)
apply (*rewrite* **in** ‹*let* - = *list-update* - (*nat-of-lit* -) - *in* -› *Let-def*)
apply (*rewrite* **in** ‹*let* - = *list-update* - (*nat-of-lit* -) - *in* -› *Let-def*)
apply (*rewrite* **in** ‹*let* - = *False in* -› *Let-def*)
apply (*rewrite at* ‹*let* - = *get-trail-wl-heur* - *in* - ›*Let-def*)
apply (*rewrite at* ‹*let* - = *get-clauses-wl-heur* - *in* - ›*Let-def*)
apply (*rewrite at* ‹*let* - = *get-vmtf-heur* - *in* - ›*Let-def*)
apply (*rewrite at* ‹*let* - = *get-lbd* - *in* - ›*Let-def*)
apply (*rewrite at* ‹*let* - = *get-aivdom* - *in* - ›*Let-def*)
apply (*rewrite at* ‹*let* - = *get-watched-wl-heur* - *in* - ›*Let-def*)
apply (*rewrite at* ‹*let* - = *get-learned-count* - *in* - › *Let-def*)
apply (*rewrite at* ‹*let* - = *get-heur* - *in* - › *Let-def*)
apply (*rewrite at* ‹*let* - = *get-stats-heur* - *in* - › *Let-def*)
apply (*rewrite at* ‹*let* - = *set-learned-count-wl-heur* - - *in* - › *Let-def*)
apply (*rewrite at* ‹*let* - = *set-aivdom-wl-heur* - - *in* - › *Let-def*)
apply (*rewrite at* ‹*let* - = *set-heur-wl-heur* - - *in* - › *Let-def*)
apply (*rewrite at* ‹*let* - = *set-stats-wl-heur* - - *in* - › *Let-def*)
apply (*rewrite at* ‹*let* - = *set-literals-to-update-wl-heur* - - *in* - › *Let-def*)
apply (*rewrite at* ‹*let* - = *set-count-max-wl-heur* - - *in* - › *Let-def*)
apply (*rewrite at* ‹*let* - = *set-vmtf-wl-heur* - - *in* - › *Let-def*)
apply (*rewrite at* ‹*let* - = *set-lbd-wl-heur* - - *in* - › *Let-def*)

apply (*rewrite at* ‹*let - = set-clauses-wl-heur - - in -* › *Let-def*)
apply (*rewrite at* ‹*let - = set-trail-wl-heur - - in -* › *Let-def*)
apply (*rewrite at* ‹*let - = set-watched-wl-heur - - in -* › *Let-def*)
apply (*refine-rcg cons-trail-Propagated-tr2*[*of - - - - - -* ‹*all-atms-st U'*›]
)
subgoal using *valid* **by** (*auto dest*!: *valid-arena-vdom-subset*)
subgoal using *valid size-mset-mono*[*OF avdom*] **by** (*auto dest*!: *valid-arena-vdom-subset*)
subgoal using ‹*nat-of-lit* (*C* ! *Suc 0*) < *length ?W'*› **by** *simp*
subgoal using ‹*nat-of-lit* (− *lit-of* (*hd* (*get-trail-wl S'*))) < *length ?W'*›
by (*simp add*: *S' lit-of-hd-trail-def*)
subgoal using *le-C-ge* **.**
subgoal by (*auto simp*: *append-and-length-fast-code-pre-def isasat-fast-def*
snat64-max-def unat32-max-def)
subgoal
using *D' C-1-neq-hd vmtf avdom M1'-M1 size-learned-clss-dom-m*[*of N*] *valid-arena-size-dom-m-le-arena*[*OF valid*]
by (*auto simp*: *propagate-bt-wl-D-heur-def twl-st-heur-def lit-of-hd-trail-st-heur-def*
phase-saving-def atms-of-def S' U' lit-of-hd-trail-def all-atms-def[*symmetric*] *isasat-fast-def*
snat64-max-def unat32-max-def)

subgoal for *x uu x1 x2 vm uua- glue uub D'' xa x'*
by (*auto simp*: *update-lbd-pre-def arena-is-valid-clause-idx-def*)
subgoal using *length-watched-le*[*of S' S* ‹−*lit-of-hd-trail M*›] *corr SS' uM-*$\mathcal{L}_{all}$ *W'-eq S-arena*
by (*auto simp*: *isasat-fast-def length-ll-def S' lit-of-hd-trail-def simp flip*: *all-atms-def*)
subgoal using *length-watched-le*[*of S' S* ‹*C* ! *Suc 0*›] *corr SS' W'-eq S-arena C-1-neq-hd C-Suc1-in*
by (*auto simp*: *length-ll-def S' lit-of-hd-trail-def isasat-fast-def simp flip*: *all-atms-def*)
subgoal using *D' C-1-neq-hd vmtf avdom*
by (*auto simp*: *DECISION-REASON-def*
dest: *valid-arena-one-notin-vdomD*
intro!: *vm*)
subgoal
using *M1'-M1*
by (*rule cons-trail-Propagated-tr-pre*)
(*use undef uM-*$\mathcal{L}_{all}$ **in** ‹*auto simp*: *lit-of-hd-trail-def S' U' all-atms-def*[*symmetric*]
all-atms-st-def›)
subgoal using *M1'-M1* **by** (*auto simp*: *lit-of-hd-trail-def S' U' all-atms-def*[*symmetric*])
subgoal using *uM-*$\mathcal{L}_{all}$ **by** (*auto simp*: *S' U' uminus-*$\mathcal{A}_{in}$*-iff lit-of-hd-trail-def all-atms-st-def*)
subgoal
using *D' C-1-neq-hd vmtf avdom*
by (*auto simp*: *propagate-bt-wl-D-heur-def twl-st-heur-def lit-of-hd-trail-st-heur-def*
intro!: *ASSERT-refine-left ASSERT-leI RES-refine exI*[*of - C*] *valid-arena-update-lbd-and-mark-used*
dest: *valid-arena-one-notin-vdomD*
intro!: *vm*)
apply *assumption*
apply (*rule log-new-clause-heur-log-clause*)
subgoal *final-rel*
supply *All-atms-rew*[*simp*]
unfolding *twl-st-heur-def*
using *D' C-1-neq-hd vmtf avdom aivdom M1'-M1 bounded nempty r r' arena-le*
set-mset-mono[*OF ivdom*] *occs*
by (*clarsimp-all simp add*: *propagate-bt-wl-D-heur-def twl-st-heur-def*
Let-def T' U' rescore-clause-def S' map-fun-rel-def
list-of-mset2-def vmtf-flush-def RES-RES2-RETURN-RES RES-RETURN-RES uminus-$\mathcal{A}_{in}$*-iff*
get-fresh-index-def RES-RETURN-RES2 RES-RES-RETURN-RES2 lit-of-hd-trail-def
RES-RES-RETURN-RES lbd-empty-def get-LBD-def DECISION-REASON-def
all-atms-def[*symmetric*] *All-atms-rew learned-clss-count-def all-atms-st-def*

```
      aivdom-inv-dec-intro-add-mset valid-arena-update-lbd-and-mark-used old clss-size-corr-intro(2)
      intro!: valid-arena-update-lbd-and-mark-used aivdom-inv-intro-add-mset
      simp del: isasat-input-bounded-def isasat-input-nempty-def
    dest: valid-arena-one-notin-vdomD
      get-learned-count-learned-clss-countD)
    (auto
    intro!: valid-arena-update-lbd-and-mark-used aivdom-inv-intro-add-mset
    simp: vdom-m-simps5
      simp del: isasat-input-bounded-def isasat-input-nempty-def
     dest: valid-arena-one-notin-vdomD)
  subgoal by auto
  subgoal by auto
  apply (rule maybe-mark-added-clause-heur2-id[unfolded conc-fun-RETURN])
  subgoal
    apply (drule final-rel)
    apply assumption+
    done
  subgoal by auto
  subgoal
    supply All-atms-rew[simp]
    unfolding twl-st-heur-def
    using D' C-1-neq-hd vmtf avdom aivdom M1'-M1 bounded nempty r r' arena-le
      set-mset-mono[OF ivdom]
    by (clarsimp-all simp add: propagate-bt-wl-D-heur-def twl-st-heur-def
        Let-def T' U' rescore-clause-def S' map-fun-rel-def
       list-of-mset2-def vmtf-flush-def RES-RES2-RETURN-RES RES-RETURN-RES uminus-A_in-iff
        get-fresh-index-def RES-RETURN-RES2 RES-RES-RETURN-RES2 lit-of-hd-trail-def
        RES-RES-RETURN-RES lbd-empty-def get-LBD-def DECISION-REASON-def
        all-atms-def[symmetric] All-atms-rew learned-clss-count-def all-atms-st-def
       aivdom-inv-dec-intro-add-mset valid-arena-update-lbd-and-mark-used old clss-size-corr-intro(2)
        intro!: valid-arena-update-lbd-and-mark-used aivdom-inv-intro-add-mset
        simp del: isasat-input-bounded-def isasat-input-nempty-def
      dest: valid-arena-one-notin-vdomD
        get-learned-count-learned-clss-countD)
  done
qed


have propagate-unit-bt-wl-D-int: ‹propagate-unit-bt-wl-D-int LK U
  ≤ ⇓ ?S
    (propagate-unit-bt-wl LK' U')›
 if
  SS': ‹(S, S') ∈ ?R› and
  ‹backtrack-wl-inv S'› and
  ‹backtrack-wl-D-heur-inv S› and
  ‹(TnC, T') ∈ ?shorter S' S› and
  [simp]: ‹nC = (n, C)› and
  [simp]: ‹TnC = (T, nC)› and
  find-decomp: ‹(U, U') ∈ ?find-decomp S T' n› and
  ‹¬ 1 < length C› and
  ‹¬ 1 < size (the (get-conflict-wl U'))› and
  KK': ‹(LK, LK') ∈ {(L, L'). L = L' ∧ L = lit-of (hd (get-trail-wl S'))}›
 for S S' TnC T' T nC n C U U' LK LK'
proof −
 have
  TT': ‹(T, del-conflict-wl T') ∈ twl-st-heur-bt› and
```

n: ‹*n* = *get-maximum-level* (*get-trail-wl* *T'*)
(*remove1-mset* (− *lit-of* (*hd* (*get-trail-wl* *T'*))) (*mset* *C*))› **and**
T-C: ‹*get-conflict-wl* *T'* = *Some* (*mset* *C*)› **and**
T'S': ‹*equality-except-conflict-wl* *T'* *S'*› **and**
‹*C* ≠ []› **and**
hd-C: ‹*hd* *C* = − *lit-of* (*hd* (*get-trail-wl* *T'*))› **and**
incl: ‹*mset* *C* ⊆# *the* (*get-conflict-wl* *S'*)› **and**
dist-S': ‹*distinct-mset* (*the* (*get-conflict-wl* *S'*))› **and**
list-confl-S': ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-atms-st* *S'*) (*the* (*get-conflict-wl* *S'*))› **and**
‹*get-conflict-wl* *S'* ≠ *None*› **and**
‹*C* ≠ []› **and**
uL-M: ‹− *lit-of* (*hd* (*get-trail-wl* *S'*)) ∈# $\mathcal{L}_{all}$ (*all-atms-st* *S'*)› **and**
tr-nempty: ‹*get-trail-wl* *T'* ≠ []›
using ‹(*TnC*, *T'*) ∈ *?shorter* *S'* *S*› ‹~1 < *length* *C*›
by (*auto*)
obtain *K* *M2* **where**
UU': ‹(*U*, *U'*) ∈ *twl-st-heur-bt*› **and**
U'U': ‹*equality-except-trail-wl* *U'* *T'*› **and**
lev-K: ‹*get-level* (*get-trail-wl* *T'*) *K* = *Suc* (*get-maximum-level* (*get-trail-wl* *T'*)
(*remove1-mset* (− *lit-of* (*hd* (*get-trail-wl* *T'*)))
(*the* (*get-conflict-wl* *T'*))))› **and**
decomp: ‹(*Decided* *K* # *get-trail-wl* *U'*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*get-trail-wl* *T'*))›
and
r: ‹*length* (*get-clauses-wl-heur* *S*) = *r*›
using *find-decomp* *SS'*
by (*auto*)

obtain *M* *N* *NE* *UE* *NEk* *UEk* *NS* *US* *N0* *U0* *Q* *W* **where**
T': ‹*T'* = (*M*, *N*, *Some* (*mset* *C*), *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*)›
using *TT'* *T-C* ‹¬1 < *length* *C*›
apply (*cases* *T'*; *cases* *S'*)
by (*auto simp*: *find-lit-of-max-level-wl-def*)
obtain *D'* **where**
S': ‹*S'* = (*M*, *N*, *D'*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*)›
using *T'S'*
apply (*cases* *S'*)
by (*auto simp*: *find-lit-of-max-level-wl-def* *T'* *del-conflict-wl-def*)

obtain *M1* **where**
U': ‹*U'* = (*M1*, *N*, *Some* (*mset* *C*), *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*)›
using ‹(*TnC*, *T'*) ∈ *?shorter* *S'* *S*› *find-decomp*
apply (*cases* *U'*)
by (*auto simp*: *find-lit-of-max-level-wl-def* *T'*)
have [*simp*]:
‹*LK'* = *lit-of* (*hd* (*get-trail-wl* *T'*))›
‹*LK* = *LK'*›
using *KK'* *SS'* *S'* **by** (*auto simp*: *T'*)
let *?M1'* = ‹*get-trail-wl-heur* *U*›
let *?arena* = ‹*get-clauses-wl-heur* *U*›
let *?D'* = ‹*get-conflict-wl-heur* *U*›
let *?W'* = ‹*get-watched-wl-heur* *U*›
let *?vm'* = ‹*get-vmtf-heur* *U*›
let *?clvls* = ‹*get-count-max-lvls-heur* *U*›
let *?cach* = ‹*get-conflict-cach* *U*›
let *?outl* = ‹*get-outlearned-heur* *U*›
let *?lcount* = ‹*get-learned-count* *U*›

```
    let ?heur = ‹get-heur U›
    let ?lbd = ‹get-lbd U›
    let ?aivdom = ‹get-aivdom U›
    have
        r': ‹length (get-clauses-wl-heur U) = r›
            ‹get-learned-count U = get-learned-count S›
            ‹learned-clss-count U ≤ u› and
      old: ‹get-old-arena U = []›
      using SS' UU' find-decomp r ‹(TnC, T') ∈ ?shorter S' S›
        get-learned-count-learned-clss-countD2[of U S]
      by (auto simp: U' T' twl-st-heur-bt-def)
    let ?vdom = ‹set (get-vdom-aivdom ?aivdom)›
    have
      M'M: ‹(?M1', M1) ∈ trail-pol (all-atms-st U')› and
      W'W: ‹(?W', W) ∈ ⟨Id⟩map-fun-rel (D₀ (all-atms-st U'))› and
      vmtf: ‹?vm' ∈ bump-heur (all-atms-st U') M1› and
      n-d-M1: ‹no-dup M1› and
      empty-cach: ‹cach-refinement-empty (all-atms-st U') ?cach› and
      ‹length ?outl = Suc 0› and
      outl: ‹out-learned M1 None ?outl› and
      lcount: ‹clss-size-corr N NE UE NEk UEk NS US N0 U0 ?lcount› and
      vdom: ‹vdom-m (all-atms-st U') W N ⊆ ?vdom› and
      valid: ‹valid-arena ?arena N ?vdom› and
      D': ‹(?D', None) ∈ option-lookup-clause-rel (all-atms-st U')› and
      bounded: ‹isasat-input-bounded (all-atms-st U')› and
      nempty: ‹isasat-input-nempty (all-atms-st U')› and
      dist-vdom: ‹distinct (get-vdom-aivdom ?aivdom)› and
      aivdom: ‹aivdom-inv-dec ?aivdom (dom-m N)› and
      heur: ‹heuristic-rel (all-atms-st U') ?heur› and
      occs: ‹(get-occs U, empty-occs-list (all-atms-st U')) ∈ occurrence-list-ref›
      using UU' by (auto simp: out-learned-def twl-st-heur-bt-def U' all-atms-def[symmetric]
        aivdom-inv-dec-alt-def)
    have [simp]: ‹C ! 0 = - lit-of (hd M)› and
      n-d: ‹no-dup M›
      using SS' hd-C ‹C ≠ []› by (auto simp: S' U' T' twl-st-heur-conflict-ana-def hd-conv-nth)
    have undef: ‹undefined-lit M1 (lit-of (hd M))›
      using decomp n-d
      by (auto dest!: get-all-ann-decomposition-exists-prepend simp: T' hd-append U' neq-Nil-conv
          split: if-splits)
    have C: ‹C = [- lit-of (hd M)]›
      using ‹C ≠ []› ‹C ! 0 = - lit-of (hd M)› ‹¬1 < length C›
      by (cases C) (auto simp del: ‹C ! 0 = - lit-of (hd M)›)
    have propagate-unit-bt-wl-alt-def:
      ‹propagate-unit-bt-wl = (λL (M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, Q, W). do {
        ASSERT(L ∈# all-lits-st (M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, Q, W));
        ASSERT(propagate-unit-bt-wl-pre L (M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, Q, W));
- ← RETURN ();
- ← RETURN ();
- ← RETURN ();
- ← RETURN ();
M ← cons-trail-propagate-l (-L) 0 M;
        RETURN (M, N, None, NE, UE, NEk, add-mset (the D) UEk, NS, US, N0, U0, {#L#}, W)
      })›
      unfolding propagate-unit-bt-wl-def Let-def by (auto intro!: ext bind-cong[OF refl]
       simp: propagate-unit-bt-wl-pre-def propagate-unit-bt-l-pre-def
         single-of-mset-def RES-RETURN-RES image-iff)
```

have [*refine0*]:
‹*lbd-empty ?lbd* ≤ *SPEC* (λ*c*. (*c*, ()) ∈ {(*c*, -). *c* = *replicate* (*length ?lbd*) *False*})›
by (*auto simp*: *lbd-empty-def*)
have [*refine0*]: ‹*mop-isa-length-trail ?M1'* ≤ ⇓ {(*j*, -). *j* = *length M1*} (*RETURN* ())›
by (*rule order-trans*, *rule mop-isa-length-trail-length-u*[*THEN fref-to-Down-Id-keep*, *OF* - *M'M*])
(*auto simp*: *RETURN-def conc-fun-RES*)

have [*refine0*]: ‹*isa-bump-heur-flush ?M1' ?vm'* ≤
SPEC(λ*c*. (*c*, ()) ∈ {(*vm'*, -). *vm'* ∈ *bump-heur* (*all-atms-st U'*) *M1*})›
for *vm i L*
proof −
show *?thesis*
apply (*rule isa-bump-heur-flush-isa-bump-flush*[*THEN fref-to-Down-curry*, *of* ‹*all-atms-st U'*› *M1 ?vm'*, *THEN order-trans*])
subgoal by (*use M'M bounded nempty vmtf* **in** *auto*)
subgoal by (*use M'M bounded nempty* **in** *auto*)
subgoal using *M'M* **by** (*auto simp*: *isa-bump-flush-def*)
done
qed
have [*refine0*]: ‹*get-LBD ?lbd* ≤ *SPEC*(λ*c*. (*c*, ()) ∈ *UNIV*)›
by (*auto simp*: *get-LBD-def*)

have *tr-S*: ‹(*get-trail-wl-heur S*, *M*) ∈ *trail-pol* (*all-atms-st S'*)›
using *SS'* **by** (*auto simp*: *S' twl-st-heur-conflict-ana-def all-atms-def*)

have *hd-SM*: ‹*lit-of-last-trail-pol* (*get-trail-wl-heur S*) = *lit-of* (*hd M*)›
unfolding *lit-of-hd-trail-def lit-of-hd-trail-st-heur-def*
by (*subst lit-of-last-trail-pol-lit-of-last-trail*[*THEN fref-to-Down-unRET-Id*])
(*use M'M tr-S tr-nempty* **in** ‹*auto simp*: *lit-of-hd-trail-def T' S'*›)
have *uL-M*: ‹− *lit-of* (*hd* (*get-trail-wl S'*)) ∈# $\mathcal{L}_{all}$ (*all-atms-st U'*)›
using *uL-M* **by** (*simp add*: *S' U' all-atms-st-def*)
let *?NE* = ‹*add-mset* {#− *lit-of* (*hd M*)#} (*NE* + *NEk* + *UE* + *UEk* + (*NS* + *US*) + *N0* + *U0*)›
let *?NE-after* = ‹(*NE* + *NEk* + *UE* + *UEk* + (*NS* + *US*) + *N0* + *U0*)›
have *All-atms-rew*: ‹*set-mset* (*all-atms* (*N*) (*?NE*)) =
set-mset (*all-atms N ?NE-after*)› (**is** *?A*)
‹*trail-pol* (*all-atms* (*N*) (*?NE*)) =
trail-pol (*all-atms N ?NE-after*)› (**is** *?B*)
‹*bump-heur* (*all-atms* (*N*) (*?NE*)) =
bump-heur (*all-atms N ?NE-after*)› (**is** *?C*)
‹*option-lookup-clause-rel* (*all-atms* (*N*) (*?NE*)) =
option-lookup-clause-rel (*all-atms N ?NE-after*)› (**is** *?D*)
‹⟨*Id*⟩*map-fun-rel* (*D*$_0$ (*all-atms* (*N*) (*?NE*))) =
⟨*Id*⟩*map-fun-rel* (*D*$_0$ (*all-atms N ?NE-after*))› (**is** *?E*)
‹*set-mset* ($\mathcal{L}_{all}$ (*all-atms* (*N*) (*?NE*))) =
set-mset ($\mathcal{L}_{all}$ (*all-atms N ?NE-after*))›
‹*phase-saving* ((*all-atms* (*N*) (*?NE*))) =
phase-saving ((*all-atms N ?NE-after*))› (**is** *?F*)
‹*cach-refinement-empty* ((*all-atms* (*N*) (*?NE*))) =
cach-refinement-empty ((*all-atms N ?NE-after*))› (**is** *?G*)
‹*vdom-m* ((*all-atms* (*N*) (*?NE*))) =
vdom-m ((*all-atms N ?NE-after*))› (**is** *?H*)
‹*isasat-input-bounded* ((*all-atms* (*N*) (*?NE*))) =
isasat-input-bounded ((*all-atms N ?NE-after*))› (**is** *?I*)
‹*isasat-input-nempty* ((*all-atms* (*N*) (*?NE*))) =
isasat-input-nempty ((*all-atms N ?NE-after*))› (**is** *?J*)

```
    ‹vdom-m (all-atms N ?NE) W (N) =
      (vdom-m (all-atms N ?NE-after) W N)› (is ?K)
    ‹heuristic-rel ((all-atms (N) (?NE))) =
    heuristic-rel ((all-atms N ?NE-after))› (is ?L)
    ‹empty-occs-list ((all-atms (N) (?NE))) =
       empty-occs-list ((all-atms N ?NE-after))› (is ?M)
    for b x' C'
  proof −
    show A: ?A
      using uL-M
      apply (cases ‹hd M›)
      by (auto simp: all-atms-def all-lits-def ran-m-mapsto-upd-notin all-lits-of-mm-add-mset
          U' S' in-\mathcal{L}_{all}-atm-of-\mathcal{A}_{in} literals-are-in-\mathcal{L}_{in}-def atm-of-eq-atm-of
          all-lits-of-m-add-mset ac-simps lits-of-def all-atms-st-def)
    have A2: ‹set-mset (\mathcal{L}_{all} (all-atms N (?NE))) =
      set-mset (\mathcal{L}_{all} (all-atms N ?NE-after))›
      using A unfolding \mathcal{L}_{all}-def C by (auto simp: A)
    then show ‹set-mset (\mathcal{L}_{all} (all-atms (N) (?NE))) =
      set-mset (\mathcal{L}_{all} (all-atms N ?NE-after))›
      using A unfolding \mathcal{L}_{all}-def C by (auto simp: A)
    have A3: ‹set-mset (all-atms N (?NE)) =
      set-mset (all-atms N ?NE-after)›
      using A unfolding \mathcal{L}_{all}-def C by (auto simp: A)

    show ?B and ?C and ?D and ?E and ?F and ?G and ?H and ?I and ?J and ?K and ?L
and ?M
      unfolding trail-pol-def A A2 ann-lits-split-reasons-def isasat-input-bounded-def
        vmtf-def distinct-atoms-rel-def vmtf-\mathcal{L}_{all}-def atms-of-def
        distinct-hash-atoms-rel-def heuristic-rel-stats-def
        atoms-hash-rel-def A A2 A3 C option-lookup-clause-rel-def
        lookup-clause-rel-def phase-saving-def cach-refinement-empty-def
        cach-refinement-def
        cach-refinement-list-def vdom-m-def
        isasat-input-bounded-def heuristic-rel-def
        isasat-input-nempty-def cach-refinement-nonull-def vdom-m-def
        phase-save-heur-rel-def phase-saving-def empty-occs-list-def
        isa-vmtf-cong'[OF A, unfolded C]
      unfolding trail-pol-def[symmetric] ann-lits-split-reasons-def[symmetric]
        isasat-input-bounded-def[symmetric]
        vmtf-def[symmetric]
        distinct-atoms-rel-def[symmetric]
        vmtf-\mathcal{L}_{all}-def[symmetric] atms-of-def[symmetric]
        distinct-hash-atoms-rel-def[symmetric]
        atoms-hash-rel-def[symmetric]
        option-lookup-clause-rel-def[symmetric]
        lookup-clause-rel-def[symmetric]
        phase-saving-def[symmetric] cach-refinement-empty-def[symmetric]
        cach-refinement-def[symmetric]
        cach-refinement-list-def[symmetric]
        vdom-m-def[symmetric]
        isasat-input-bounded-def[symmetric] cach-refinement-nonull-def[symmetric]
      isasat-input-nempty-def[symmetric] heuristic-rel-def[symmetric] heuristic-rel-stats-def[symmetric]
        phase-save-heur-rel-def[symmetric] phase-saving-def[symmetric] empty-occs-list-def[symmetric]
        bump-heur-def[symmetric]
      apply auto
      done
```

qed
have *stuff*: ‹(*NE*+*NEk*) + (*UE*+*UEk*) + (*NS* + *US*) + *N0* + *U0* = *?NE-after*›
by *auto*
have [*simp*]: ‹*clss-size-corr N NE* (*add-mset C UE*) *NEk UEk NS US N0 U0*
(*clss-size-incr-lcountUE* (*get-learned-count S*))› **for** *C*
using *lcount UU' r'* **unfolding** *U'*
by (*cases S*)
(*simp add*: *twl-st-heur-bt-def clss-size-corr-intro*)

show *?thesis*
using *empty-cach n-d-M1 W'W outl vmtf C undef uL-M vdom lcount valid D' aivdom*
unfolding *U' propagate-unit-bt-wl-D-int-def prod.simps hd-SM*
propagate-unit-bt-wl-alt-def
apply (*rewrite at* ‹*let* - = *incr-units-since-last-GC* (*incr-uset* -) *in* -› *Let-def*)
apply (*rewrite at* ‹*let* - = *get-trail-wl-heur* - *in* - ›*Let-def*)
apply (*rewrite at* ‹*let* - = *get-clauses-wl-heur* - *in* - ›*Let-def*)
apply (*rewrite at* ‹*let* - = *get-vmtf-heur* - *in* - ›*Let-def*)
apply (*rewrite at* ‹*let* - = *get-lbd* - *in* - ›*Let-def*)
apply (*rewrite at* ‹*let* - = *get-aivdom* - *in* - ›*Let-def*)
apply (*rewrite at* ‹*let* - = *get-watched-wl-heur* - *in* - ›*Let-def*)
apply (*rewrite at* ‹*let* - = *get-learned-count* - *in* - › *Let-def*)
apply (*rewrite at* ‹*let* - = *get-heur* - *in* - › *Let-def*)
apply (*rewrite at* ‹*let* - = *get-stats-heur* - *in* - › *Let-def*)
apply (*refine-rcg cons-trail-Propagated-tr2*[**where** $\mathcal{A}$ = ‹*all-atms-st U'*›])
subgoal by (*auto simp*: *DECISION-REASON-def*)
subgoal
using *M'M* **by** (*rule cons-trail-Propagated-tr-pre*)
(*use undef uL-M* **in** ‹*auto simp*: *hd-SM all-atms-def*[*symmetric*] *T'*
lit-of-hd-trail-def S'›)
subgoal
using *M'M* **by** (*auto simp*: *U' lit-of-hd-trail-st-heur-def RETURN-def*
single-of-mset-def vmtf-flush-def twl-st-heur-def lbd-empty-def get-LBD-def
RES-RES2-RETURN-RES RES-RETURN-RES S' uminus-$\mathcal{A}_{in}$*-iff RES-RES-RETURN-RES*
DECISION-REASON-def hd-SM lit-of-hd-trail-st-heur-def
intro!: *ASSERT-refine-left RES-refine exI*[*of* - ‹−*lit-of* (*hd M*)›]
intro!: *vmtf-consD*
simp del: *isasat-input-bounded-def isasat-input-nempty-def*)
subgoal
by (*auto simp*: *U' lit-of-hd-trail-st-heur-def RETURN-def*
single-of-mset-def vmtf-flush-def twl-st-heur-def lbd-empty-def get-LBD-def
RES-RES2-RETURN-RES RES-RETURN-RES S' uminus-$\mathcal{A}_{in}$*-iff RES-RES-RETURN-RES*
DECISION-REASON-def hd-SM T'
intro!: *ASSERT-refine-left RES-refine exI*[*of* - ‹−*lit-of* (*hd M*)›]
intro!: *vmtf-consD*
simp del: *isasat-input-bounded-def isasat-input-nempty-def*)
subgoal
using *bounded nempty dist-vdom r' heur aivdom occs*
by (*auto simp*: *U' lit-of-hd-trail-st-heur-def RETURN-def*
single-of-mset-def vmtf-flush-def twl-st-heur-def lbd-empty-def get-LBD-def
RES-RES2-RETURN-RES RES-RETURN-RES S' uminus-$\mathcal{A}_{in}$*-iff RES-RES-RETURN-RES*
learned-clss-count-def all-atms-st-def old
DECISION-REASON-def hd-SM All-atms-rew all-atms-def[*symmetric*]
intro!: *ASSERT-refine-left RES-refine exI*[*of* - ‹−*lit-of* (*hd M*)›]
intro!: *isa-vmtf-consD*
simp del: *isasat-input-bounded-def isasat-input-nempty-def*)
done

qed

have *trail-nempty*: ‹*fst* (*get-trail-wl-heur S*) ≠ []›
if
‹(*S*, *S'*) ∈ *?R*› **and**
‹*backtrack-wl-inv S'*›
for *S S'*
proof −
show *?thesis*
using *that* **unfolding** *backtrack-wl-inv-def backtrack-wl-D-heur-inv-def backtrack-l-inv-def backtrack-inv-def*
backtrack-l-inv-def **apply** −
by *normalize-goal+*
(*auto simp*: *twl-st-heur-conflict-ana-def trail-pol-def ann-lits-split-reasons-def*)
qed

have *H*:
‹(*x*, *y*) ∈ *twl-st-heur-conflict-ana* ⟹ *fst* (*get-trail-wl-heur x*) ≠ [] ⟷ *get-trail-wl y* ≠ []› **for** *x y*
by (*auto simp*: *twl-st-heur-conflict-ana-def trail-pol-def ann-lits-split-reasons-def*)
have [*refine*]: ‹⋀*x y*. (*x*, *y*)
∈ {(*S*, *T*).
(*S*, *T*) ∈ *twl-st-heur-conflict-ana* ∧
length (*get-clauses-wl-heur S*) = *r*} ⟹
lit-of-hd-trail-st-heur x
≤ ⇓ {(*L*, *L'*). *L* = *L'* ∧ *L* = *lit-of* (*hd* (*get-trail-wl y*))} (*mop-lit-hd-trail-wl y*)›
unfolding *mop-lit-hd-trail-wl-def lit-of-hd-trail-st-heur-def*
apply *refine-rcg*
subgoal for *x y*
unfolding *mop-lit-hd-trail-wl-pre-def mop-lit-hd-trail-l-pre-def mop-lit-hd-trail-pre-def in-pair-collect-simp*
by *normalize-goal+*
(*simp add*: *H*)
subgoal for *x y*
unfolding *mop-lit-hd-trail-wl-pre-def mop-lit-hd-trail-l-pre-def mop-lit-hd-trail-pre-def in-pair-collect-simp*
apply *normalize-goal+*
apply (*subst lit-of-last-trail-pol-lit-of-last-trail*[*THEN fref-to-Down-unRET-Id*, *of* ‹*get-trail-wl y*› ‹*get-trail-wl-heur x*› ‹*all-atms-st y*›])
apply (*simp-all add*: *H*)
apply (*auto simp*: *twl-st-heur-conflict-ana-def mop-lit-hd-trail-wl-pre-def mop-lit-hd-trail-l-pre-def mop-lit-hd-trail-pre-def state-wl-l-def twl-st-l-def lit-of-hd-trail-def RETURN-RES-refine-iff*)
done
done
have *backtrack-wl-alt-def*:
‹*backtrack-wl S* =
do {
ASSERT(*backtrack-wl-inv S*);
L ← *mop-lit-hd-trail-wl S*;
S ← *extract-shorter-conflict-wl S*;
S ← *find-decomp-wl L S*;

if size (*the* (*get-conflict-wl S*)) > *1*
then do {
L' ← *find-lit-of-max-level-wl S L*;
S ← *propagate-bt-wl L L' S*;
RETURN S
}
else do {

```
        propagate-unit-bt-wl L S
      }
    }› for S
    unfolding backtrack-wl-def while.imonad2
    by auto

  have save-phase-st: ‹(xb, x') ∈ ?S ⟹
      save-phase-st xb
      ≤ SPEC
        (λc. (c, x')
            ∈ {(S, T).
              (S, T) ∈ twl-st-heur ∧
              length (get-clauses-wl-heur S)
              ≤ MAX-HEADER-SIZE+1 + r + unat32-max div 2 ∧
              learned-clss-count S ≤ Suc u})› for xb x'
    unfolding save-phase-st-def
    by (refine-vcg save-phase-heur-spec[THEN order-trans, of ‹all-atms-st x'›])
      (auto simp: twl-st-heur-def learned-clss-count-def)
  show ?thesis
    supply [[goals-limit=1]]
    apply (intro frefI nres-relI)
    unfolding backtrack-wl-D-nlit-heur-alt-def backtrack-wl-alt-def
    apply (refine-rcg shorter)
    subgoal by (rule inv)
    subgoal by (rule trail-nempty)
    subgoal by auto
    subgoal for x y xa S x1 x2 x1a x2a
      by (auto simp: twl-st-heur-state-simp equality-except-conflict-wl-get-clauses-wl)
    subgoal for x y xa S x1 x2 x1a x2a
      by (auto simp: twl-st-heur-state-simp equality-except-conflict-wl-get-clauses-wl)
    apply (rule find-decomp-wl-nlit; assumption)
    subgoal by (auto simp: twl-st-heur-state-simp equality-except-conflict-wl-get-clauses-wl
        equality-except-trail-wl-get-clauses-wl)
    subgoal by (auto simp: twl-st-heur-state-simp equality-except-conflict-wl-get-clauses-wl
        equality-except-trail-wl-get-clauses-wl)
    subgoal for x y L La xa S x1 x2 x1a x2a Sa Sb
      by (auto simp: twl-st-heur-state-simp equality-except-trail-wl-get-conflict-wl)
    apply (rule fst-find-lit-of-max-level-wl; solves assumption)
    apply (rule propagate-bt-wl-D-heur; assumption)
    apply (rule save-phase-st; assumption)
    apply (rule propagate-unit-bt-wl-D-int; assumption)
    done
qed


end
theory IsaSAT-VMTF-State-LLVM
  imports IsaSAT-VMTF-LLVM IsaSAT-Setup-LLVM
begin
hide-const (open) NEMonad.ASSERT NEMonad.RETURN

lemma find-decomp-wl-st-int-alt-def:
  ‹find-decomp-wl-st-int = (λhighest S. do{
    let (M, S) = extract-trail-wl-heur S;
    let (vm, S) = extract-vmtf-wl-heur S;
    (M', vm) ← isa-find-decomp-wl-imp M highest vm;
```

```
    let S = update-trail-wl-heur M' S;
    let S = update-vmtf-wl-heur vm S;
    RETURN S
  })›
  by (auto simp: find-decomp-wl-st-int-def state-extractors intro!: ext split: isasat-int-splits)

sepref-def find-decomp-wl-imp'-fast-code
  is ‹uncurry find-decomp-wl-st-int›
  :: ‹uint32-nat-assn^k *_a isasat-bounded-assn^d →_a
      isasat-bounded-assn›
  unfolding find-decomp-wl-st-int-alt-def PR-CONST-def
  supply [[goals-limit = 1]]
  by sepref

experiment begin

export-llvm
  find-decomp-wl-imp'-fast-code

end

end
theory IsaSAT-Proofs-LLVM
  imports IsaSAT-Proofs IsaSAT-Setup-LLVM
begin

hide-const (open) NEMonad.ASSERT NEMonad.RETURN

sepref-def log-literal-impl
  is ‹RETURN o log-literal›
  :: ‹unat-lit-assn^k →_a unit-assn›
  unfolding log-literal-def
  by sepref


sepref-def log-start-new-clause-impl
  is ‹RETURN o log-start-new-clause›
  :: ‹unat64-assn^k →_a unit-assn›
  unfolding log-start-new-clause-def
  by sepref

sepref-def log-start-del-clause-impl
  is ‹RETURN o log-start-del-clause›
  :: ‹unat64-assn^k →_a unit-assn›
  unfolding log-start-del-clause-def
  by sepref

sepref-def log-end-clause-impl
  is ‹RETURN o log-end-clause›
  :: ‹unat64-assn^k →_a unit-assn›
  unfolding log-end-clause-def
  by sepref

sepref-def log-clause-heur-impl
  is ‹uncurry log-clause-heur›
  :: ‹[λ(S, C). length (get-clauses-wl-heur S) ≤ snat64-max]_a isasat-bounded-assn^k *_a snat64-assn^k →
```

unit-assn›
unfolding *log-clause-heur-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *log-new-clause-heur-impl*
is ‹*uncurry log-new-clause-heur*›
:: ‹*isasat-bounded-assn*^{k} $*_a$ *snat64-assn*^{k} $\rightarrow_a$ *unit-assn*›
supply [*dest*] = *isasat-bounded-assn-length-arenaD*
unfolding *log-new-clause-heur-def*
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *log-del-clause-heur-impl*
is ‹*uncurry log-del-clause-heur*›
:: ‹*isasat-bounded-assn*^{k} $*_a$ *snat64-assn*^{k} $\rightarrow_a$ *unit-assn*›
supply [*dest*] = *isasat-bounded-assn-length-arenaD*
unfolding *log-del-clause-heur-def*
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-register *log-del-clause-heur log-new-clause-heur-impl log-unit-clause log-del-binary-clause*

sepref-def *log-unit-clause-impl*
is ‹*RETURN o log-unit-clause*›
:: ‹*unat-lit-assn*^{k} $\rightarrow_a$ *unit-assn*›
unfolding *log-unit-clause-def*
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *log-del-binary-clause-impl*
is ‹*uncurry* (*RETURN oo log-del-binary-clause*)›
:: ‹*unat-lit-assn*^{k} $*_a$ *unat-lit-assn*^{k} $\rightarrow_a$ *unit-assn*›
unfolding *log-del-binary-clause-def*
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *mark-literal-for-unit-deletion-impl*
is ‹*RETURN o mark-literal-for-unit-deletion*›
:: ‹*unat-lit-assn*^{k} $\rightarrow_a$ *unit-assn*›
unfolding *mark-literal-for-unit-deletion-def*
by *sepref*

sepref-def *mark-clause-for-unit-as-unchanged-impl*
is ‹*RETURN o mark-clause-for-unit-as-unchanged*›
:: ‹*unat64-assn*^{k} $\rightarrow_a$ *unit-assn*›
unfolding *mark-clause-for-unit-as-unchanged-def*
by *sepref*

sepref-def *mark-clause-for-unit-as-changed-impl*
is ‹*RETURN o mark-clause-for-unit-as-changed*›
:: ‹*unat64-assn*^{k} $\rightarrow_a$ *unit-assn*›
unfolding *mark-clause-for-unit-as-changed-def*
by *sepref*

end
theory *IsaSAT-Backtrack-LLVM*
imports *IsaSAT-Backtrack-Defs IsaSAT-VMTF-State-LLVM IsaSAT-Lookup-Conflict-LLVM*
IsaSAT-Rephase-State-LLVM IsaSAT-LBD-LLVM IsaSAT-Proofs-LLVM
IsaSAT-Stats-LLVM
begin

hide-const (**open**) *NEMonad.ASSERT NEMonad.RETURN*

lemma *isa-empty-conflict-and-extract-clause-heur-alt-def*:
‹*isa-empty-conflict-and-extract-clause-heur M D outl* = *do* {
let C = *replicate* (*length outl*) (*outl*!*0*);
(*D*, *C*, -) ← $WHILE_T$
(λ(*D*, *C*, *i*). *i* < *length-uint32-nat outl*)
(λ(*D*, *C*, *i*). *do* {
ASSERT(*i* < *length outl*);
ASSERT(*i* < *length C*);
ASSERT(*lookup-conflict-remove1-pre* (*outl* ! *i*, *D*));
let D = *lookup-conflict-remove1* (*outl* ! *i*) *D*;
let C = *C*[*i* := *outl* ! *i*];
ASSERT(*get-level-pol-pre* (*M*, *C*!*i*));
ASSERT(*get-level-pol-pre* (*M*, *C*!*1*));
ASSERT(*1* < *length C*);
let L1 = *C*!*i*;
let L2 = *C*!*1*;
let C = (*if get-level-pol M L1* > *get-level-pol M L2 then swap C 1 i else C*);
ASSERT(*i*+*1* ≤ *unat32-max*);
RETURN (*D*, *C*, *i*+*1*)
})
(*D*, *C*, *1*);
ASSERT(*length outl* ≠ *1* ⟶ *length C* > *1*);
ASSERT(*length outl* ≠ *1* ⟶ *get-level-pol-pre* (*M*, *C*!*1*));
RETURN ((*True*, *D*), *C*, *if length outl* = *1 then 0 else get-level-pol M* (*C*!*1*))
}›
unfolding *isa-empty-conflict-and-extract-clause-heur-def*
by *auto*

sepref-def *empty-conflict-and-extract-clause-heur-fast-code*
is ‹*uncurry2* (*isa-empty-conflict-and-extract-clause-heur*)›
:: ‹[λ((*M*, *D*), *outl*). *outl* ≠ [] ∧ *length outl* ≤ *unat32-max*]$_a$
trail-pol-fast-assn^k *$*_a$ *lookup-clause-rel-assn*^{d} *$*_a$ *out-learned-assn*^{k} →
(*conflict-option-rel-assn*) ×$_a$ *clause-ll-assn* ×$_a$ *uint32-nat-assn*›
supply [[*goals-limit*=*1*]] *image-image*[*simp*]
supply [*simp*] = *max-snat-def unat32-max-def*
unfolding *isa-empty-conflict-and-extract-clause-heur-alt-def*
larray-fold-custom-replicate length-uint32-nat-def conflict-option-rel-assn-def
apply (*rewrite at* ‹¤› **in** ‹- !*1*› *snat-const-fold*[**where** *'a*=*64*])+
apply (*rewrite at* ‹¤› **in** ‹- !*0*› *snat-const-fold*[**where** *'a*=*64*])
apply (*rewrite at* ‹*swap* - ¤ -› *snat-const-fold*[**where** *'a*=*64*])
apply (*rewrite at* ‹¤› **in** ‹(-, -, - + *1*)› *snat-const-fold*[**where** *'a*=*64*])
apply (*rewrite at* ‹¤› **in** ‹(-, -, *1*)› *snat-const-fold*[**where** *'a*=*64*])
apply (*rewrite at* ‹¤› **in** ‹*If* (*length* - = ¤)› *snat-const-fold*[**where** *'a*=*64*])
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
unfolding *gen-swap convert-swap*
by *sepref*

lemma *emptied-list-alt-def*: ‹*emptied-list xs = take 0 xs*›
by (*auto simp*: *emptied-list-def*)

sepref-def *empty-cach-code*
is ‹*empty-cach-ref-set*›
:: ‹*cach-refinement-l-assn*^{d} →$_a$ *cach-refinement-l-assn*›
supply [[*goals-limit=1*]]
unfolding *empty-cach-ref-set-def comp-def cach-refinement-l-assn-def emptied-list-alt-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
apply (*rewrite at* ‹-[⌑ := *SEEN-UNKNOWN*]› *value-of-atm-def*[*symmetric*])
apply (*rewrite at* ‹-[⌑ := *SEEN-UNKNOWN*]› *index-of-atm-def*[*symmetric*])
by *sepref*

theorem *empty-cach-code-empty-cach-ref*[*sepref-fr-rules*]:
‹(*empty-cach-code*, *RETURN* ∘ *empty-cach-ref*)
∈ [*empty-cach-ref-pre*]$_a$
cach-refinement-l-assn^d → *cach-refinement-l-assn*›
(**is** ‹*?c* ∈ [*?pre*]$_a$ *?im* → *?f*›)
proof −
have *H*: ‹*?c*
∈[*comp-PRE Id*
(λ(*cach*, *supp*).
(∀ *L*∈*set supp*. *L* < *length cach*) ∧
length supp ≤ *Suc* (*unat32-max div 2*) ∧
(∀ *L*<*length cach*. *cach* ! *L* ≠ *SEEN-UNKNOWN* ⟶ *L* ∈ *set supp*))
(λ*x y*. *True*)
(λ*x*. *nofail* ((*RETURN* ∘ *empty-cach-ref*) *x*))]$_a$
hrp-comp (*cach-refinement-l-assn*^{d})
Id → *hr-comp cach-refinement-l-assn Id*›
(**is** ‹- ∈ [*?pre'*]$_a$ *?im'* → *?f'*›)
using *hfref-compI-PRE*[*OF empty-cach-code.refine*[*unfolded PR-CONST-def*]
empty-cach-ref-set-empty-cach-ref] **by** *simp*
have *pre*: ‹*?pre' h x*› **if** ‹*?pre x*› **for** *x h*
using *that* **by** (*auto simp*: *comp-PRE-def trail-pol-def*
ann-lits-split-reasons-def empty-cach-ref-pre-def)
have *im*: ‹*?im'* = *?im*›
by *simp*
have *f*: ‹*?f'* = *?f*›
by *auto*
show *?thesis*
apply (*rule hfref-weaken-pre*[*OF*])
defer
using *H* **unfolding** *im f* **apply** *assumption*
using *pre* **..**
qed

sepref-register *fm-add-new-fast*

lemma *isasat-fast-length-leD*: ‹*isasat-fast S* ⟹ *Suc* (*length* (*get-clauses-wl-heur S*)) < *max-snat 64*›
by (*cases S*) (*auto simp*: *isasat-fast-def max-snat-def snat64-max-def*)

sepref-register *update-propagation-heuristics*

sepref-def *update-heuristics-stats-impl*
is ‹*uncurry* (*RETURN oo update-propagation-heuristics-stats*)›
:: ‹*uint32-nat-assn*^{k} $*_a$ *heuristic-int-assn*^{d} $\to_a$ *heuristic-int-assn*›
unfolding *update-propagation-heuristics-stats-def heuristic-int-assn-def*
by *sepref*

sepref-def *update-heuristics-impl*
is ‹*uncurry* (*RETURN oo update-propagation-heuristics*)›
:: ‹*uint32-nat-assn*^{k} $*_a$ *heuristic-assn*^{d} $\to_a$ *heuristic-assn*›
unfolding *update-propagation-heuristics-def*
by *sepref*

lemma *isasat-fast-countD-tmp*:
‹*isasat-fast S* $\Longrightarrow$ *clss-size-lcountUEk* (*get-learned-count S*) < *unat64-max*›
by (*auto simp*: *isasat-fast-def learned-clss-count-def*)

lemma *propagate-unit-bt-wl-D-int-alt-def*:
‹*propagate-unit-bt-wl-D-int* = (λ*L S*$_0$. *do* {
let (*M*, *S*) = *extract-trail-wl-heur S*$_0$;
let (*N*, *S*) = *extract-arena-wl-heur S*;
ASSERT (*N* = *get-clauses-wl-heur S*$_0$);
let (*lcount*, *S*) = *extract-lcount-wl-heur S*;
ASSERT (*lcount* = *get-learned-count S*$_0$);
let (*heur*, *S*) = *extract-heur-wl-heur S*;
let (*stats*, *S*) = *extract-stats-wl-heur S*;
let (*lbd*, *S*) = *extract-lbd-wl-heur S*;
let (*vm0*, *S*) = *extract-vmtf-wl-heur S*;
vm ← *isa-bump-heur-flush M vm0*;
glue ← *get-LBD lbd*;
lbd ← *lbd-empty lbd*;
j ← *mop-isa-length-trail M*;
ASSERT(*0* ≠ *DECISION-REASON*);
ASSERT(*cons-trail-Propagated-tr-pre* ((− *L*, *0*::*nat*), *M*));
M ← *cons-trail-Propagated-tr* (− *L*) *0 M*;
let stats = *incr-units-since-last-GC* (*incr-uset stats*);
let S = *update-stats-wl-heur stats S*;
let S = *update-trail-wl-heur M S*;
let S = *update-lbd-wl-heur lbd S*;
let S = *update-literals-to-update-wl-heur j S*;
let S = *update-heur-wl-heur* (*unset-fully-propagated-heur* (*heuristic-reluctant-tick* (*update-propagation-heuristics glue heur*))) *S*;
let S = *update-lcount-wl-heur* (*clss-size-incr-lcountUEk lcount*) *S*;
let S = *update-arena-wl-heur N S*;
let S = *update-vmtf-wl-heur vm S*;
let - = *log-unit-clause* (−*L*);
RETURN S})›
by (*auto simp*: *propagate-unit-bt-wl-D-int-def state-extractors log-unit-clause-def intro*!: *ext split*: *isasat-int-splits*)

sepref-register *cons-trail-Propagated-tr update-heur-wl-heur*
sepref-def *propagate-unit-bt-wl-D-fast-code*
is ‹*uncurry propagate-unit-bt-wl-D-int*›
:: ‹[λ(*L*, *S*). *isasat-fast S*]$_a$ *unat-lit-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\to$ *isasat-bounded-assn*›
supply [[*goals-limit* = *1*]]
isasat-fast-countD[*dest*] *isasat-fast-countD-tmp*[*dest*]
unfolding *propagate-unit-bt-wl-D-int-alt-def*

PR-CONST-def
apply (*annot-snat-const ‹TYPE(64)›*)
by *sepref*

definition *propagate-bt-wl-D-heur-extract* **where**
‹*propagate-bt-wl-D-heur-extract* = (λS_0. *do* {
let (*M*,*S*) = *extract-trail-wl-heur* S_0;
let (*vdom*, *S*) = *extract-vdom-wl-heur S*;
let (*N0*, *S*) = *extract-arena-wl-heur S*;
let (*W0*, *S*) = *extract-watchlist-wl-heur S*;
let (*lcount*, *S*) = *extract-lcount-wl-heur S*;
let (*heur*, *S*) = *extract-heur-wl-heur S*;
let (*stats*, *S*) = *extract-stats-wl-heur S*;
let (*lbd*, *S*) = *extract-lbd-wl-heur S*;
let (*vm0*, *S*) = *extract-vmtf-wl-heur S*;
RETURN (*M*, *vdom*, *N0*, *W0*, *lcount*, *heur*, *stats*, *lbd*, *vm0*, *S*)})›

sepref-def *propagate-bt-wl-D-heur-extract-impl*
is ‹*propagate-bt-wl-D-heur-extract*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *trail-pol-fast-assn* ×$_a$ *aivdom-assn* ×$_a$ *arena-fast-assn* ×$_a$
watchlist-fast-assn ×$_a$ *lcount-assn* ×$_a$ *heuristic-assn* ×$_a$ *isasat-stats-assn* ×$_a$ *lbd-assn* ×$_a$
heuristic-bump-assn ×$_a$ *isasat-bounded-assn*›
unfolding *propagate-bt-wl-D-heur-extract-def*
by *sepref*

definition *propagate-bt-wl-D-heur-update* **where**
‹*propagate-bt-wl-D-heur-update* = (λS_0 *M vdom N0 W0 lcount heur stats lbd vm0 j*. *do* {
let (*S*) = *update-trail-wl-heur M* S_0;
let (*S*) = *update-vdom-wl-heur vdom S*;
let (*S*) = *update-arena-wl-heur N0 S*;
let (*S*) = *update-watchlist-wl-heur W0 S*;
let (*S*) = *update-lcount-wl-heur lcount S*;
let (*S*) = *update-heur-wl-heur heur S*;
let (*S*) = *update-stats-wl-heur stats S*;
let S = *update-lbd-wl-heur lbd S*;
let S = *update-vmtf-wl-heur vm0 S*;
let S = *update-clvls-wl-heur 0 S*;
let S = *update-literals-to-update-wl-heur j S*;
RETURN (*S*)})›

sepref-def *propagate-bt-wl-D-heur-update-impl*
is ‹*uncurry10 propagate-bt-wl-D-heur-update*›
:: ‹*isasat-bounded-assn*^{d} ×$_a$ *trail-pol-fast-assn*^{d} ×$_a$ *aivdom-assn*^{d} ×$_a$ *arena-fast-assn*^{d} ×$_a$
watchlist-fast-assn^d ×$_a$ *lcount-assn*^{d} ×$_a$ *heuristic-assn*^{d} ×$_a$ *isasat-stats-assn*^{d} ×$_a$ *lbd-assn*^{d} ×$_a$
heuristic-bump-assn^d ×$_a$ *sint64-nat-assn*^{k} →$_a$ *isasat-bounded-assn*›
supply [[*goals-limit* = *1*]]
unfolding *propagate-bt-wl-D-heur-update-def*
apply (*rewrite at ‹update-clvls-wl-heur ⌑ -›* *unat-const-fold*[**where** *'a=32*])
by *sepref*

lemma *propagate-bt-wl-D-heur-alt-def*:
‹*propagate-bt-wl-D-heur* = (λ*L C* S_0. *do* {
(*M*, *vdom*, *N0*, *W0*, *lcount*, *heur*, *stats*, *lbd*, *vm0*, *S*) ← *propagate-bt-wl-D-heur-extract* S_0;
ASSERT (*N0* = *get-clauses-wl-heur* S_0);
ASSERT (*vdom* = *get-aivdom* S_0);

```
    ASSERT(length (get-vdom-aivdom vdom) ≤ length N0);
    ASSERT(length (get-avdom-aivdom vdom) ≤ length N0);
    ASSERT(nat-of-lit (C!1) < length W0 ∧ nat-of-lit (−L) < length W0);
    ASSERT(length C > 1);
    let L' = C!1;
    ASSERT(length C ≤ unat32-max div 2 + 1);
    vm ← isa-bump-rescore C M vm0;
    glue ← get-LBD lbd;
    let b = False;
    let l = 2;
    let b' = (length C = l);
    ASSERT(isasat-fast S₀ ⟶ append-and-length-fast-code-pre ((b, C), N0));
    ASSERT(isasat-fast S₀ ⟶ clss-size-lcount lcount < snat64-max);
    (N, i) ← fm-add-new b C N0;
    ASSERT(update-lbd-pre ((i, glue), N));
    let N = update-lbd-and-mark-used i glue N;
    ASSERT(isasat-fast S₀ ⟶ length-ll W0 (nat-of-lit (−L)) < snat64-max);
    let W = W0[nat-of-lit (− L) := W0 ! nat-of-lit (− L) @ [(i, L', b')]];
    ASSERT(isasat-fast S₀ ⟶ length-ll W (nat-of-lit L') < snat64-max);
    let W = W[nat-of-lit L' := W!nat-of-lit L' @ [(i, −L, b')]];
    lbd ← lbd-empty lbd;
    j ← mop-isa-length-trail M;
    ASSERT(i ≠ DECISION-REASON);
    ASSERT(cons-trail-Propagated-tr-pre ((−L, i), M));
    M ← cons-trail-Propagated-tr (− L) i M;
    vm ← isa-bump-heur-flush M vm;
    heur ← mop-save-phase-heur (atm-of L') (is-neg L') heur;
    S ← propagate-bt-wl-D-heur-update S M (add-learned-clause-aivdom i vdom) N
     W (clss-size-incr-lcount lcount) (unset-fully-propagated-heur (heuristic-reluctant-tick (update-propagation-heuristics
glue heur))) (add-lbd (of-nat glue) stats) lbd vm j;
     - ← log-new-clause-heur S i;
    S ← maybe-mark-added-clause-heur2 S i;
    RETURN (S)
  })›
  unfolding propagate-bt-wl-D-heur-def Let-def propagate-bt-wl-D-heur-update-def
      propagate-bt-wl-D-heur-extract-def nres-monad3
  by (auto simp: propagate-bt-wl-D-heur-def Let-def state-extractors propagate-bt-wl-D-heur-update-def
      propagate-bt-wl-D-heur-extract-def intro!: ext bind-cong[OF refl]
      split: isasat-int-splits)

lemmas [sepref-bounds-simps] =
  max-snat-def[of 64, simplified]
  max-unat-def[of 64, simplified]

definition two-sint64 :: nat where [simp]: ‹two-sint64 = 2›
lemma [sepref-fr-rules]:
  ‹(uncurry0 (Mreturn 2), uncurry0 (RETURN two-sint64)) ∈ unit-assn^k →_a sint64-nat-assn›
  apply sepref-to-hoare
  apply (vcg, auto simp: snat-rel-def snat.rel-def br-def snat-invar-def ENTAILS-def
    snat-numeral max-snat-def exists-eq-star-conv Exists-eq-simp
    sep-conj-commute pure-true-conv)
  done
```

16.2 Backtrack with direct extraction of literal if highest level

lemma *le-unat32-max-div-2-le-unat32-max*: ‹$a \leq$ *unat32-max div 2 + 1* $\Longrightarrow a \leq$ *unat32-max*›
by (*auto simp*: *unat32-max-def snat64-max-def*)

lemma *propagate-bt-wl-D-fast-code-isasat-fastI2*: ‹*isasat-fast b* $\Longrightarrow$
a < *length* (*get-clauses-wl-heur b*) $\Longrightarrow a \leq$ *snat64-max*›
by (*cases b*) (*auto simp*: *isasat-fast-def*)

lemma *propagate-bt-wl-D-fast-code-isasat-fastI3*: ‹*isasat-fast b* $\Longrightarrow$
a $\leq$ *length* (*get-clauses-wl-heur b*) $\Longrightarrow a <$ *snat64-max*›
by (*cases b*) (*auto simp*: *isasat-fast-def snat64-max-def unat32-max-def*)

sepref-register *propagate-bt-wl-D-heur-update propagate-bt-wl-D-heur-extract two-sint64*
sepref-def *propagate-bt-wl-D-fast-codeXX*
is ‹*uncurry2 propagate-bt-wl-D-heur*›
:: ‹$[\lambda((L,\ C),\ S).$ *isasat-fast S*$]_a$
unat-lit-assn^k $*_a$ *clause-ll-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\to$ *isasat-bounded-assn*›
supply [[*goals-limit* = *1*]] *append-ll-def*[*simp*] *isasat-fast-length-leD*[*dest*]
propagate-bt-wl-D-fast-code-isasat-fastI2[*intro*] *length-ll-def*[*simp*]
propagate-bt-wl-D-fast-code-isasat-fastI3[*intro*]
isasat-fast-countD[*dest*]
unfolding *propagate-bt-wl-D-heur-alt-def*
fm-add-new-fast-def[*symmetric*]
unfolding *delete-index-and-swap-update-def*[*symmetric*] *append-update-def*[*symmetric*]
append-ll-def[*symmetric*] *append-ll-def*[*symmetric*] *two-sint64-def*[*symmetric*]
PR-CONST-def save-phase-def fold-tuple-optimizations
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *extract-shorter-conflict-list-heur-st-alt-def*:
‹*extract-shorter-conflict-list-heur-st* = (λS_0. *do* {
let (*M*,*S*) = *extract-trail-wl-heur* S_0;
let (*N*, *S*) = *extract-arena-wl-heur S*;
ASSERT (*N* = *get-clauses-wl-heur* S_0);
let (*lbd*, *S*) = *extract-lbd-wl-heur S*;
let (*vm0*, *S*) = *extract-vmtf-wl-heur S*;
let (*outl*, *S*) = *extract-outl-wl-heur S*;
let (*bD*, *S*) = *extract-conflict-wl-heur S*;
let (*ccach*, *S*) = *extract-ccach-wl-heur S*;
lbd $\leftarrow$ *mark-lbd-from-list-heur M outl lbd*;
let D = *the-lookup-conflict bD*;
ASSERT(*fst M* $\neq$ []);
let K = *lit-of-last-trail-pol M*;
ASSERT(*0* < *length outl*);
ASSERT(*lookup-conflict-remove1-pre* (−*K*, *D*));
let D = *lookup-conflict-remove1* (−*K*) *D*;
let outl = *outl*[*0* := −*K*];
vm $\leftarrow$ *isa-vmtf-mark-to-rescore-also-reasons M N outl* (−*K*) *vm0*;
(*D*, *ccach*, *outl*) $\leftarrow$ *isa-minimize-and-extract-highest-lookup-conflict M N D ccach lbd outl*;
ASSERT(*empty-cach-ref-pre ccach*);
let ccach = *empty-cach-ref ccach*;
ASSERT(*outl* $\neq$ [] $\wedge$ *length outl* $\leq$ *unat32-max*);
(*D*, *C*, *n*) $\leftarrow$ *isa-empty-conflict-and-extract-clause-heur M D outl*;
let S = *update-trail-wl-heur M S*;
let S = *update-arena-wl-heur N S*;

```
    let S = update-vmtf-wl-heur vm S;
    let S = update-lbd-wl-heur lbd S;
    let S = update-outl-wl-heur (take 1 outl) S;
    let S = update-ccach-wl-heur ccach S;
    let S = update-conflict-wl-heur D S;
   RETURN (S, n, C)
 })⟩
 unfolding extract-shorter-conflict-list-heur-st-def
 by (auto simp: the-lookup-conflict-def Let-def state-extractors intro!: ext bind-cong[OF refl]
   split: isasat-int-splits)

sepref-register isa-minimize-and-extract-highest-lookup-conflict
   isa-vmtf-mark-to-rescore-also-reasons

sepref-def extract-shorter-conflict-list-heur-st-fast
 is ‹extract-shorter-conflict-list-heur-st›
 :: ‹[λS. length (get-clauses-wl-heur S) ≤ snat64-max]_a
      isasat-bounded-assn^d → isasat-bounded-assn ×_a uint32-nat-assn ×_a clause-ll-assn›
 supply [[goals-limit=1]]
 unfolding extract-shorter-conflict-list-heur-st-alt-def PR-CONST-def
 unfolding delete-index-and-swap-update-def[symmetric] append-update-def[symmetric]
 apply (annot-snat-const ‹TYPE(64)›)
 by sepref

sepref-register find-lit-of-max-level-wl
 extract-shorter-conflict-list-heur-st lit-of-hd-trail-st-heur propagate-bt-wl-D-heur
 propagate-unit-bt-wl-D-int
sepref-register backtrack-wl

lemma get-learned-count-learned-clss-countD2:
 ‹get-learned-count S = (get-learned-count T) ⟹
     learned-clss-count S ≤ learned-clss-count T›
 by (cases S; cases T) (auto simp: learned-clss-count-def)

lemma backtrack-wl-D-nlit-heurI:
 ‹isasat-fast x ⟹
     get-clauses-wl-heur xc = get-clauses-wl-heur x ⟹
     get-learned-count xc = get-learned-count x ⟹ isasat-fast xc›
 by (auto simp: isasat-fast-def dest: get-learned-count-learned-clss-countD2)

sepref-register save-phase-st
sepref-def backtrack-wl-D-fast-code
 is ‹backtrack-wl-D-nlit-heur›
 :: ‹[isasat-fast]_a isasat-bounded-assn^d → isasat-bounded-assn›
 supply [[goals-limit=1]]
   size-conflict-wl-def[simp] isasat-fast-length-leD[intro] backtrack-wl-D-nlit-heurI[intro]
   isasat-fast-countD[dest] IsaSAT-Setup.isasat-fast-length-leD[dest]
 unfolding backtrack-wl-D-nlit-heur-def PR-CONST-def
 unfolding delete-index-and-swap-update-def[symmetric] append-update-def[symmetric]
   append-ll-def[symmetric]
   size-conflict-wl-def[symmetric]
 apply (annot-snat-const ‹TYPE(64)›)
 by sepref


lemmas [llvm-inline] = add-lbd-def
```

experiment
begin
export-llvm
empty-conflict-and-extract-clause-heur-fast-code
empty-cach-code
update-heuristics-impl
update-heuristics-impl
isa-vmtf-flush-fast-code
get-LBD-code
mop-isa-length-trail-fast-code
cons-trail-Propagated-tr-fast-code
update-heuristics-impl
append-and-length-fast-code
update-lbd-impl
reluctant-tick-impl
propagate-bt-wl-D-fast-codeXX
end

end
theory *Tuple15*
imports
More-Sepref.WB-More-Refinement IsaSAT-Literals
begin

This is the setup for accessing and modifying the state as an abstract tuple of 15 elements. The construction is kept generic (even if still targetting only our state). There is a lot of copy-paste that would be nice to automate at some point.

We define 3 sort of operations:

1. extracting an element, replacing it by an default element. Modifies the state. The name starts with *exctr*
2. reinserting an element, freeing the current one. Modifies the state. The name starts with *update*
3. in-place reading a value, possibly with pure parameters. Does not modify the state. The name starts with *read*

datatype ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*) *tuple15* = *Tuple15*
(*Tuple15-a*: '*a*)
(*Tuple15-b*: '*b*)
(*Tuple15-c*: '*c*)
(*Tuple15-d*: '*d*)
(*Tuple15-e*: '*e*)
(*Tuple15-f*: '*f*)
(*Tuple15-g*: '*g*)
(*Tuple15-h*: '*h*)
(*Tuple15-i*: '*i*)
(*Tuple15-j*: '*j*)
(*Tuple15-k*: '*k*)
(*Tuple15-l*: '*l*)
(*Tuple15-m*: '*m*)
(*Tuple15-n*: '*n*)

(*Tuple15-o*: *'o*)

context
begin

qualified fun *set-a* :: ‹*'a* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*) *tuple15* ⇒ -› **where**
‹*set-a M* (*Tuple15* - *N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*) = (*Tuple15 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*)›

qualified fun *set-b* :: ‹*'b* ⇒ - ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*) *tuple15*› **where**
‹*set-b N* (*Tuple15 M* - *D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*) = (*Tuple15 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*)›

qualified fun *set-c* :: ‹*'c* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*) *tuple15* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*) *tuple15*› **where**
‹*set-c D* (*Tuple15 M N* - *i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*) = (*Tuple15 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*)›

qualified fun *set-d* :: ‹*'d* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*) *tuple15* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*) *tuple15*› **where**
‹*set-d i* (*Tuple15 M N D* - *W ivmtf icount ccach lbd outl heur stats aivdom clss opts*) = (*Tuple15 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*)›

qualified fun *set-e* :: ‹*'e* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*) *tuple15* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*) *tuple15*› **where**
‹*set-e W* (*Tuple15 M N D i* - *ivmtf icount ccach lbd outl heur stats aivdom clss opts*) = (*Tuple15 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*)›

qualified fun *set-f* :: ‹*'f* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*) *tuple15* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*) *tuple15*› **where**
‹*set-f ivmtf* (*Tuple15 M N D i W* - *icount ccach lbd outl heur stats aivdom clss opts*) = (*Tuple15 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*)›

qualified fun *set-g* :: ‹*'g* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*) *tuple15* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*) *tuple15*› **where**
‹*set-g icount* (*Tuple15 M N D i W ivmtf* - *ccach lbd outl heur stats aivdom clss opts*) = (*Tuple15 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*)›

qualified fun *set-h* :: ‹*'h* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*) *tuple15* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*) *tuple15*› **where**
‹*set-h ccach* (*Tuple15 M N D i W ivmtf icount* - *lbd outl heur stats aivdom clss opts*) = (*Tuple15 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*)›

qualified fun *set-i* :: ‹*'i* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*) *tuple15* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*) *tuple15*› **where**
‹*set-i lbd* (*Tuple15 M N D i W ivmtf icount ccach* - *outl heur stats aivdom clss opts*) = (*Tuple15 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*)›

qualified fun *set-j* :: ‹*'j* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*) *tuple15* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*) *tuple15*› **where**
‹*set-j outl* (*Tuple15 M N D i W ivmtf icount ccach lbd* - *heur stats aivdom clss opts*) = (*Tuple15 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*)›

qualified fun *set-l* :: ‹*'l* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*) *tuple15* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*, *'k*, *'l*, *'m*, *'n*, *'o*) *tuple15*› **where**
‹*set-l heur* (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats* - *aivdom clss opts*) = (*Tuple15 M N*

D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts)›

qualified fun *set-k* :: ‹'*k* ⇒ ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*) *tuple15* ⇒ ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*) *tuple15*› **where**
‹*set-k stats* (*Tuple15 M N D i W ivmtf icount ccach lbd outl - heur aivdom clss opts*) = (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*)›

qualified fun *set-m* :: ‹'*m* ⇒ ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*) *tuple15* ⇒ ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*) *tuple15*› **where**
‹*set-m aivdom* (*Tuple15 M N D i W ivmtf icount ccach lbd outl heur stats - clss opts*) = (*Tuple15 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*)›

qualified fun *set-n* :: ‹'*n* ⇒ ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*) *tuple15* ⇒ ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*) *tuple15*› **where**
‹*set-n clss* (*Tuple15 M N D i W ivmtf icount ccach lbd outl heur stats aivdom - opts*) = (*Tuple15 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*)›

qualified fun *set-o* :: ‹'*o* ⇒ ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*) *tuple15* ⇒ ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*) *tuple15*› **where**
‹*set-o opts* (*Tuple15 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss -*) = (*Tuple15 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*)›

end

lemma *lambda-comp-true*: ‹(λ*S*. *True*) ∘ *f* = (λ-. *True*)› ‹*uncurry* (λ*a b*. *True*) = (λ-. *True*)› ‹*uncurry2* (λ*a b c*. *True*) = (λ-. *True*)›
‹*case-tuple15* (λ*M* - - - - - - - - - - - - - -. *True*) = (λ-. *True*)›
by (*auto intro*!: *ext split*: *Tuple15.tuple15.splits*)

named-theorems *Tuple15-state-simp* ‹*Simplify the state setter and extractors*›
lemma [*Tuple15-state-simp*]:
‹*Tuple15-a* (*Tuple15.set-a a S*) = *a*›
‹*Tuple15-b* (*Tuple15.set-a b S*) = *Tuple15-b S*›
‹*Tuple15-c* (*Tuple15.set-a b S*) = *Tuple15-c S*›
‹*Tuple15-d* (*Tuple15.set-a b S*) = *Tuple15-d S*›
‹*Tuple15-e* (*Tuple15.set-a b S*) = *Tuple15-e S*›
‹*Tuple15-f* (*Tuple15.set-a b S*) = *Tuple15-f S*›
‹*Tuple15-g* (*Tuple15.set-a b S*) = *Tuple15-g S*›
‹*Tuple15-h* (*Tuple15.set-a b S*) = *Tuple15-h S*›
‹*Tuple15-i* (*Tuple15.set-a b S*) = *Tuple15-i S*›
‹*Tuple15-j* (*Tuple15.set-a b S*) = *Tuple15-j S*›
‹*Tuple15-k* (*Tuple15.set-a b S*) = *Tuple15-k S*›
‹*Tuple15-l* (*Tuple15.set-a b S*) = *Tuple15-l S*›
‹*Tuple15-m* (*Tuple15.set-a b S*) = *Tuple15-m S*›
‹*Tuple15-n* (*Tuple15.set-a b S*) = *Tuple15-n S*›
‹*Tuple15-o* (*Tuple15.set-a b S*) = *Tuple15-o S*›
by (*cases S*; *auto*; *fail*)+

lemma [*Tuple15-state-simp*]:
‹*Tuple15-a* (*Tuple15.set-b b S*) = *Tuple15-a S*›
‹*Tuple15-b* (*Tuple15.set-b b S*) = *b*›
‹*Tuple15-c* (*Tuple15.set-b b S*) = *Tuple15-c S*›
‹*Tuple15-d* (*Tuple15.set-b b S*) = *Tuple15-d S*›
‹*Tuple15-e* (*Tuple15.set-b b S*) = *Tuple15-e S*›
‹*Tuple15-f* (*Tuple15.set-b b S*) = *Tuple15-f S*›
‹*Tuple15-g* (*Tuple15.set-b b S*) = *Tuple15-g S*›

‹*Tuple15-h* (*Tuple15.set-b b S*) = *Tuple15-h S*›
‹*Tuple15-i* (*Tuple15.set-b b S*) = *Tuple15-i S*›
‹*Tuple15-j* (*Tuple15.set-b b S*) = *Tuple15-j S*›
‹*Tuple15-k* (*Tuple15.set-b b S*) = *Tuple15-k S*›
‹*Tuple15-l* (*Tuple15.set-b b S*) = *Tuple15-l S*›
‹*Tuple15-m* (*Tuple15.set-b b S*) = *Tuple15-m S*›
‹*Tuple15-n* (*Tuple15.set-b b S*) = *Tuple15-n S*›
‹*Tuple15-o* (*Tuple15.set-b b S*) = *Tuple15-o S*›
by (*cases S*; *auto*; *fail*)+

lemma [*Tuple15-state-simp*]:
‹*Tuple15-a* (*Tuple15.set-c b S*) = *Tuple15-a S*›
‹*Tuple15-b* (*Tuple15.set-c b S*) = *Tuple15-b S*›
‹*Tuple15-c* (*Tuple15.set-c b S*) = *b*›
‹*Tuple15-d* (*Tuple15.set-c b S*) = *Tuple15-d S*›
‹*Tuple15-e* (*Tuple15.set-c b S*) = *Tuple15-e S*›
‹*Tuple15-f* (*Tuple15.set-c b S*) = *Tuple15-f S*›
‹*Tuple15-g* (*Tuple15.set-c b S*) = *Tuple15-g S*›
‹*Tuple15-h* (*Tuple15.set-c b S*) = *Tuple15-h S*›
‹*Tuple15-i* (*Tuple15.set-c b S*) = *Tuple15-i S*›
‹*Tuple15-j* (*Tuple15.set-c b S*) = *Tuple15-j S*›
‹*Tuple15-k* (*Tuple15.set-c b S*) = *Tuple15-k S*›
‹*Tuple15-l* (*Tuple15.set-c b S*) = *Tuple15-l S*›
‹*Tuple15-m* (*Tuple15.set-c b S*) = *Tuple15-m S*›
‹*Tuple15-n* (*Tuple15.set-c b S*) = *Tuple15-n S*›
‹*Tuple15-o* (*Tuple15.set-c b S*) = *Tuple15-o S*›
by (*cases S*; *auto*; *fail*)+

lemma [*Tuple15-state-simp*]:
‹*Tuple15-a* (*Tuple15.set-d b S*) = *Tuple15-a S*›
‹*Tuple15-b* (*Tuple15.set-d b S*) = *Tuple15-b S*›
‹*Tuple15-c* (*Tuple15.set-d b S*) = *Tuple15-c S*›
‹*Tuple15-d* (*Tuple15.set-d b S*) = *b*›
‹*Tuple15-e* (*Tuple15.set-d b S*) = *Tuple15-e S*›
‹*Tuple15-f* (*Tuple15.set-d b S*) = *Tuple15-f S*›
‹*Tuple15-g* (*Tuple15.set-d b S*) = *Tuple15-g S*›
‹*Tuple15-h* (*Tuple15.set-d b S*) = *Tuple15-h S*›
‹*Tuple15-i* (*Tuple15.set-d b S*) = *Tuple15-i S*›
‹*Tuple15-j* (*Tuple15.set-d b S*) = *Tuple15-j S*›
‹*Tuple15-k* (*Tuple15.set-d b S*) = *Tuple15-k S*›
‹*Tuple15-l* (*Tuple15.set-d b S*) = *Tuple15-l S*›
‹*Tuple15-m* (*Tuple15.set-d b S*) = *Tuple15-m S*›
‹*Tuple15-n* (*Tuple15.set-d b S*) = *Tuple15-n S*›
‹*Tuple15-o* (*Tuple15.set-d b S*) = *Tuple15-o S*›
by (*cases S*; *auto*; *fail*)+

lemma [*Tuple15-state-simp*]:
‹*Tuple15-a* (*Tuple15.set-e b S*) = *Tuple15-a S*›
‹*Tuple15-b* (*Tuple15.set-e b S*) = *Tuple15-b S*›
‹*Tuple15-c* (*Tuple15.set-e b S*) = *Tuple15-c S*›
‹*Tuple15-d* (*Tuple15.set-e b S*) = *Tuple15-d S*›
‹*Tuple15-e* (*Tuple15.set-e b S*) = *b*›
‹*Tuple15-f* (*Tuple15.set-e b S*) = *Tuple15-f S*›
‹*Tuple15-g* (*Tuple15.set-e b S*) = *Tuple15-g S*›
‹*Tuple15-h* (*Tuple15.set-e b S*) = *Tuple15-h S*›
‹*Tuple15-i* (*Tuple15.set-e b S*) = *Tuple15-i S*›

‹*Tuple15-j* (*Tuple15.set-e b S*) = *Tuple15-j S*›
‹*Tuple15-k* (*Tuple15.set-e b S*) = *Tuple15-k S*›
‹*Tuple15-l* (*Tuple15.set-e b S*) = *Tuple15-l S*›
‹*Tuple15-m* (*Tuple15.set-e b S*) = *Tuple15-m S*›
‹*Tuple15-n* (*Tuple15.set-e b S*) = *Tuple15-n S*›
‹*Tuple15-o* (*Tuple15.set-e b S*) = *Tuple15-o S*›
by (*cases S*; *auto*; *fail*)+

lemma [*Tuple15-state-simp*]:
‹*Tuple15-a* (*Tuple15.set-f b S*) = *Tuple15-a S*›
‹*Tuple15-b* (*Tuple15.set-f b S*) = *Tuple15-b S*›
‹*Tuple15-c* (*Tuple15.set-f b S*) = *Tuple15-c S*›
‹*Tuple15-d* (*Tuple15.set-f b S*) = *Tuple15-d S*›
‹*Tuple15-e* (*Tuple15.set-f b S*) = *Tuple15-e S*›
‹*Tuple15-f* (*Tuple15.set-f b S*) = *b*›
‹*Tuple15-g* (*Tuple15.set-f b S*) = *Tuple15-g S*›
‹*Tuple15-h* (*Tuple15.set-f b S*) = *Tuple15-h S*›
‹*Tuple15-i* (*Tuple15.set-f b S*) = *Tuple15-i S*›
‹*Tuple15-j* (*Tuple15.set-f b S*) = *Tuple15-j S*›
‹*Tuple15-k* (*Tuple15.set-f b S*) = *Tuple15-k S*›
‹*Tuple15-l* (*Tuple15.set-f b S*) = *Tuple15-l S*›
‹*Tuple15-m* (*Tuple15.set-f b S*) = *Tuple15-m S*›
‹*Tuple15-n* (*Tuple15.set-f b S*) = *Tuple15-n S*›
‹*Tuple15-o* (*Tuple15.set-f b S*) = *Tuple15-o S*›
by (*cases S*; *auto*; *fail*)+

lemma [*Tuple15-state-simp*]:
‹*Tuple15-a* (*Tuple15.set-g b S*) = *Tuple15-a S*›
‹*Tuple15-b* (*Tuple15.set-g b S*) = *Tuple15-b S*›
‹*Tuple15-c* (*Tuple15.set-g b S*) = *Tuple15-c S*›
‹*Tuple15-d* (*Tuple15.set-g b S*) = *Tuple15-d S*›
‹*Tuple15-e* (*Tuple15.set-g b S*) = *Tuple15-e S*›
‹*Tuple15-f* (*Tuple15.set-g b S*) = *Tuple15-f S*›
‹*Tuple15-g* (*Tuple15.set-g b S*) = *b*›
‹*Tuple15-h* (*Tuple15.set-g b S*) = *Tuple15-h S*›
‹*Tuple15-i* (*Tuple15.set-g b S*) = *Tuple15-i S*›
‹*Tuple15-j* (*Tuple15.set-g b S*) = *Tuple15-j S*›
‹*Tuple15-k* (*Tuple15.set-g b S*) = *Tuple15-k S*›
‹*Tuple15-l* (*Tuple15.set-g b S*) = *Tuple15-l S*›
‹*Tuple15-m* (*Tuple15.set-g b S*) = *Tuple15-m S*›
‹*Tuple15-n* (*Tuple15.set-g b S*) = *Tuple15-n S*›
‹*Tuple15-o* (*Tuple15.set-g b S*) = *Tuple15-o S*›
by (*cases S*; *auto*; *fail*)+

lemma [*Tuple15-state-simp*]:
‹*Tuple15-a* (*Tuple15.set-h b S*) = *Tuple15-a S*›
‹*Tuple15-b* (*Tuple15.set-h b S*) = *Tuple15-b S*›
‹*Tuple15-c* (*Tuple15.set-h b S*) = *Tuple15-c S*›
‹*Tuple15-d* (*Tuple15.set-h b S*) = *Tuple15-d S*›
‹*Tuple15-e* (*Tuple15.set-h b S*) = *Tuple15-e S*›
‹*Tuple15-f* (*Tuple15.set-h b S*) = *Tuple15-f S*›
‹*Tuple15-g* (*Tuple15.set-h b S*) = *Tuple15-g S*›
‹*Tuple15-h* (*Tuple15.set-h b S*) = *b*›
‹*Tuple15-i* (*Tuple15.set-h b S*) = *Tuple15-i S*›
‹*Tuple15-j* (*Tuple15.set-h b S*) = *Tuple15-j S*›
‹*Tuple15-k* (*Tuple15.set-h b S*) = *Tuple15-k S*›

‹*Tuple15-l* (*Tuple15.set-h b S*) = *Tuple15-l S*›
‹*Tuple15-m* (*Tuple15.set-h b S*) = *Tuple15-m S*›
‹*Tuple15-n* (*Tuple15.set-h b S*) = *Tuple15-n S*›
‹*Tuple15-o* (*Tuple15.set-h b S*) = *Tuple15-o S*›
by (*cases S*; *auto*; *fail*)+

lemma [*Tuple15-state-simp*]:
‹*Tuple15-a* (*Tuple15.set-i b S*) = *Tuple15-a S*›
‹*Tuple15-b* (*Tuple15.set-i b S*) = *Tuple15-b S*›
‹*Tuple15-c* (*Tuple15.set-i b S*) = *Tuple15-c S*›
‹*Tuple15-d* (*Tuple15.set-i b S*) = *Tuple15-d S*›
‹*Tuple15-e* (*Tuple15.set-i b S*) = *Tuple15-e S*›
‹*Tuple15-f* (*Tuple15.set-i b S*) = *Tuple15-f S*›
‹*Tuple15-g* (*Tuple15.set-i b S*) = *Tuple15-g S*›
‹*Tuple15-h* (*Tuple15.set-i b S*) = *Tuple15-h S*›
‹*Tuple15-i* (*Tuple15.set-i b S*) = *b*›
‹*Tuple15-j* (*Tuple15.set-i b S*) = *Tuple15-j S*›
‹*Tuple15-k* (*Tuple15.set-i b S*) = *Tuple15-k S*›
‹*Tuple15-l* (*Tuple15.set-i b S*) = *Tuple15-l S*›
‹*Tuple15-m* (*Tuple15.set-i b S*) = *Tuple15-m S*›
‹*Tuple15-n* (*Tuple15.set-i b S*) = *Tuple15-n S*›
‹*Tuple15-o* (*Tuple15.set-i b S*) = *Tuple15-o S*›
by (*cases S*; *auto*; *fail*)+

lemma [*Tuple15-state-simp*]:
‹*Tuple15-a* (*Tuple15.set-j b S*) = *Tuple15-a S*›
‹*Tuple15-b* (*Tuple15.set-j b S*) = *Tuple15-b S*›
‹*Tuple15-c* (*Tuple15.set-j b S*) = *Tuple15-c S*›
‹*Tuple15-d* (*Tuple15.set-j b S*) = *Tuple15-d S*›
‹*Tuple15-e* (*Tuple15.set-j b S*) = *Tuple15-e S*›
‹*Tuple15-f* (*Tuple15.set-j b S*) = *Tuple15-f S*›
‹*Tuple15-g* (*Tuple15.set-j b S*) = *Tuple15-g S*›
‹*Tuple15-h* (*Tuple15.set-j b S*) = *Tuple15-h S*›
‹*Tuple15-i* (*Tuple15.set-j b S*) = *Tuple15-i S*›
‹*Tuple15-j* (*Tuple15.set-j b S*) = *b*›
‹*Tuple15-k* (*Tuple15.set-j b S*) = *Tuple15-k S*›
‹*Tuple15-l* (*Tuple15.set-j b S*) = *Tuple15-l S*›
‹*Tuple15-m* (*Tuple15.set-j b S*) = *Tuple15-m S*›
‹*Tuple15-n* (*Tuple15.set-j b S*) = *Tuple15-n S*›
‹*Tuple15-o* (*Tuple15.set-j b S*) = *Tuple15-o S*›
by (*cases S*; *auto*; *fail*)+

lemma [*Tuple15-state-simp*]:
‹*Tuple15-a* (*Tuple15.set-k b S*) = *Tuple15-a S*›
‹*Tuple15-b* (*Tuple15.set-k b S*) = *Tuple15-b S*›
‹*Tuple15-c* (*Tuple15.set-k b S*) = *Tuple15-c S*›
‹*Tuple15-d* (*Tuple15.set-k b S*) = *Tuple15-d S*›
‹*Tuple15-e* (*Tuple15.set-k b S*) = *Tuple15-e S*›
‹*Tuple15-f* (*Tuple15.set-k b S*) = *Tuple15-f S*›
‹*Tuple15-g* (*Tuple15.set-k b S*) = *Tuple15-g S*›
‹*Tuple15-h* (*Tuple15.set-k b S*) = *Tuple15-h S*›
‹*Tuple15-i* (*Tuple15.set-k b S*) = *Tuple15-i S*›
‹*Tuple15-j* (*Tuple15.set-k b S*) = *Tuple15-j S*›
‹*Tuple15-k* (*Tuple15.set-k b S*) = *b*›
‹*Tuple15-l* (*Tuple15.set-k b S*) = *Tuple15-l S*›
‹*Tuple15-m* (*Tuple15.set-k b S*) = *Tuple15-m S*›

‹*Tuple15-n* (*Tuple15.set-k b S*) = *Tuple15-n S*›
‹*Tuple15-o* (*Tuple15.set-k b S*) = *Tuple15-o S*›
by (*cases S*; *auto*; *fail*)+

lemma [*Tuple15-state-simp*]:
‹*Tuple15-a* (*Tuple15.set-l b S*) = *Tuple15-a S*›
‹*Tuple15-b* (*Tuple15.set-l b S*) = *Tuple15-b S*›
‹*Tuple15-c* (*Tuple15.set-l b S*) = *Tuple15-c S*›
‹*Tuple15-d* (*Tuple15.set-l b S*) = *Tuple15-d S*›
‹*Tuple15-e* (*Tuple15.set-l b S*) = *Tuple15-e S*›
‹*Tuple15-f* (*Tuple15.set-l b S*) = *Tuple15-f S*›
‹*Tuple15-g* (*Tuple15.set-l b S*) = *Tuple15-g S*›
‹*Tuple15-h* (*Tuple15.set-l b S*) = *Tuple15-h S*›
‹*Tuple15-i* (*Tuple15.set-l b S*) = *Tuple15-i S*›
‹*Tuple15-j* (*Tuple15.set-l b S*) = *Tuple15-j S*›
‹*Tuple15-k* (*Tuple15.set-l b S*) = *Tuple15-k S*›
‹*Tuple15-l* (*Tuple15.set-l b S*) = *b*›
‹*Tuple15-m* (*Tuple15.set-l b S*) = *Tuple15-m S*›
‹*Tuple15-n* (*Tuple15.set-l b S*) = *Tuple15-n S*›
‹*Tuple15-o* (*Tuple15.set-l b S*) = *Tuple15-o S*›
by (*cases S*; *auto*; *fail*)+

lemma [*Tuple15-state-simp*]:
‹*Tuple15-a* (*Tuple15.set-m b S*) = *Tuple15-a S*›
‹*Tuple15-b* (*Tuple15.set-m b S*) = *Tuple15-b S*›
‹*Tuple15-c* (*Tuple15.set-m b S*) = *Tuple15-c S*›
‹*Tuple15-d* (*Tuple15.set-m b S*) = *Tuple15-d S*›
‹*Tuple15-e* (*Tuple15.set-m b S*) = *Tuple15-e S*›
‹*Tuple15-f* (*Tuple15.set-m b S*) = *Tuple15-f S*›
‹*Tuple15-g* (*Tuple15.set-m b S*) = *Tuple15-g S*›
‹*Tuple15-h* (*Tuple15.set-m b S*) = *Tuple15-h S*›
‹*Tuple15-i* (*Tuple15.set-m b S*) = *Tuple15-i S*›
‹*Tuple15-j* (*Tuple15.set-m b S*) = *Tuple15-j S*›
‹*Tuple15-k* (*Tuple15.set-m b S*) = *Tuple15-k S*›
‹*Tuple15-l* (*Tuple15.set-m b S*) = *Tuple15-l S*›
‹*Tuple15-m* (*Tuple15.set-m b S*) = *b*›
‹*Tuple15-n* (*Tuple15.set-m b S*) = *Tuple15-n S*›
‹*Tuple15-o* (*Tuple15.set-m b S*) = *Tuple15-o S*›
by (*cases S*; *auto*; *fail*)+

lemma [*Tuple15-state-simp*]:
‹*Tuple15-a* (*Tuple15.set-n b S*) = *Tuple15-a S*›
‹*Tuple15-b* (*Tuple15.set-n b S*) = *Tuple15-b S*›
‹*Tuple15-c* (*Tuple15.set-n b S*) = *Tuple15-c S*›
‹*Tuple15-d* (*Tuple15.set-n b S*) = *Tuple15-d S*›
‹*Tuple15-e* (*Tuple15.set-n b S*) = *Tuple15-e S*›
‹*Tuple15-f* (*Tuple15.set-n b S*) = *Tuple15-f S*›
‹*Tuple15-g* (*Tuple15.set-n b S*) = *Tuple15-g S*›
‹*Tuple15-h* (*Tuple15.set-n b S*) = *Tuple15-h S*›
‹*Tuple15-i* (*Tuple15.set-n b S*) = *Tuple15-i S*›
‹*Tuple15-j* (*Tuple15.set-n b S*) = *Tuple15-j S*›
‹*Tuple15-k* (*Tuple15.set-n b S*) = *Tuple15-k S*›
‹*Tuple15-l* (*Tuple15.set-n b S*) = *Tuple15-l S*›
‹*Tuple15-m* (*Tuple15.set-n b S*) = *Tuple15-m S*›
‹*Tuple15-n* (*Tuple15.set-n b S*) = *b*›

‹*Tuple15-o* (*Tuple15.set-n b S*) = *Tuple15-o S*›
by (*cases S*; *auto*; *fail*)+

lemma [*Tuple15-state-simp*]:
‹*Tuple15-a* (*Tuple15.set-o b S*) = *Tuple15-a S*›
‹*Tuple15-b* (*Tuple15.set-o b S*) = *Tuple15-b S*›
‹*Tuple15-c* (*Tuple15.set-o b S*) = *Tuple15-c S*›
‹*Tuple15-d* (*Tuple15.set-o b S*) = *Tuple15-d S*›
‹*Tuple15-e* (*Tuple15.set-o b S*) = *Tuple15-e S*›
‹*Tuple15-f* (*Tuple15.set-o b S*) = *Tuple15-f S*›
‹*Tuple15-g* (*Tuple15.set-o b S*) = *Tuple15-g S*›
‹*Tuple15-h* (*Tuple15.set-o b S*) = *Tuple15-h S*›
‹*Tuple15-i* (*Tuple15.set-o b S*) = *Tuple15-i S*›
‹*Tuple15-j* (*Tuple15.set-o b S*) = *Tuple15-j S*›
‹*Tuple15-k* (*Tuple15.set-o b S*) = *Tuple15-k S*›
‹*Tuple15-l* (*Tuple15.set-o b S*) = *Tuple15-l S*›
‹*Tuple15-m* (*Tuple15.set-o b S*) = *Tuple15-m S*›
‹*Tuple15-n* (*Tuple15.set-o b S*) = *Tuple15-n S*›
‹*Tuple15-o* (*Tuple15.set-o b S*) = *b*›
by (*cases S*; *auto*; *fail*)+

declare *Tuple15-state-simp*[*simp*]
end
theory *IsaSAT-Bump-Heuristics-Init-State*
imports *Watched-Literals-VMTF IsaSAT-ACIDS*
Tuple4 IsaSAT-ACIDS Pairing-Heap-LLVM.Relational-Pairing-Heaps Pairing-Heap-LLVM.Pairing-Heaps-Impl
begin

type-synonym *vmtf-remove-int-option-fst-As* = ‹*nat-vmtf-node list* × *nat* × *nat option* × *nat option* × *nat option*›

definition *isa-vmtf-init*
:: ‹*nat multiset* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *vmtf-remove-int-option-fst-As set*›
where
‹*isa-vmtf-init* $\mathcal{A}_{in}$ *M* = {(*ns*, *m*, *fst-As*, *lst-As*, *next-search*).
$\mathcal{A}_{in} \neq \{\#\} \longrightarrow$ (*fst-As* ≠ *None* ∧ *lst-As* ≠ *None* ∧ (*ns*, *m*, *the fst-As*, *the lst-As*, *next-search*) ∈ *vmtf* $\mathcal{A}_{in}$ *M*)}›

type-synonym *bump-heuristics-init* = ‹((*nat*,*nat*)*acids*,*vmtf-remove-int-option-fst-As*, *bool*, *nat list* × *bool list*) *tuple4*›

abbreviation *Bump-Heuristics-Init* :: ‹- ⇒ - ⇒ - ⇒ - ⇒ *bump-heuristics-init*› **where**
‹*Bump-Heuristics-Init a b c d* ≡ *Tuple4 a b c d*›

lemmas *bump-heuristics-init-splits* = *Tuple4.tuple4.splits*
hide-fact *tuple4.splits*

abbreviation *get-stable-heuristics* :: ‹*bump-heuristics-init* ⇒ (*nat*,*nat*) *acids*› **where**
‹*get-stable-heuristics* ≡ *Tuple4-a*›

abbreviation *get-focused-heuristics* :: ‹*bump-heuristics-init* ⇒ *vmtf-remove-int-option-fst-As*› **where**
‹*get-focused-heuristics* ≡ *Tuple4-b*›

abbreviation *is-focused-heuristics* :: ‹*bump-heuristics-init* ⇒ *bool*› **where**

‹*is-focused-heuristics* ≡ *Tuple4-c*›

abbreviation *is-stable-heuristics*:: ‹*bump-heuristics-init* ⇒ *bool*› **where**
‹*is-stable-heuristics x* ≡ ¬*is-focused-heuristics x*›

abbreviation *get-bumped-variables* :: ‹*bump-heuristics-init* ⇒ *nat list* × *bool list*› **where**
‹*get-bumped-variables* ≡ *Tuple4-d*›

abbreviation *set-stable-heuristics* :: ‹(*nat*,*nat*)*acids* ⇒*bump-heuristics-init* ⇒ -› **where**
‹*set-stable-heuristics* ≡ *Tuple4.set-a*›

abbreviation *set-focused-heuristics* :: ‹*vmtf-remove-int-option-fst-As* ⇒*bump-heuristics-init* ⇒ -› **where**
‹*set-focused-heuristics* ≡ *Tuple4.set-b*›

abbreviation *set-is-focused-heuristics* :: ‹*bool* ⇒*bump-heuristics-init* ⇒ -› **where**
‹*set-is-focused-heuristics* ≡ *Tuple4.set-c*›

abbreviation *set-bumped-variables* :: ‹*nat list* × *bool list* ⇒*bump-heuristics-init* ⇒ -› **where**
‹*set-bumped-variables* ≡ *Tuple4.set-d*›

definition *get-unit-trail* **where**
‹*get-unit-trail M* = (*rev* (*takeWhile* (λ*x*. ¬*is-decided x*) (*rev M*)))›

definition *bump-heur-init* :: ‹- ⇒ - ⇒ *bump-heuristics-init set*› **where**
‹*bump-heur-init* 𝒜 *M* = {*x*.
  get-stable-heuristics x ∈ *acids* 𝒜 *M* ∧
  get-focused-heuristics x ∈ *isa-vmtf-init* 𝒜 *M* ∧
(*get-bumped-variables x*, *set* (*fst* (*get-bumped-variables x*))) ∈ *distinct-atoms-rel* 𝒜 ∧
count-decided M = *0*
}›

lemma *get-unit-trail-count-decided-0*[*simp*]: ‹*count-decided M* = *0* ⟹ *get-unit-trail M* = *M*›
by (*auto simp*: *get-unit-trail-def count-decided-0-iff*)
(*metis rev-swap set-rev takeWhile-eq-all-conv*)

16.2.1 Access Function

definition *vmtf-heur-import-variable* :: ‹*nat* ⇒ *vmtf-remove-int-option-fst-As* ⇒ -› **where**
‹*vmtf-heur-import-variable L* = (λ(*n*, *stmp*, *fst*, *last*, *cnext*).
  (*vmtf-cons n L cnext stmp*, *stmp+1*, *fst*, *Some L*, *cnext*))›

definition *acids-heur-import-variable* :: ‹*nat* ⇒ (*nat*, *nat*) *acids* ⇒ -› **where**
‹*acids-heur-import-variable L* = (λ(*bw*, *m*). *do* {
  ASSERT (*m* ≤ *unat32-max*);
  bw ← *ACIDS.mop-prio-insert L m bw*;
  RETURN (*bw*, (*m+1*))
})›

definition *initialise-VMTF* :: ‹*nat list* ⇒ *nat* ⇒ *vmtf-remove-int-option-fst-As nres*› **where**
‹*initialise-VMTF N n* = *do* {
  let A = *replicate n* (*VMTF-Node 0 None None*);
  ASSERT(*length N* ≤ *unat32-max*);
  (*n*, *A*, *cnext*) ← *WHILE*$_T$

```
    (λ(i, A, cnext). i < length-uint32-nat N)
    (λ(i, A, cnext). do {
      ASSERT(i < length-uint32-nat N);
      let L = (N ! (length N − 1 − i));
      ASSERT(L < length A);
      ASSERT(cnext ≠ None ⟶ the cnext < length A);
      ASSERT(i + 1 ≤ unat32-max);
      RETURN (i + 1, vmtf-cons A L cnext (i), Some L)
    })
    (0, A, None);
  RETURN ((A, n, cnext, (if N = [] then None else Some ((N!(length N − 1)))), cnext))
 }›

definition init-ACIDS0 :: ‹- ⇒ nat ⇒ (nat multiset × nat multiset × (nat ⇒ nat)) nres› where
 ‹init-ACIDS0 A n = do {
   ASSERT (A ≠ {#} ⟶ n > Max (insert 0 (set-mset A)));
   RETURN ((A, {#}, λ-. 0))
 }›

definition hp-init-ACIDS0 where
 ‹hp-init-ACIDS0 - n = do {
   RETURN ((replicate n None, replicate n None, replicate n None, replicate n None, replicate n 0,
None))
 }›

lemma hp-acids-empty:
 ‹(hp-init-ACIDS0 A, init-ACIDS0 A) ∈
  Id →f ⟨((⟨nat-rel⟩option-rel, ⟨nat-rel⟩option-rel⟩pairing-heaps-rel)) O
  acids-encoded-hmrel⟩nres-rel›
proof −
  have 1: ‹((A, (λ-. None, λ-. None, λ-. None, λ-. None, λ-. Some 0), None), (A, {#}, λ-. 0)) ∈
acids-encoded-hmrel›
    by (auto simp: acids-encoded-hmrel-def pairing-heaps-rel-def map-fun-rel-def
      ACIDS.hmrel-def encoded-hp-prop-list-conc-def encoded-hp-prop-def empty-outside-def
      intro!: relcompI)
  have H: ‹mset-nodes ya ≠ {#}› for ya
    by (cases ya) auto
  show ?thesis
    unfolding uncurry0-def hp-init-ACIDS0-def init-ACIDS0-def
    apply (intro frefI nres-relI)
    apply refine-rcg
    apply (rule relcompI[of])
    defer
    apply (rule 1)
    by (auto simp add: acids-encoded-hmrel-def  encoded-hp-prop-def hp-init-ACIDS0-def
      ACIDS.hmrel-def encoded-hp-prop-list-conc-def pairing-heaps-rel-def H map-fun-rel-def
      split: option.splits dest!: multi-member-split)
qed

definition init-ACIDS where
 ‹init-ACIDS A n = do {
 ac ← init-ACIDS0 A n;
 RETURN (ac, 0)
 }›
```

definition *initialise-ACIDS* :: ‹*nat list* ⇒ *nat* ⇒ (*nat*, *nat*) *acids nres*› **where**
‹*initialise-ACIDS N n* = *do* {
  A ← *init-ACIDS* (*mset N*) *n*;
  ASSERT(*length N* ≤ *unat32-max*);
  (*n*, *A*) ← $WHILE_T$ $^{\lambda\text{-}.\ True}$
    (λ(*i*, *A*). *i* < *length-uint32-nat N*)
    (λ(*i*, *A*). *do* {
      ASSERT(*i* < *length-uint32-nat N*);
      let L = (*N* ! *i*);
      ASSERT (*snd A* = *i*);
       ASSERT(*i* + *1* ≤ *unat32-max*);
      A ← *acids-heur-import-variable L A*;
      RETURN (*i* + *1*, *A*)
    })
    (*0*, *A*);
  RETURN A
 }›

definition *initialise-ACIDS-rev* **where**
 ‹*initialise-ACIDS-rev N* = *initialise-ACIDS* (*rev N*)›

definition (**in** −) *distinct-atms-empty* **where**
 ‹*distinct-atms-empty* - = {}›

definition (**in** −) *distinct-atms-int-empty* **where**
 ‹*distinct-atms-int-empty n* = *RETURN* ([], *replicate n False*)›

lemma *distinct-atms-int-empty-distinct-atms-empty*:
 ‹(*distinct-atms-int-empty*, *RETURN o distinct-atms-empty*) ∈
   [λ*n*. (∀ *L*∈#$\mathcal{L}_{all}$ $\mathcal{A}$. *atm-of L* < *n*)]$_f$ *nat-rel* → ⟨*distinct-atoms-rel* $\mathcal{A}$⟩*nres-rel*›
 apply (*intro frefI nres-relI*)
 apply (*auto simp*: *distinct-atoms-rel-alt-def distinct-atms-empty-def distinct-atms-int-empty-def*)
 by (*metis atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *atms-of-def imageE*)

lemma *initialise-VMTF*:
 shows ‹(*uncurry initialise-VMTF*, *uncurry* (λ*N n*. *RES* (*isa-vmtf-init N* []))) ∈
   [λ(*N*,*n*). (∀ *L*∈# *N*. *L* < *n*) ∧ (*distinct-mset N*) ∧ *size N* < *unat32-max* ∧ *set-mset N* = *set-mset* $\mathcal{A}$]$_f$
   (⟨*nat-rel*⟩*list-rel-mset-rel*) ×$_f$ *nat-rel* → ⟨*Id*⟩*nres-rel*›
  (**is** ‹(*?init*, *?R*) ∈ -›)
proof −
 have *vmtf-ns-notin-empty*: ‹*vmtf-ns-notin* [] *0* (*replicate n* (*VMTF-Node 0 None None*))› **for** *n*
  unfolding *vmtf-ns-notin-def*
  by *auto*

 have *K2*: ‹*distinct N* ⟹ *fst-As* < *length N* ⟹ *N*!*fst-As* ∈ *set* (*take fst-As N*) ⟹ *False*›
  for *fst-As x N*
  by (*metis* (*no-types*, *lifting*) *in-set-conv-nth length-take less-not-refl min-less-iff-conj*
   nth-eq-iff-index-eq nth-take)
 have *W-ref*: ‹$WHILE_T$ (λ(*i*, *A*, *cnext*). *i* < *length-uint32-nat N'*)
     (λ(*i*, *A*, *cnext*). *do* {
       - ← *ASSERT* (*i* < *length-uint32-nat N'*);
       let L = (*N'* ! (*length N'* − *1* − *i*));
       - ← *ASSERT* (*L* < *length A*);
       - ← *ASSERT* (*cnext* ≠ *None* ⟶ *the cnext* < *length A*);

```
          - ← ASSERT (i + 1 ≤ unat32-max);
          RETURN
           (i + 1,
            vmtf-cons A L cnext (i), Some L)
         })
     (0, replicate n' (VMTF-Node 0 None None),
      None)
 ≤ SPEC(λ(i, A', cnext).
   vmtf-ns (rev ((take i (rev N')))) i A'
    ∧ cnext = (option-last (take i (rev N'))) ∧  i = length N' ∧
      length A' = n ∧ vmtf-ns-notin (rev ((take i (rev N')))) i A'
  )›
 (is ‹- ≤ SPEC ?P›)
 if H: ‹case y of (N, n) ⇒(∀ L∈# N. L < n) ∧ distinct-mset N ∧ size N < unat32-max ∧
     set-mset N = set-mset A› and
   ref: ‹(x, y) ∈ ⟨Id⟩list-rel-mset-rel ×f nat-rel› and
   st[simp]: ‹x = (N', n')› ‹y = (N, n)›
  for N N' n n' A x y
proof −
have [simp]: ‹n = n'› and NN': ‹(N', N) ∈ ⟨Id⟩list-rel-mset-rel›
 using ref unfolding st by auto
then have dist: ‹distinct N'›
 using NN' H by (auto simp: list-rel-def br-def list-mset-rel-def list.rel-eq
   list-all2-op-eq-map-right-iff' distinct-image-mset-inj list-rel-mset-rel-def)

have L-N: ‹L < n› if ‹L∈set N'› for L
 using H ref that by (auto simp: list-rel-def br-def list-mset-rel-def
   list-all2-op-eq-map-right-iff' list-rel-mset-rel-def list.rel-eq)
let ?Q = ‹λ(i, A', cnext).
   vmtf-ns (rev ((take i (rev N')))) i A' ∧ i ≤ length N' ∧
   cnext = (option-last (take i (rev N'))) ∧
 length A' = n ∧ vmtf-ns-notin (rev ((take i (rev N')))) i A'›
have[simp]: ‹N' ! (length N' − Suc a) ∉ set (take a (rev N'))› if ‹a < length N'› for a
 by (metis K2 dist distinct-rev length-rev rev-nth that)
show ?thesis
 apply (refine-vcg WHILET-rule[where R = ‹measure (λ(i, -). length N' + 1 − i)› and I = ‹?Q›])
 subgoal by auto
 subgoal by (auto intro: vmtf-ns.intros)
 subgoal by auto
 subgoal by auto
 subgoal by auto
 subgoal for S N' x2 A'
   unfolding assert-bind-spec-conv vmtf-ns-notin-def
   using L-N dist
   by (auto 5 5 simp: take-Suc-conv-app-nth hd-drop-conv-nth nat-shiftr-div2
       option-last-def hd-rev last-map intro!: vmtf-cons dest: K2)
 subgoal by auto
 subgoal for s
   using L-N[of ‹N' ! (length N' − 1 − fst s)›] dist nth-mem[of ‹length N' − 1 − fst s› N']
   by (auto simp: take-Suc-conv-app-nth hd-drop-conv-nth nat-shiftr-div2
     option-last-def hd-rev last-map simp del: nth-mem)
 subgoal
   using L-N dist
   by (auto simp: last-take-nth-conv option-last-def nth-rev)
 subgoal
   using H dist ref
```

by (*auto simp*: *last-take-nth-conv option-last-def list-rel-mset-rel-imp-same-length*)
subgoal
using *L-N dist*
by (*auto 5 5 simp*: *take-Suc-conv-app-nth option-last-def hd-rev nth-rev last-map intro*!: *vmtf-cons dest*: *K2*)
subgoal by (*auto simp*: *take-Suc-conv-app-nth*)
subgoal by (*auto simp*: *take-Suc-conv-app-nth nth-rev*)
subgoal by *auto*
subgoal
using *L-N dist*
by (*auto 5 5 simp*: *take-Suc-conv-app-nth hd-rev last-map option-last-def nth-rev intro*!: *vmtf-notin-vmtf-cons dest*: *K2 split*: *if-splits*)
subgoal by *auto*
subgoal by (*auto simp*: *nth-rev*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed
have [*simp*]: ‹*vmtf-*$\mathcal{L}_{all}$ *n'* [] ((*set N*, {}))›
if ‹(*N*, *n'*) ∈ ⟨*Id*⟩*list-rel-mset-rel*› **for** *N N' n'*
using *that* **unfolding** *vmtf-*$\mathcal{L}_{all}$*-def*
by (*auto simp*: $\mathcal{L}_{all}$*-def atms-of-def image-image image-Un list-rel-def br-def list-mset-rel-def list-all2-op-eq-map-right-iff' list-rel-mset-rel-def list.rel-eq*)
have *in-N-in-N1*: ‹*L* ∈ *set N'* ⟹ *L* ∈ *atms-of* ($\mathcal{L}_{all}$ *N*)›
if ‹(*N'*, *N*) ∈ *list-mset-rel*› **for** *L N N' y*
using *that* **by** (*auto simp*: $\mathcal{L}_{all}$*-def atms-of-def image-image image-Un list-rel-def list.rel-eq br-def list-mset-rel-def list-all2-op-eq-map-right-iff'*)

have *length-ba*: ‹∀ *L*∈# *N*. *L* < *length ba* ⟹ *L* ∈ *atms-of* ($\mathcal{L}_{all}$ *N*) ⟹ *L* < *length ba*›
if ‹(*N'*, *y*) ∈ ⟨*Id*⟩*list-rel-mset-rel*›
for *L ba N N' y*
using *that*
by (*auto simp*: $\mathcal{L}_{all}$*-def nat-shiftr-div2 list.rel-eq atms-of-def image-image image-Un split*: *if-splits*)
show *?thesis*
supply *list.rel-eq*[*simp*]
apply (*intro frefI nres-relI*)
unfolding *initialise-VMTF-def uncurry-def conc-Id id-def isa-vmtf-init-def distinct-atms-int-empty-def nres-monad1*
apply (*subst Let-def*)
apply (*refine-vcg*)
subgoal by (*auto dest*: *list-rel-mset-rel-imp-same-length*)
apply (*rule W-ref*[*THEN order-trans*]; *assumption?*)
subgoal for *N' N'n' n' Nn*
apply (*auto dest*: *list-rel-mset-rel-imp-same-length simp*: *vmtf-def*)
apply (*rule exI*[*of* - ‹((*fst N'*))›])
apply (*rule-tac exI*[*of* - ‹[]›])
apply (*auto dest*: *list-rel-mset-rel-imp-same-length simp*: *vmtf-def hd-rev last-conv-nth rev-nth atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$)
apply (*auto dest*: *list-rel-mset-rel-imp-same-length simp*: *vmtf-def hd-rev last-conv-nth rev-nth atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$ *list-rel-mset-rel-def list-mset-rel-def br-def hd-conv-nth*)
done

```
    done
qed


lemma initialise-ACIDS:
  shows ‹(uncurry initialise-ACIDS, uncurry (λN n. RES (acids N ([]::(nat,nat)ann-lits)))) ∈
     [λ(N,n). (∀ L∈# N. L < n) ∧ (distinct-mset N) ∧ size N < unat32-max ∧ set-mset N = set-mset
𝒜]_f
      (⟨nat-rel⟩list-rel-mset-rel) ×_f nat-rel → ⟨Id⟩nres-rel›
    (is ‹(?init, ?R) ∈ -›)
proof −
   define I' where ‹I' x ≡ (λ(a,b::(nat,nat)acids). b ∈ acids 𝒜 (map (λa. (Decided (Pos a)) ::
(nat,nat)ann-lit) (drop a (fst x))) ∧ a ≤ length (fst x) ∧
    snd b = a ∧ fst (snd (fst b)) = mset (take a (fst x)))› for x :: ‹nat list × nat›
 have I0: ‹(∀ L∈#fst y. L < snd y) ∧
    distinct-mset (fst y) ∧
    size (fst y) < unat32-max ∧ set-mset (fst y) = set-mset 𝒜 ⟹
    (x, y) ∈ ⟨nat-rel⟩list-rel-mset-rel ×_f nat-rel ⟹
   length (fst x) ≤ unat32-max ⟹ I' x (0, (mset (fst x), {#}, λ-. 0), 0)› for x y
   by (cases x, cases y)
    (auto simp: I'-def acids-def list-rel-mset-rel-def list-mset-rel-def br-def defined-lit-map
     dest!: multi-member-split)

 have I'-Suc: ‹I' x (fst s + 1,
      (fst (fst (snd s)), add-mset (fst x ! fst s) (fst (snd (fst (snd s)))),
    (snd (snd (fst (snd s))))(fst x ! fst s := snd (snd s))),
   snd (snd s) + 1)› and
   pre: ‹fst x ! fst s ∉# fst (snd (fst (snd s)))›
   if
     ‹(∀ L∈#fst y. L < snd y) ∧
     distinct-mset (fst y) ∧
     size (fst y) < unat32-max ∧ set-mset (fst y) = set-mset 𝒜› and
     ‹(x, y) ∈ ⟨nat-rel⟩list-rel-mset-rel ×_f nat-rel› and
     ‹x = (a, b)› and
     ‹y = (aa, ba)› and
     ‹length (fst x) ≤ unat32-max› and
     ‹I' x s› and
     ‹fst s < length-uint32-nat (fst x)› and
     ‹fst s < length-uint32-nat (fst x)› and
     ‹snd (snd s) = fst s› and
     ‹fst s + 1 ≤ unat32-max›
   for a b aa ba s x y
 proof −
   have ‹mset (take (Suc (fst s)) a) ⊆# 𝒜›
     apply (rule subset-mset.trans[of - ‹mset a›])
     using that
     apply (clarsimp-all simp: acids-def defined-lit-map list-rel-mset-rel-def br-def list-mset-rel-def
       image-image acids-heur-import-variable-def I'-def mset-take-subseteq)
     by (metis distinct-subseteq-iff fst-eqD le-refl set-take-subset take-all-iff that(1))

   then show ‹I' x (fst s + 1,
      (fst (fst (snd s)), add-mset (fst x ! fst s) (fst (snd (fst (snd s))))),
    (snd (snd (fst (snd s))))(fst x ! fst s := snd (snd s))),
   snd (snd s) + 1)› ‹fst x ! fst s ∉# fst (snd (fst (snd s)))›
     using that
    by (force simp: acids-def take-Suc-conv-app-nth defined-lit-map list-rel-mset-rel-def br-def list-mset-rel-def
```

image-image acids-heur-import-variable-def I′-def Cons-nth-drop-Suc[*symmetric*] *distinct-in-set-take-iff*
dest: *multi-member-split*)+
qed

show *?thesis*
unfolding *uncurry-def case-prod-beta initialise-ACIDS-def*
acids-heur-import-variable-def ACIDS.mop-prio-insert-def nres-monad3
init-ACIDS-def nres-monad3 init-ACIDS0-def
bind-to-let-conv Let-def
apply (*intro frefI nres-relI*)
subgoal for *x y*
apply (*cases x*, *cases y*)
apply (*refine-vcg specify-left*[*OF WHILEIT-rule-stronger-inv*[**where** Φ = ‹λ(*a*::*nat*,*b*::(*nat*,*nat*)*acids*).
b ∈ *acids* *A* ([]::(*nat*,*nat*)*ann-lits*)› **and**
I′ = ‹*I′ x*› **and**
R = ‹*measure* (λ(*a*,*b*). *length* (*fst x*) − *a*)›]])
subgoal **apply** (*auto simp*: *list-rel-mset-rel-def list-mset-rel-def br-def acids-def*)
using *gt-or-eq-0* **by** *blast*
subgoal by (*auto simp*: *list-rel-mset-rel-def list-mset-rel-def br-def*)
subgoal by *auto*
subgoal by *auto*
subgoal by (*rule I0*)
subgoal by (*auto simp*: *I′-def*)
subgoal by (*auto simp*:)
subgoal by *auto*
subgoal by (*rule pre*)
subgoal by (*auto simp*: *I′-def list-rel-mset-rel-def list-mset-rel-def br-def*
acids-def)
subgoal by (*rule I′-Suc*)
subgoal by (*auto simp*: *I′-def*)
subgoal
by (*auto simp add*: *I′-def*)
subgoal apply (*auto simp*: *list-rel-mset-rel-def list-mset-rel-def br-def acids-def*)
by (*metis distinct-subseteq-iff set-mset-mset*)
done
done
qed

lemma *initialise-ACIDS-rev*:
shows ‹(*uncurry initialise-ACIDS-rev*, *uncurry* (λ*N n*. *RES* (*acids N* ([]::(*nat*,*nat*)*ann-lits*)))) ∈
[λ(*N*,*n*). (∀ *L*∈# *N*. *L* < *n*) ∧ (*distinct-mset N*) ∧ *size N* < *unat32-max* ∧ *set-mset N* = *set-mset*
A]$_f$
(⟨*nat-rel*⟩*list-rel-mset-rel*) ×$_f$ *nat-rel* → ⟨*Id*⟩*nres-rel*›
unfolding *initialise-ACIDS-rev-def*
apply (*intro frefI nres-relI*)+
unfolding *uncurry-def case-prod-beta*
apply (*rule initialise-ACIDS*[**where** *A*=*A*, *THEN fref-to-Down-curry*])
by (*auto simp*: *list-mset-rel-def list-rel-mset-rel-def br-def*)

definition *initialize-Bump-Init* :: ‹*nat list* ⇒ *nat* ⇒ *bump-heuristics-init nres*› **where**
‹*initialize-Bump-Init A n* = *do* {
focused ← *initialise-VMTF A n*;
hstable ← *initialise-ACIDS-rev A n*;
to-remove ← *distinct-atms-int-empty n*;
RETURN (*Tuple4 hstable focused True to-remove*)

}›

lemma *specify-left-RES*:
assumes $m \le RES\ \Phi$
assumes $\bigwedge x.\ x \in \Phi \Longrightarrow f\ x \le M$
shows $do\ \{\ x \leftarrow m;\ f\ x\ \} \le M$
using *assms* **by** (*auto simp*: *pw-le-iff refine-pw-simps*)

lemma *initialize-Bump-Init*:
shows ‹(*uncurry initialize-Bump-Init*, *uncurry* ($\lambda N\ n.\ RES$ (*bump-heur-init N* [])))
$\in$ [$\lambda(N,n).\ (\forall L{\in}\#\ N.\ L < n) \wedge$ (*distinct-mset N*) $\wedge$ *size N* $<$ *unat32-max* $\wedge$ *set-mset N* $=$ *set-mset* $\mathcal{A}$]$_f$
(⟨*nat-rel*⟩*list-rel-mset-rel*) $\times_f$ *nat-rel* $\to$ ⟨*Id*⟩*nres-rel*›
(**is** ‹(*?init*, *?R*) $\in$ -›)
unfolding *initialize-Bump-Init-def uncurry-def distinct-atms-int-empty-def nres-monad1*
apply (*intro frefI nres-relI*)
apply (*refine-rcg*)
apply *hypsubst*
apply (*rule specify-left-RES*[*OF initialise-VMTF*[**where** $\mathcal{A}$=$\mathcal{A}$, *THEN fref-to-Down-curry*, *unfolded conc-fun-RES*]])
apply *assumption*+
apply (*rule specify-left-RES*[*OF initialise-ACIDS-rev*[**where** $\mathcal{A}$=$\mathcal{A}$, *THEN fref-to-Down-curry*, *unfolded conc-fun-RES*]])
apply *assumption*+
apply (*auto simp*: *bump-heur-init-def distinct-atoms-rel-def distinct-hash-atoms-rel-def atoms-hash-rel-def intro*!: *relcompI*)
done

type-synonym *bump-heuristics-option-fst-As* = ‹*vmtf-remove-int-option-fst-As*›

lemma *isa-vmtf-init-consD*:
‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*)) $\in$ *isa-vmtf-init* $\mathcal{A}$ *M* $\Longrightarrow$
((*ns*, *m*, *fst-As*, *lst-As*, *next-search*)) $\in$ *isa-vmtf-init* $\mathcal{A}$ (*L* # *M*)›
by (*auto simp*: *isa-vmtf-init-def dest*: *vmtf-consD*)

lemma *isa-vmtf-init-consD'*:
‹*vm* $\in$ *isa-vmtf-init* $\mathcal{A}$ *M* $\Longrightarrow$ *vm* $\in$ *isa-vmtf-init* $\mathcal{A}$ (*L* # *M*)›
by (*auto simp*: *isa-vmtf-init-def dest*: *vmtf-consD*)

lemma *vmtf-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ *L* $\in$ *vmtf* $\mathcal{A}$ *M* $\Longrightarrow$ *L* $\in$ *vmtf* $\mathcal{B}$ *M*›
using $\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
unfolding *vmtf-def vmtf-*$\mathcal{L}_{all}$*-def*
by *auto*
lemma *isa-vmtf-init-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ *isa-vmtf-init* $\mathcal{A}$ *M* =*isa-vmtf-init* $\mathcal{B}$ *M*›
using $\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *vmtf-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *vmtf-cong*[*of* $\mathcal{B}$ $\mathcal{A}$]
unfolding *isa-vmtf-init-def vmtf-*$\mathcal{L}_{all}$*-def*
by *auto*
lemma *isa-acids-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ $\Longrightarrow$ *acids* $\mathcal{A}$ = *acids* $\mathcal{B}$›
using $\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
unfolding *acids-def*

by (*auto intro*!: *ext intro*!: *distinct-subseteq-iff*[*THEN iffD1*])

lemma *distinct-atoms-rel-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *distinct-atoms-rel* $\mathcal{A}$ = *distinct-atoms-rel* $\mathcal{B}$›
using $\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
unfolding *vmtf-def vmtf-*$\mathcal{L}_{all}$*-def distinct-atoms-rel-def atoms-hash-rel-def distinct-hash-atoms-rel-def*
by *auto*

lemma *bump-heur-init-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *bump-heur-init* $\mathcal{A}$ *M* = *bump-heur-init* $\mathcal{B}$ *M*›
using *isa-vmtf-init-cong*[*of* $\mathcal{A}$ $\mathcal{B}$ *M*]
$\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *distinct-atoms-rel-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *isa-acids-cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
unfolding *bump-heur-init-def isa-acids-cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
by *auto*

lemma *bump-heur-init-consD'*:
‹*vm* ∈ *bump-heur-init* $\mathcal{A}$ *M* ⟹ *vm* ∈ *bump-heur-init* $\mathcal{A}$ (*Propagated L n* # *M*)›
by (*auto simp*: *bump-heur-init-def dest*: *isa-vmtf-init-consD' acids-prepend*)

end
theory *IsaSAT-Initialisation*
imports *Watched-Literals.Watched-Literals-Watch-List-Initialisation*
IsaSAT-Setup IsaSAT-VMTF WB-More-Word
IsaSAT-Mark Tuple15
IsaSAT-Bump-Heuristics-Init-State
Automatic-Refinement.Relators — for more lemmas
begin
lemma *in-mset-rel-eq-f-iff*:
‹(*a*, *b*) ∈ ⟨{(*c*, *a*). *a* = *f c*}⟩*mset-rel* ⟷ *b* = *f* `# *a*›
using *ex-mset*[*of a*]
by (*auto simp*: *mset-rel-def br-def rel2p-def*[*abs-def*] *p2rel-def rel-mset-def*
list-all2-op-eq-map-right-iff' *cong*: *ex-cong*)

lemma *in-mset-rel-eq-f-iff-set*:
‹⟨{(*c*, *a*). *a* = *f c*}⟩*mset-rel* = {(*b*, *a*). *a* = *f* `# *b*}›
using *in-mset-rel-eq-f-iff*[*of - - f*] **by** *blast*

Chapter 17

Initialisation

17.1 Code for the initialisation of the Data Structure

The initialisation is done in three different steps:

1. First, we extract all the atoms that appear in the problem and initialise the state with empty values. This part is called *initialisation* below.
2. Then, we go over all clauses and insert them in our memory module. We call this phase *parsing*.
3. Finally, we calculate the watch list.

Splitting the second from the third step makes it easier to add preprocessing and more important to add a bounded mode.

17.1.1 Initialisation of the state

definition (**in** −) *atoms-hash-empty* **where**
[*simp*]: ‹*atoms-hash-empty* - = {}›

definition (**in** −) *atoms-hash-int-empty* **where**
‹*atoms-hash-int-empty n* = *RETURN* (*replicate n False*)›

lemma *atoms-hash-int-empty-atoms-hash-empty*:
‹(*atoms-hash-int-empty*, *RETURN o atoms-hash-empty*) ∈
[λ*n*. (∀ *L*∈#$\mathcal{L}_{all}$ $\mathcal{A}$. *atm-of L* < *n*)]$_f$ *nat-rel* → ⟨*atoms-hash-rel* $\mathcal{A}$⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*use Max-less-iff* **in** ‹*auto simp*: *atoms-hash-rel-def atoms-hash-int-empty-def atoms-hash-empty-def*
in-$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff Ball-def*
dest: *spec*[*of* - ‹*Pos* -›]›)

type-synonym (**in** −) *twl-st-wl-heur-init* =
‹(*trail-pol*, *arena*, *conflict-option-rel*, *nat*,
(*nat* × *nat literal* × *bool*) *list list*, *bump-heuristics-init*, *bool list*,
nat, *conflict-min-cach-l*, *lbd*, *vdom*, *vdom*, *bool*, *clss-size*, *lookup-clause-rel*) *tuple15*›

type-synonym (**in** −) *twl-st-wl-heur-init-full* =

‹(*trail-pol*, *arena*, *conflict-option-rel*, *nat*,
(*nat* × *nat literal* × *bool*) *list list*, *bump-heuristics-init*, *bool list*,
nat, *conflict-min-cach-l*, *lbd*, *vdom*, *vdom*, *bool*, *clss-size*, *lookup-clause-rel*) *tuple15*›

abbreviation *get-trail-init-wl-heur* :: ‹*twl-st-wl-heur-init* ⇒ *trail-pol*› **where**
‹*get-trail-init-wl-heur* ≡ *Tuple15-a*›

abbreviation *set-trail-init-wl-heur* :: ‹*trail-pol* ⇒ - ⇒ *twl-st-wl-heur-init*› **where**
‹*set-trail-init-wl-heur* ≡ *Tuple15.set-a*›

abbreviation *get-clauses-wl-heur-init* :: ‹*twl-st-wl-heur-init* ⇒ *arena*› **where**
‹*get-clauses-wl-heur-init* ≡ *Tuple15-b*›

abbreviation *set-clauses-wl-heur-init* :: ‹*arena* ⇒ - ⇒ *twl-st-wl-heur-init*› **where**
‹*set-clauses-wl-heur-init* ≡ *Tuple15.set-b*›

abbreviation *get-conflict-wl-heur-init* :: ‹*twl-st-wl-heur-init* ⇒ *conflict-option-rel*› **where**
‹*get-conflict-wl-heur-init* ≡ *Tuple15-c*›

abbreviation *set-conflict-wl-heur-init* :: ‹*conflict-option-rel* ⇒ - ⇒ *twl-st-wl-heur-init*› **where**
‹*set-conflict-wl-heur-init* ≡ *Tuple15.set-c*›

abbreviation *get-literals-to-update-wl-init* :: ‹*twl-st-wl-heur-init* ⇒ *nat*› **where**
‹*get-literals-to-update-wl-init* ≡ *Tuple15-d*›

abbreviation *set-literals-to-update-wl-init* :: ‹*nat* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init*› **where**
‹*set-literals-to-update-wl-init* ≡ *Tuple15.set-d*›

abbreviation *get-watchlist-wl-heur-init* :: ‹*twl-st-wl-heur-init* ⇒ (*nat* × *nat literal* × *bool*) *list list*›
where
‹*get-watchlist-wl-heur-init* ≡ *Tuple15-e*›

abbreviation *set-watchlist-wl-heur-init* :: ‹(*nat* × *nat literal* × *bool*) *list list* ⇒ *twl-st-wl-heur-init* ⇒
twl-st-wl-heur-init› **where**
‹*set-watchlist-wl-heur-init* ≡ *Tuple15.set-e*›

abbreviation *get-vmtf-wl-heur-init* :: ‹*twl-st-wl-heur-init* ⇒ *bump-heuristics-init*› **where**
‹*get-vmtf-wl-heur-init* ≡ *Tuple15-f*›

abbreviation *set-vmtf-wl-heur-init* :: ‹*bump-heuristics-init* ⇒ *twl-st-wl-heur-init* ⇒ -› **where**
‹*set-vmtf-wl-heur-init* ≡ *Tuple15.set-f*›

abbreviation *get-phases-wl-heur-init* :: ‹*twl-st-wl-heur-init* ⇒ *bool list*› **where**
‹*get-phases-wl-heur-init* ≡ *Tuple15-g*›

abbreviation *set-phases-wl-heur-init* :: ‹*bool list* ⇒ *twl-st-wl-heur-init* ⇒ -› **where**
‹*set-phases-wl-heur-init* ≡ *Tuple15.set-g*›

abbreviation *get-clvls-wl-heur-init* :: ‹*twl-st-wl-heur-init* ⇒ *nat*› **where**
‹*get-clvls-wl-heur-init* ≡ *Tuple15-h*›

abbreviation *set-clvls-wl-heur-init* :: ‹*nat* ⇒ *twl-st-wl-heur-init* ⇒ -› **where**
‹*set-clvls-wl-heur-init* ≡ *Tuple15.set-h*›

abbreviation *get-cach-wl-heur-init* :: ‹*twl-st-wl-heur-init* ⇒ *conflict-min-cach-l*› **where**
‹*get-cach-wl-heur-init* ≡ *Tuple15-i*›

abbreviation *set-cach-wl-heur-init* :: ‹*conflict-min-cach-l* ⇒ *twl-st-wl-heur-init* ⇒ -› **where**
‹*set-cach-wl-heur-init* ≡ *Tuple15.set-i*›

abbreviation *get-lbd-wl-heur-init* :: ‹*twl-st-wl-heur-init* ⇒ *lbd*› **where**
‹*get-lbd-wl-heur-init* ≡ *Tuple15-j*›

abbreviation *set-lbd-wl-heur-init* :: ‹*lbd* ⇒ *twl-st-wl-heur-init* ⇒ -› **where**
‹*set-lbd-wl-heur-init* ≡ *Tuple15.set-j*›

abbreviation *get-vdom-heur-init* :: ‹*twl-st-wl-heur-init* ⇒ *vdom*› **where**
‹*get-vdom-heur-init* ≡ *Tuple15-k*›

abbreviation *set-vdom-heur-init* :: ‹*vdom* ⇒ *twl-st-wl-heur-init* ⇒ -› **where**
‹*set-vdom-heur-init* ≡ *Tuple15.set-k*›

abbreviation *get-ivdom-heur-init* :: ‹*twl-st-wl-heur-init* ⇒ *vdom*› **where**
‹*get-ivdom-heur-init* ≡ *Tuple15-l*›

abbreviation *set-ivdom-heur-init* :: ‹*vdom* ⇒ *twl-st-wl-heur-init* ⇒ -› **where**
‹*set-ivdom-heur-init* ≡ *Tuple15.set-l*›

abbreviation *is-failed-heur-init* :: ‹*twl-st-wl-heur-init* ⇒ *bool*› **where**
‹*is-failed-heur-init* ≡ *Tuple15-m*›

abbreviation *set-failed-heur-init* :: ‹*bool* ⇒ *twl-st-wl-heur-init* ⇒ -› **where**
‹*set-failed-heur-init* ≡ *Tuple15.set-m*›

abbreviation *get-learned-count-init* :: ‹*twl-st-wl-heur-init* ⇒ *clss-size*› **where**
‹*get-learned-count-init* ≡ *Tuple15-n*›

abbreviation *set-learned-count-init* :: ‹*clss-size* ⇒ *twl-st-wl-heur-init* ⇒ -› **where**
‹*set-learned-count-init* ≡ *Tuple15.set-n*›

abbreviation *get-mark-wl-heur-init* :: ‹*twl-st-wl-heur-init* ⇒ *lookup-clause-rel*› **where**
‹*get-mark-wl-heur-init* ≡ *Tuple15-o*›

abbreviation *set-mark-wl-heur-init* :: ‹*lookup-clause-rel* ⇒ *twl-st-wl-heur-init* ⇒ -› **where**
‹*set-mark-wl-heur-init* ≡ *Tuple15.set-o*›

The initialisation relation is stricter in the sense that it already includes the relation of atom inclusion.

Remark that we replace $D = None \longrightarrow j \leq length\ M$ by $j \leq length\ M$: this simplifies the proofs and does not make a difference in the generated code, since there are no conflict analysis at that level anyway.

KILL duplicates below, but difference: vmtf vs vmtf_init watch list vs no WL OC vs non-OC

definition *twl-st-heur-parsing-no-WL*
:: ‹*nat multiset* ⇒ *bool* ⇒ (*twl-st-wl-heur-init* × *nat twl-st-wl-init*) *set*›
where
[*unfolded Let-def*]: ‹*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* =
{(*S*,
((*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*), *OC*)).
let M′ = *get-trail-init-wl-heur S*; *N′* = *get-clauses-wl-heur-init S*; *failed* = *is-failed-heur-init S*;
vdom = *get-vdom-heur-init S*; *ivdom* = *get-ivdom-heur-init S*; *D′* = *get-conflict-wl-heur-init S*; *vm*

= *get-vmtf-wl-heur-init S*;
mark = *get-mark-wl-heur-init S*; *lcount* = *get-learned-count-init S*; W' = *get-watchlist-wl-heur-init S*;
φ = *get-phases-wl-heur-init S*; *cach* = *get-cach-wl-heur-init S*; *lbd* = *get-lbd-wl-heur-init S*;
j = *get-literals-to-update-wl-init S*
in
(*unbdd* $\longrightarrow$ $\neg$*failed*) $\wedge$
((*unbdd* $\vee$ $\neg$*failed*) $\longrightarrow$
(*valid-arena* N' *N* (*set vdom*) $\wedge$
set-mset
(*all-lits-of-mm*
({#*mset* (*fst x*). *x* $\in\#$ *ran-m N*#} + *NE*+*NEk* + *UE* + *UEk* + *NS* + *US* + *N0* + *U0*)) $\subseteq$ *set-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$) $\wedge$
mset vdom = *dom-m N* $\wedge$ *ivdom* = *vdom*)) $\wedge$
(M', *M*) $\in$ *trail-pol* $\mathcal{A}$ $\wedge$
(D', *D*) $\in$ *option-lookup-clause-rel* $\mathcal{A}$ $\wedge$
j $\leq$ *length M* $\wedge$
Q = *uminus* ‘# *lit-of* ‘# *mset* (*drop j* (*rev M*)) $\wedge$
vm $\in$ *bump-heur-init* $\mathcal{A}$ *M* $\wedge$
phase-saving $\mathcal{A}$ φ $\wedge$
no-dup M $\wedge$
cach-refinement-empty $\mathcal{A}$ *cach* $\wedge$
(W', *empty-watched* $\mathcal{A}$) $\in$ $\langle Id \rangle$*map-fun-rel* (D_0 $\mathcal{A}$) $\wedge$
isasat-input-bounded $\mathcal{A}$ $\wedge$
distinct vdom $\wedge$
clss-size-corr N NE UE NEk UEk NS US N0 U0 lcount $\wedge$
(*mark*, {#}) $\in$ *lookup-clause-rel* $\mathcal{A}$
}›

definition *twl-st-heur-parsing*
:: ‹*nat multiset* $\Rightarrow$ *bool* $\Rightarrow$ (*twl-st-wl-heur-init* $\times$ (*nat twl-st-wl* $\times$ *nat clauses*)) *set*›
where
[*unfolded Let-def*]: ‹*twl-st-heur-parsing* $\mathcal{A}$ *unbdd* =
{(*S*,
((*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*,*Q*, *W*), *OC*)).
let M' = *get-trail-init-wl-heur S*; N' = *get-clauses-wl-heur-init S*; *failed* = *is-failed-heur-init S*;
vdom = *get-vdom-heur-init S*; *ivdom* = *get-ivdom-heur-init S*; D' = *get-conflict-wl-heur-init S*; *vm* = *get-vmtf-wl-heur-init S*;
mark = *get-mark-wl-heur-init S*; *lcount* = *get-learned-count-init S*; W' = *get-watchlist-wl-heur-init S*;
φ = *get-phases-wl-heur-init S*; *cach* = *get-cach-wl-heur-init S*; *lbd* = *get-lbd-wl-heur-init S*;
j = *get-literals-to-update-wl-init S*
in
(*unbdd* $\longrightarrow$ $\neg$*failed*) $\wedge$
((*unbdd* $\vee$ $\neg$*failed*) $\longrightarrow$
((M', *M*) $\in$ *trail-pol* $\mathcal{A}$ $\wedge$
valid-arena N' *N* (*set vdom*) $\wedge$
(D', *D*) $\in$ *option-lookup-clause-rel* $\mathcal{A}$ $\wedge$
j $\leq$ *length M* $\wedge$
Q = *uminus* ‘# *lit-of* ‘# *mset* (*drop j* (*rev M*)) $\wedge$
vm $\in$ *bump-heur-init* $\mathcal{A}$ *M* $\wedge$
phase-saving $\mathcal{A}$ φ $\wedge$
no-dup M $\wedge$
cach-refinement-empty $\mathcal{A}$ *cach* $\wedge$
mset vdom = *dom-m N* $\wedge$

$vdom\text{-}m\ \mathcal{A}\ W\ N = set\text{-}mset\ (dom\text{-}m\ N) \wedge$
$set\text{-}mset$
$(all\text{-}lits\text{-}of\text{-}mm$
$(\{\#mset\ (fst\ x).\ x \in\#\ ran\text{-}m\ N\#\} + (NE{+}NEk) + (UE{+}UEk) + NS + US + N0 + U0)) \subseteq$
$set\text{-}mset\ (\mathcal{L}_{all}\ \mathcal{A}) \wedge$
$(W', W) \in \langle Id\rangle map\text{-}fun\text{-}rel\ (D_0\ \mathcal{A}) \wedge$
$isasat\text{-}input\text{-}bounded\ \mathcal{A} \wedge$
$distinct\ vdom \wedge$
$ivdom = vdom \wedge$
$clss\text{-}size\text{-}corr\ N\ NE\ UE\ NEk\ UEk\ NS\ US\ N0\ U0\ lcount \wedge$
$(mark, \{\#\}) \in lookup\text{-}clause\text{-}rel\ \mathcal{A}))$
$\}\rangle$

definition $twl\text{-}st\text{-}heur\text{-}parsing\text{-}no\text{-}WL\text{-}wl :: \langle nat\ multiset \Rightarrow bool \Rightarrow (- \times nat\ twl\text{-}st\text{-}wl\text{-}init') set\rangle$ **where**
$\langle twl\text{-}st\text{-}heur\text{-}parsing\text{-}no\text{-}WL\text{-}wl\ \mathcal{A}\ \ unbdd =$
$\{(S,$
$(M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, Q)).$
$let\ M' = get\text{-}trail\text{-}init\text{-}wl\text{-}heur\ S;\ N' = get\text{-}clauses\text{-}wl\text{-}heur\text{-}init\ S;\ failed = is\text{-}failed\text{-}heur\text{-}init\ S;$
$vdom = get\text{-}vdom\text{-}heur\text{-}init\ S;\ ivdom = get\text{-}ivdom\text{-}heur\text{-}init\ S;\ D' = get\text{-}conflict\text{-}wl\text{-}heur\text{-}init\ S;\ vm$
$= get\text{-}vmtf\text{-}wl\text{-}heur\text{-}init\ S;$
$mark = get\text{-}mark\text{-}wl\text{-}heur\text{-}init\ S;\ lcount = get\text{-}learned\text{-}count\text{-}init\ S;\ W' = get\text{-}watchlist\text{-}wl\text{-}heur\text{-}init$
$S;$
$\varphi = get\text{-}phases\text{-}wl\text{-}heur\text{-}init\ S;\ cach = get\text{-}cach\text{-}wl\text{-}heur\text{-}init\ S;\ lbd = get\text{-}lbd\text{-}wl\text{-}heur\text{-}init\ S;$
$j = get\text{-}literals\text{-}to\text{-}update\text{-}wl\text{-}init\ S$
in
$(unbdd \longrightarrow \neg failed) \wedge$
$((unbdd \vee \neg failed) \longrightarrow$
$(valid\text{-}arena\ N'\ N\ (set\ vdom) \wedge set\text{-}mset\ (dom\text{-}m\ N) \subseteq set\ vdom)) \wedge$
$(M', M) \in trail\text{-}pol\ \mathcal{A} \wedge$
$(D', D) \in option\text{-}lookup\text{-}clause\text{-}rel\ \mathcal{A} \wedge$
$j \leq length\ M \wedge$
$Q = uminus\ `\#\ lit\text{-}of\ `\#\ mset\ (drop\ j\ (rev\ M)) \wedge$
$vm \in bump\text{-}heur\text{-}init\ \mathcal{A}\ M \wedge$
$phase\text{-}saving\ \mathcal{A}\ \varphi \wedge$
$no\text{-}dup\ M \wedge$
$cach\text{-}refinement\text{-}empty\ \mathcal{A}\ cach \wedge$
$set\text{-}mset\ (all\text{-}lits\text{-}of\text{-}mm\ (\{\#mset\ (fst\ x).\ x \in\#\ ran\text{-}m\ N\#\} + (NE{+}NEk) + (UE{+}UEk) + NS +$
$US + N0 + U0))$
$\subseteq set\text{-}mset\ (\mathcal{L}_{all}\ \mathcal{A}) \wedge$
$(W', empty\text{-}watched\ \mathcal{A}) \in \langle Id\rangle map\text{-}fun\text{-}rel\ (D_0\ \mathcal{A}) \wedge$
$isasat\text{-}input\text{-}bounded\ \mathcal{A} \wedge$
$distinct\ vdom \wedge ivdom = vdom \wedge$
$clss\text{-}size\text{-}corr\ N\ NE\ UE\ NEk\ UEk\ NS\ US\ N0\ U0\ lcount \wedge$
$(mark, \{\#\}) \in lookup\text{-}clause\text{-}rel\ \mathcal{A}$
$\}\rangle$

definition $twl\text{-}st\text{-}heur\text{-}parsing\text{-}no\text{-}WL\text{-}wl\text{-}no\text{-}watched :: \langle nat\ multiset \Rightarrow bool \Rightarrow (twl\text{-}st\text{-}wl\text{-}heur\text{-}init\text{-}full$
$\times nat\ twl\text{-}st\text{-}wl\text{-}init) set\rangle$ **where**
$\langle twl\text{-}st\text{-}heur\text{-}parsing\text{-}no\text{-}WL\text{-}wl\text{-}no\text{-}watched\ \mathcal{A}\ unbdd =$
$\{(S,$
$((M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, Q), OC)).$
$let\ M' = get\text{-}trail\text{-}init\text{-}wl\text{-}heur\ S;\ N' = get\text{-}clauses\text{-}wl\text{-}heur\text{-}init\ S;\ failed = is\text{-}failed\text{-}heur\text{-}init\ S;$
$vdom = get\text{-}vdom\text{-}heur\text{-}init\ S;\ ivdom = get\text{-}ivdom\text{-}heur\text{-}init\ S;\ D' = get\text{-}conflict\text{-}wl\text{-}heur\text{-}init\ S;\ vm$
$= get\text{-}vmtf\text{-}wl\text{-}heur\text{-}init\ S;$
$mark = get\text{-}mark\text{-}wl\text{-}heur\text{-}init\ S;\ lcount = get\text{-}learned\text{-}count\text{-}init\ S;\ W' = get\text{-}watchlist\text{-}wl\text{-}heur\text{-}init$

S;
φ = *get-phases-wl-heur-init* S; *cach* = *get-cach-wl-heur-init* S; *lbd* = *get-lbd-wl-heur-init* S;
j = *get-literals-to-update-wl-init* S
in
(*unbdd* $\longrightarrow$ $\neg$*failed*) $\wedge$
((*unbdd* $\vee$ $\neg$*failed*) $\longrightarrow$
(*valid-arena* N' N (*set vdom*) $\wedge$ *set-mset* (*dom-m* N) $\subseteq$ *set vdom*) $\wedge$ *ivdom* = *vdom*) $\wedge$ (M', M) $\in$ *trail-pol* $\mathcal{A}$ $\wedge$
(D', D) $\in$ *option-lookup-clause-rel* $\mathcal{A}$ $\wedge$
$j \leq$ *length* M $\wedge$
Q = *uminus* ‘# *lit-of* ‘# *mset* (*drop* j (*rev* M)) $\wedge$
vm $\in$ *bump-heur-init* $\mathcal{A}$ M $\wedge$
phase-saving $\mathcal{A}$ φ $\wedge$
no-dup M $\wedge$
cach-refinement-empty $\mathcal{A}$ *cach* $\wedge$
set-mset (*all-lits-of-mm* ({#*mset* (*fst x*). x ∈# *ran-m* N#} + (*NE*+*NEk*) + (*UE*+*UEk*) + *NS* + *US* + *N0* + *U0*))
$\subseteq$ *set-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$) $\wedge$
(W', *empty-watched* $\mathcal{A}$) $\in$ $\langle Id \rangle$*map-fun-rel* (D_0 $\mathcal{A}$) $\wedge$
isasat-input-bounded $\mathcal{A}$ $\wedge$
distinct vdom $\wedge$
clss-size-corr N NE UE NEk UEk NS US N0 U0 lcount $\wedge$
(*mark*, {#}) $\in$ *lookup-clause-rel* $\mathcal{A}$
}›

definition *twl-st-heur-post-parsing-wl* :: ‹*bool* $\Rightarrow$ (*twl-st-wl-heur-init-full* $\times$ *nat twl-st-wl*) *set*› **where**
‹*twl-st-heur-post-parsing-wl unbdd* =
{(S,
(M, N, D, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, Q, W)).
let M' = *get-trail-init-wl-heur* S; N' = *get-clauses-wl-heur-init* S; *failed* = *is-failed-heur-init* S;
vdom = *get-vdom-heur-init* S; *ivdom* = *get-ivdom-heur-init* S; D' = *get-conflict-wl-heur-init* S; *vm* = *get-vmtf-wl-heur-init* S;
mark = *get-mark-wl-heur-init* S; *lcount* = *get-learned-count-init* S; W' = *get-watchlist-wl-heur-init* S;
φ = *get-phases-wl-heur-init* S; *cach* = *get-cach-wl-heur-init* S; *lbd* = *get-lbd-wl-heur-init* S;
j = *get-literals-to-update-wl-init* S
in
(*unbdd* $\longrightarrow$ $\neg$*failed*) $\wedge$
((*unbdd* $\vee$ $\neg$*failed*) $\longrightarrow$
((M', M) $\in$ *trail-pol* (*all-atms* N ((*NE*+*NEk*) + (*UE*+*UEk*) + *NS* + *US* + *N0* + *U0*)) $\wedge$
set-mset (*dom-m* N) $\subseteq$ *set vdom* $\wedge$
valid-arena N' N (*set vdom*) $\wedge$ *ivdom*=*vdom*)) $\wedge$
(D', D) $\in$ *option-lookup-clause-rel* (*all-atms* N ((*NE*+*NEk*) + (*UE*+*UEk*) + *NS* + *US* + *N0* + *U0*)) $\wedge$
$j \leq$ *length* M $\wedge$
Q = *uminus* ‘# *lit-of* ‘# *mset* (*drop* j (*rev* M)) $\wedge$
vm $\in$ *bump-heur-init* (*all-atms* N ((*NE*+*NEk*) + (*UE*+*UEk*) + *NS* + *US* + *N0* + *U0*)) M $\wedge$
phase-saving (*all-atms* N ((*NE*+*NEk*) + (*UE*+*UEk*) + *NS* + *US* + *N0* + *U0*)) φ $\wedge$
no-dup M $\wedge$
cach-refinement-empty (*all-atms* N ((*NE*+*NEk*) + (*UE*+*UEk*) + *NS* + *US* + *N0* + *U0*)) *cach* $\wedge$
vdom-m (*all-atms* N ((*NE*+*NEk*) + (*UE*+*UEk*) + *NS* + *US* + *N0* + *U0*)) W N $\subseteq$ *set vdom* $\wedge$
set-mset (*all-lits-of-mm* ({#*mset* (*fst x*). x ∈# *ran-m* N#} + (*NE*+*NEk*) + (*UE*+*UEk*) + *NS* + *US* + *N0* + *U0*))
$\subseteq$ *set-mset* ($\mathcal{L}_{all}$ (*all-atms* N ((*NE*+*NEk*) + (*UE*+*UEk*) + *NS* + *US* + *N0* + *U0*))) $\wedge$
(W', W) $\in$ $\langle Id \rangle$*map-fun-rel* (D_0 (*all-atms* N ((*NE*+*NEk*) + (*UE*+*UEk*) + *NS* + *US* + *N0* + *U0*))) $\wedge$

isasat-input-bounded (*all-atms N* ((*NE*+*NEk*) + (*UE*+*UEk*) + *NS* + *US* + *N0* + *U0*)) ∧
distinct vdom ∧
clss-size-corr N NE UE NEk UEk NS US N0 U0 lcount ∧
(*mark*, {#}) ∈ *lookup-clause-rel* (*all-atms N* ((*NE*+*NEk*) + (*UE*+*UEk*) + *NS* + *US* + *N0* + *U0*))
}›

17.1.2 Parsing

definition *propagate-unit-cls*
:: ‹*nat literal* ⇒ *nat twl-st-wl-init* ⇒ *nat twl-st-wl-init›*
where
‹*propagate-unit-cls* = (λ*L* ((*M*, *N*, *D*, *NE*, *UE*, *Q*), *OC*).
((*Propagated L 0* # *M*, *N*, *D*, *add-mset* {#*L*#} *NE*, *UE*, *Q*), *OC*))›

definition *propagate-unit-cls-heur*
:: ‹*bool* ⇒ *nat literal* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres›*
where
‹*propagate-unit-cls-heur* = (λ*unbdd L S*.
do {
M ← *cons-trail-Propagated-tr L 0* (*get-trail-init-wl-heur S*);
RETURN (*set-trail-init-wl-heur M S*)
})›

fun *get-unit-clauses-init-wl* :: ‹'*v twl-st-wl-init* ⇒ '*v clauses›* **where**
‹*get-unit-clauses-init-wl* ((*M*, *N*, *D*, *NE*, *UE*, *Q*), *OC*) = *NE* + *UE*›

fun *get-subsumed-clauses-init-wl* :: ‹'*v twl-st-wl-init* ⇒ '*v clauses›* **where**
‹*get-subsumed-clauses-init-wl* ((*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *N0*, *U0*,*Q*), *OC*) = *NS* + *US*›

fun *get-subsumed-init-clauses-init-wl* :: ‹'*v twl-st-wl-init* ⇒ '*v clauses›* **where**
‹*get-subsumed-init-clauses-init-wl* ((*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *N0*, *U0*,*Q*), *OC*) = *NS*›

abbreviation *all-lits-st-init* :: ‹'*v twl-st-wl-init* ⇒ '*v literal multiset›* **where**
‹*all-lits-st-init S* ≡ *all-lits* (*get-clauses-init-wl S*)
(*get-unit-clauses-init-wl S* + *get-subsumed-init-clauses-init-wl S*)›

definition *all-atms-init* :: ‹- ⇒ - ⇒ '*v multiset›* **where**
‹*all-atms-init N NUE* = *atm-of* ‘# *all-lits N NUE›*

abbreviation *all-atms-st-init* :: ‹'*v twl-st-wl-init* ⇒ '*v multiset›* **where**
‹*all-atms-st-init S* ≡ *atm-of* ‘# *all-lits-st-init S›*

lemma *DECISION-REASON0*[*simp*]: ‹*DECISION-REASON* ≠ *0›*
by (*auto simp*: *DECISION-REASON-def*)

lemma *propagate-unit-cls-heur-propagate-unit-cls*:
‹(*uncurry* (*propagate-unit-cls-heur unbdd*), *uncurry* (*propagate-unit-init-wl*)) ∈
[λ(*L*, *S*). *undefined-lit* (*get-trail-init-wl S*) *L* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$
Id ×$_r$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* → ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel›*
unfolding *twl-st-heur-parsing-no-WL-def propagate-unit-cls-heur-def propagate-unit-init-wl-def nres-monad3*
apply (*intro frefI nres-relI*)
apply (*clarsimp simp add*: *propagate-unit-init-wl.simps cons-trail-Propagated-tr-def*[*symmetric*] *comp-def curry-def all-atms-def*[*symmetric*] *intro*!: *ASSERT-leI*)
apply (*refine-rcg cons-trail-Propagated-tr2*[**where** $\mathcal{A}$ = $\mathcal{A}$])

subgoal by *auto*
subgoal by *auto*
subgoal by (*auto intro*!: *bump-heur-init-consD'*
simp: *all-lits-of-mm-add-mset all-lits-of-m-add-mset uminus-$\mathcal{A}_{in}$-iff*)
done

definition *already-propagated-unit-cls*
:: ‹*nat literal* ⇒ *nat twl-st-wl-init* ⇒ *nat twl-st-wl-init*›
where
‹*already-propagated-unit-cls* = (λ*L* ((*M*, *N*, *D*, *NE*, *UE*, *Q*), *OC*).
((*M*, *N*, *D*, *add-mset* {#*L*#} *NE*, *UE*, *Q*), *OC*))›

definition *already-propagated-unit-cls-heur*
:: ‹*bool* ⇒ *nat clause-l* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*›
where
‹*already-propagated-unit-cls-heur* = (λ*unbdd L S*.
RETURN S)›

lemma *already-propagated-unit-cls-heur-already-propagated-unit-cls*:
‹(*uncurry* (*already-propagated-unit-cls-heur unbdd*), *uncurry* (*RETURN oo already-propagated-unit-init-wl*))
∈
[λ(*C*, *S*). *literals-are-in-$\mathcal{L}_{in}$* $\mathcal{A}$ *C*]$_f$
list-mset-rel ×$_r$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* → ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *twl-st-heur-parsing-no-WL-def already-propagated-unit-cls-heur-def*
already-propagated-unit-init-wl-def all-lits-of-mm-add-mset all-lits-of-m-add-mset
literals-are-in-$\mathcal{L}_{in}$-def)

definition (**in** −) *set-conflict-unit* :: ‹*nat literal* ⇒ *nat clause option* ⇒ *nat clause option*› **where**
‹*set-conflict-unit L* - = *Some* {#*L*#}›

definition *set-conflict-unit-heur* **where**
‹*set-conflict-unit-heur* = (λ *L* (*b*, *n*, *xs*). *RETURN* (*False*, *1*, *xs*[*atm-of L* := *Some* (*is-pos L*)]))›

lemma *set-conflict-unit-heur-set-conflict-unit*:
‹(*uncurry set-conflict-unit-heur*, *uncurry* (*RETURN oo set-conflict-unit*)) ∈
[λ(*L*, *D*). *D* = *None* ∧ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$]$_f$ *Id* ×$_f$ *option-lookup-clause-rel* $\mathcal{A}$ →
⟨*option-lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *twl-st-heur-def set-conflict-unit-heur-def set-conflict-unit-def*
option-lookup-clause-rel-def lookup-clause-rel-def in-$\mathcal{L}_{all}$-atm-of-in-atms-of-iff
intro!: *mset-as-position.intros*)

definition *conflict-propagated-unit-cls*
:: ‹*nat literal* ⇒ *nat twl-st-wl-init* ⇒ *nat twl-st-wl-init*›
where
‹*conflict-propagated-unit-cls* = (λ*L* ((*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*,*Q*), *OC*).
((*M*, *N*, *set-conflict-unit L D*, *add-mset* {#*L*#} *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*,{#}), *OC*))›

definition *conflict-propagated-unit-cls-heur*
:: ‹*bool* ⇒ *nat literal* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*›
where
‹*conflict-propagated-unit-cls-heur* = (λ*unbdd L S*.
do {
ASSERT(*atm-of L* < *length* (*snd* (*snd* (*get-conflict-wl-heur-init S*))));
D ← *set-conflict-unit-heur L* (*get-conflict-wl-heur-init S*);

```
    ASSERT(isa-length-trail-pre (get-trail-init-wl-heur S));
    let j = isa-length-trail (get-trail-init-wl-heur S);
    RETURN (set-literals-to-update-wl-init j (set-conflict-wl-heur-init D S))
  })›

definition conflict-propagated-unit-cls-heur-b :: ‹-› where
  ‹conflict-propagated-unit-cls-heur-b = conflict-propagated-unit-cls-heur False›

lemma conflict-propagated-unit-cls-heur-conflict-propagated-unit-cls:
  ‹(uncurry (conflict-propagated-unit-cls-heur unbdd), uncurry (RETURN oo set-conflict-init-wl)) ∈
   [λ(L, S). L ∈# 𝓛_all 𝒜 ∧ get-conflict-init-wl S = None]_f
      nat-lit-lit-rel ×_r twl-st-heur-parsing-no-WL 𝒜 unbdd → ⟨twl-st-heur-parsing-no-WL 𝒜 unbdd⟩
nres-rel›
proof −
  have set-conflict-init-wl-alt-def:
    ‹RETURN oo set-conflict-init-wl = (λL ((M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0,Q), OC).
do {
      D ← RETURN (set-conflict-unit L D);
      RETURN ((M, N, Some {#L#}, add-mset {#L#} NE, UE, NEk, UEk, NS, US, N0, U0,{#}),
OC)
    })›
    by (auto intro!: ext simp: set-conflict-init-wl-def)
  have [refine0]: ‹D = None ∧ L ∈# 𝓛_all 𝒜 ⟹ (y, None) ∈ option-lookup-clause-rel 𝒜 ⟹ L = L'
⟹
    set-conflict-unit-heur L' y ≤ ⇓ {(D, D'). (D, D') ∈ option-lookup-clause-rel 𝒜 ∧ D' = Some {#L#}}
       (RETURN (set-conflict-unit L D))›
    for L D y L'
    apply (rule order-trans)
    apply (rule set-conflict-unit-heur-set-conflict-unit[THEN fref-to-Down-curry,
      unfolded comp-def, of 𝒜 L D L' y])
    subgoal
      by auto
    subgoal
      by auto
    subgoal
      unfolding conc-fun-RETURN
      by (auto simp: set-conflict-unit-def)
    done
  show ?thesis
    supply RETURN-as-SPEC-refine[refine2 del]
    unfolding set-conflict-init-wl-alt-def conflict-propagated-unit-cls-heur-def uncurry-def
    apply (intro frefI nres-relI)
    apply (refine-rcg lhs-step-If)
    subgoal
      by (auto simp: twl-st-heur-parsing-no-WL-def option-lookup-clause-rel-def
        lookup-clause-rel-def atms-of-def)
    subgoal
      by auto
    subgoal
      by auto
    subgoal
      by (auto simp: twl-st-heur-parsing-no-WL-def conflict-propagated-unit-cls-heur-def conflict-propagated-unit-cls-def
        image-image set-conflict-unit-def
        intro!: set-conflict-unit-heur-set-conflict-unit[THEN fref-to-Down-curry])
    subgoal
      by auto
```

subgoal
by (*auto simp*: *twl-st-heur-parsing-no-WL-def conflict-propagated-unit-cls-heur-def*
conflict-propagated-unit-cls-def
intro!: *isa-length-trail-pre*)
subgoal
by (*auto simp*: *twl-st-heur-parsing-no-WL-def conflict-propagated-unit-cls-heur-def*
conflict-propagated-unit-cls-def
image-image set-conflict-unit-def all-lits-of-mm-add-mset all-lits-of-m-add-mset uminus-$\mathcal{A}_{in}$-iff
isa-length-trail-length-u[*THEN fref-to-Down-unRET-Id*]
intro!: *set-conflict-unit-heur-set-conflict-unit*[*THEN fref-to-Down-curry*]
isa-length-trail-pre)
done
qed

definition *add-init-cls-heur*
:: ‹*bool* ⇒ *nat clause-l* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*add-init-cls-heur unbdd* = (λ*C S*. *do* {
let C = *C*;
ASSERT(*length C* ≤ *unat32-max* + *2*);
ASSERT(*length C* ≥ *2*);
let N = *get-clauses-wl-heur-init S*;
let failed = *is-failed-heur-init S*;
if unbdd ∨ (*length N* ≤ *snat64-max* − *length C* − *5* ∧ ¬*failed*)
then do {
let vdom = *get-vdom-heur-init S*;
let ivdom = *get-ivdom-heur-init S*;
ASSERT(*length vdom* ≤ *length N* ∧ *vdom* = *ivdom*);
(*N*, *i*) ← *fm-add-new True C N*;
let vdom = *vdom* @ [*i*];
let ivdom = *ivdom* @ [*i*];
RETURN (*set-clauses-wl-heur-init N* (*set-vdom-heur-init vdom* (*set-ivdom-heur-init ivdom S*)))
} *else RETURN* (*set-failed-heur-init True S*)})›

definition *add-init-cls-heur-unb* :: ‹*nat clause-l* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*add-init-cls-heur-unb* = *add-init-cls-heur True*›

definition *add-init-cls-heur-b* :: ‹*nat clause-l* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*add-init-cls-heur-b* = *add-init-cls-heur False*›

definition *add-init-cls-heur-b′* :: ‹*nat literal list list* ⇒ *nat* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*add-init-cls-heur-b′ C i* = *add-init-cls-heur False* (*C*!*i*)›

lemma *length-C-nempty-iff*: ‹*length C* ≥ *2* ⟷ *C* ≠ [] ∧ *tl C* ≠ []›
by (*cases C*; *cases* ‹*tl C*›) *auto*

context
fixes *unbdd* :: *bool* **and** $\mathcal{A}$ :: ‹*nat multiset*› **and**
CT :: ‹*nat clause-l* × *twl-st-wl-heur-init*› **and**
CSOC :: ‹*nat clause-l* × *nat twl-st-wl-init*› **and**
SOC :: ‹*nat twl-st-wl-init*› **and**
C C′ :: ‹*nat clause-l*› **and**
S :: ‹*nat twl-st-wl-init′*› **and** *x1a* **and** *N* :: ‹*nat clauses-l*› **and**
D :: ‹*nat cconflict*› **and** *x2b* **and** *NE UE NS US N0 U0 NEk UEk* :: ‹*nat clauses*› **and**
M :: ‹(*nat*,*nat*) *ann-lits*› **and**

a b c d e1 e2 e3 f m p q r s t u v w x y ua ub z y' **and**
Q **and**
x2e :: ‹*nat lit-queue-wl*› **and** *OC* :: ‹*nat clauses*› **and**
T :: *twl-st-wl-heur-init* **and**
M' :: ‹*trail-pol*› **and** *N'* :: *arena* **and**
D' :: *conflict-option-rel* **and**
j' :: *nat* **and**
W' :: ‹-› **and**
vm :: ‹*bump-heuristics-option-fst-As*› **and**
clvls :: *nat* **and**
cach :: *conflict-min-cach-l* **and**
lbd :: *lbd* **and**
ivdom vdom :: *vdom* **and**
failed :: *bool* **and**
lcount :: *clss-size* **and**
φ :: *phase-saver* **and**
mark :: ‹*lookup-clause-rel*›
assumes
pre: ‹*case CSOC of*
(*C*, *S*) ⇒ *2* ≤ *length C* ∧ *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*) ∧ *distinct C*› **and**
xy: ‹(*CT*, *CSOC*) ∈ *Id* ×$_f$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*› **and**
st:
‹*CSOC* = (*C*, *SOC*)›
‹*SOC* = (*S*, *OC*)›
‹*S* = (*M*, *a*)›
‹*a* = (*N*, *b*)›
‹*b* = (*D*, *c*)›
‹*c* = (*NE*, *d*)›
‹*d* = (*UE*, *e1*)›
‹*e1* = (*NEk*, *e2*)›
‹*e2* = (*UEk*, *e3*)›
‹*e3* = (*NS*, *f*)›
‹*f* = (*US*, *ua*)›
‹*ua* = (*N0*, *ub*)›
‹*ub* = (*U0*, *Q*)›
‹*CT* = (*C'*, *T*)›
begin

lemma *add-init-pre1*: ‹*length C'* ≤ *unat32-max* + *2*›
using *pre clss-size-unat32-max*[*of* $\mathcal{A}$ ‹*mset C*›] *xy st*
by (*auto simp*: *twl-st-heur-parsing-no-WL-def*)

lemma *add-init-pre2*: ‹*2* ≤ *length C'*›
using *pre xy st* **by** (*auto simp*:)

private lemma
x1g-x1: ‹*C'* = *C*› **and**
‹(*get-trail-init-wl-heur T*, *M*) ∈ *trail-pol* $\mathcal{A}$› **and**
valid: ‹*valid-arena* (*get-clauses-wl-heur-init T*) *N* (*set* (*get-vdom-heur-init T*))› **and**
‹(*get-conflict-wl-heur-init T*, *D*) ∈ *option-lookup-clause-rel* $\mathcal{A}$› **and**
‹*get-literals-to-update-wl-init T* ≤ *length M*› **and**
Q: ‹*Q* = {#− *lit-of x*. *x* ∈# *mset* (*drop* (*get-literals-to-update-wl-init T*) (*rev M*))#}› **and**
‹*get-vmtf-wl-heur-init T* ∈ *bump-heur-init* $\mathcal{A}$ *M*› **and**
‹*phase-saving* $\mathcal{A}$ (*get-phases-wl-heur-init T*)› **and**
‹*no-dup M*› **and**
‹*cach-refinement-empty* $\mathcal{A}$ (*get-cach-wl-heur-init T*)› **and**

vdom: ‹*mset* (*get-vdom-heur-init T*) = *dom-m N*› **and**
ivdom: ‹(*get-ivdom-heur-init T*) = (*get-vdom-heur-init T*)› **and**
var-incl:
‹*set-mset* (*all-lits-of-mm* ({#*mset* (*fst x*). *x* ∈# *ran-m N*#} + (*NE*+*NEk*) + *NS* + (*UE*+*UEk*) + *US* + *N0* + *U0*))
⊆ *set-mset* (*$\mathcal{L}_{all}$ $\mathcal{A}$*)› **and**
watched: ‹(*get-watchlist-wl-heur-init T*, *empty-watched* $\mathcal{A}$) ∈ ⟨*Id*⟩*map-fun-rel* (*D_0 $\mathcal{A}$*)› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$› **and**
dcount: ‹*clss-size-corr N NE UE NEk UEk NS US N0 U0* (*get-learned-count-init T*)›
if ‹¬(*is-failed-heur-init T*) ∨ *unbdd*›
using *that xy* **unfolding** *st twl-st-heur-parsing-no-WL-def*
by (*auto simp*: *ac-simps*)

lemma *init-fm-add-new*:
‹¬*is-failed-heur-init T* ∨ *unbdd* ⟹ *fm-add-new True C'* (*get-clauses-wl-heur-init T*)
≤ ⇓ {((*arena*, *i*), (*N''*, *i'*)). *valid-arena arena N''* (*insert i* (*set* (*get-vdom-heur-init T*))) ∧ *i* = *i'* ∧
i ∉# *dom-m N* ∧ *i* = *length* (*get-clauses-wl-heur-init T*) + *header-size C* ∧
i ∉ *set* (*get-vdom-heur-init T*)}
(*SPEC*
(λ(*N'*, *ia*).
0 < *ia* ∧ *ia* ∉# *dom-m N* ∧ *N'* = *fmupd ia* (*C*, *True*) *N*))›
(**is** ‹- ⟹ - ≤ ⇓ *?qq* -›)
unfolding *x1g-x1*
apply (*rule order-trans*)
apply (*rule fm-add-new-append-clause*)
using *valid vdom pre xy valid-arena-in-vdom-le-arena*[*OF valid*] *arena-lifting*(*2*)[*OF valid*]
valid **unfolding** *st*
by (*fastforce simp*: *x1g-x1 vdom-m-def*
intro!: *RETURN-RES-refine valid-arena-append-clause*)

lemma *add-init-cls-final-rel*:
fixes *nN'j'* :: ‹*arena-el list* × *nat*› **and**
nNj :: ‹(*nat*, *nat literal list* × *bool*) *fmap* × *nat*› **and**
nN :: ‹-› **and**
k :: ‹*nat*› **and** *nN'* :: ‹*arena-el list*› **and**
k' :: ‹*nat*›
assumes
‹(*nN'j'*, *nNj*) ∈ {((*arena*, *i*), (*N''*, *i'*)). *valid-arena arena N''* (*insert i* (*set* (*get-vdom-heur-init T*))) ∧ *i* = *i'* ∧
i ∉# *dom-m N* ∧ *i* = *length* (*get-clauses-wl-heur-init T*) + *header-size C* ∧
i ∉ *set* (*get-vdom-heur-init T*)}› **and**
‹*nNj* ∈ *Collect* (λ(*N'*, *ia*).
0 < *ia* ∧ *ia* ∉# *dom-m N* ∧ *N'* = *fmupd ia* (*C*, *True*) *N*)› **and**
‹*nN'j'* = (*nN'*, *k'*)› **and**
‹*nNj* = (*nN*, *k*)›
shows ‹((*set-clauses-wl-heur-init nN'* (*set-vdom-heur-init* (*get-vdom-heur-init T* @ [*k'*]) (*set-ivdom-heur-init* (*get-ivdom-heur-init T* @ [*k'*]) *T*))),
(*M*, *nN*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*), *OC*)
∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*›
proof −
show *?thesis*
using *assms xy pre* **unfolding** *st*
apply (*auto simp*: *twl-st-heur-parsing-no-WL-def ac-simps*
intro!:)
apply (*auto simp*: *vdom-m-simps5 ran-m-mapsto-upd-notin all-lits-of-mm-add-mset*

literals-are-in-$\mathcal{L}_{in}$-def)
done
qed
end

lemma *learned-clss-l-fmupd-if-in*:
assumes ‹$C' \in\# \mathit{dom\text{-}m}\ \mathit{new}$›
shows
‹*learned-clss-l* (*fmupd* C' *D new*) = (*if* ¬*snd D then add-mset D else id*)(*if* ¬*irred new* C' *then* (*remove1-mset* (*the* (*fmlookup new* C')) (*learned-clss-l new*)) *else learned-clss-l new*)›
proof −
define *new'* **where** ‹*new'* = *fmdrop* C' *new*›
define *E* **where** ‹*E* = *the* (*fmlookup new* C')›
then have *new*: ‹*new* = *fmupd* C' *E new'*› **and** [*simp*]: ‹$C' \notin\#$ *dom-m new'*›
using *assms distinct-mset-dom*[*of new*]
by (*auto simp*: *fmdrop-fmupd new'-def intro*!: *fmlookup-inject*[*THEN iffD1*] *ext*
dest: *multi-member-split*)
show *?thesis*
unfolding *new*
by (*auto simp*: *learned-clss-l-mapsto-upd-irrel learned-clss-l-fmupd-if*
dest!: *multi-member-split*)
qed

lemma *clss-size-new-irredI*:
‹ *clss-size-corr x1c x1e x1f x1g x1h x2h x2i x2j x2k* (*au*, *av*, *aw*, *ax*, *be*) ⟹
clss-size-corr (*fmupd b* (*x1*, *True*) *x1c*) *x1e x1f x1g x1h x2h x2i x2j x2k* (*au*, *av*, *aw*, *ax*, *be*) ⟷
b ∉#*dom-m x1c* ∨ *irred x1c b*
›
apply (*cases* ‹*b* ∈#*dom-m x1c*›)
apply (*rule iffI*)
apply (*auto simp*: *clss-size-corr-def clss-size-def*
clss-size-def learned-clss-l-fmupd-if-in size-remove1-mset-If
learned-clss-l-mapsto-upd-notin-irrelev dest: *learned-clss-l-mapsto-upd learned-clss-l-update*
split: *if-splits*)
using *learned-clss-l-mapsto-upd learned-clss-l-update* **by** *fastforce*

lemma *add-init-cls-heur-add-init-cls*:
‹(*uncurry* (*add-init-cls-heur unbdd*), *uncurry* (*add-to-clauses-init-wl*)) ∈
[λ(*C*, *S*). *length C* ≥ *2* ∧ *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*) ∧ *distinct C*]$_f$
Id ×$_r$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* → ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel*›
proof −
have [*iff*]: ‹(∀ *b*. *b* = *0* ∨ *b* ∈# *dom-m x1c* ∨ *b* ∈# *dom-m x1c* ∧ ¬ *irred x1c b*) ⟷ *False*› **for** *x1c*
apply (*intro iffI impI*, *auto*)
apply (*drule spec*[*of* - ‹*Max-dom x1c* + *1*›])
by (*auto simp*: *ge-Max-dom-notin-dom-m*)
have [*iff*]: ‹∃ *b*>*0*. *b* ∉# *dom-m x1c* ∧ (*b* ∈# *dom-m x1c* ⟶ *irred x1c b*)› **for** *x1c x1*
by (*rule exI*[*of* - ‹*Max-dom x1c* + *1*›])
(*auto simp*: *ge-Max-dom-notin-dom-m*)

have ‹*42* + *Max-mset* (*add-mset 0* (*x1c*)) ∉# *x1c*› **and** ‹*42* + *Max-mset* (*add-mset* (*0* :: *nat*) (*x1c*)) ≠ *0*› **for** *x1c*
apply (*cases* ‹*x1c*›) **apply** (*auto simp*: *max-def*)
apply (*metis Max-ge add.commute add.right-neutral add-le-cancel-left finite-set-mset le-zero-eq set-mset-add-mset-inser union-single-eq-member zero-neq-numeral*)

by (*smt Max-ge Set.set-insert add.commute add.right-neutral add-mset-commute antisym diff-add-inverse diff-le-self finite-insert finite-set-mset insert-DiffM insert-commute set-mset-add-mset-insert union-single-eq-member zero-neq-numeral*)
then have [*iff*]: ‹(∀ *b*. *b* = (*0*::*nat*) ∨ *b* ∈# *x1c*) ⟷ *False*› ‹∃ *b*>*0*. *b* ∉# *x1c*›**for** *x1c*
by *blast+*
have *add-to-clauses-init-wl-alt-def*:
‹*add-to-clauses-init-wl* = (λ*i* ((*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*), *OC*). *do* {
let b = (*length i* = *2*);
(*N'*, *ia*) ← *SPEC* (λ(*N'*, *ia*). *ia* > *0* ∧ *ia* ∉# *dom-m N* ∧ *N'* = *fmupd ia* (*i*, *True*) *N*);
RETURN ((*M*, *N'*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*), *OC*)
})›
by (*auto simp*: *add-to-clauses-init-wl-def get-fresh-index-def Let-def RES-RES2-RETURN-RES RES-RES-RETURN-RES2 RES-RETURN-RES uncurry-def image-iff intro*!: *ext*)
show *?thesis*
unfolding *add-init-cls-heur-def add-to-clauses-init-wl-alt-def uncurry-def Let-def*
apply (*intro frefI nres-relI*)
apply (*refine-vcg init-fm-add-new*)
subgoal
by (*rule add-init-pre1*)
subgoal
by (*rule add-init-pre2*)
apply (*rule lhs-step-If*)
apply (*refine-rcg*)
subgoal unfolding *twl-st-heur-parsing-no-WL-def*
by (*force dest*!: *valid-arena-vdom-le*(*2*) *simp*: *distinct-card*)
subgoal by (*auto simp*: *twl-st-heur-parsing-no-WL-def*)
apply (*rule init-fm-add-new*)
apply *assumption+*
subgoal by *auto*
subgoal by (*rule add-init-cls-final-rel*)
subgoal for *x y x1 x2 x1a x1b x2a x1c x2b x1d x2c x1e x2d x1f x2e x1g x2f x1h x2g x2h x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m*
unfolding *RES-RES2-RETURN-RES RETURN-def*
apply *simp*
unfolding *RETURN-def* **apply** (*rule RES-refine*)
apply (*auto simp*: *twl-st-heur-parsing-no-WL-def clss-size-new-irredI RETURN-def intro*!: *RES-refine*)
apply (*meson* ‹⋀*x1c*. (∀ *b*. *b* = *0* ∨ *b* ∈# *x1c*) = *False*› *clss-size-corr-simp*(*4*))
apply (*metis* (*no-types*, *opaque-lifting*) ‹⋀*x1c*. ∃ *b*>*0*. *b* ∉# *x1c*› *clss-size-corr-simp*(*4*))
apply (*meson* ‹⋀*x1c*. (∀ *b*. *b* = *0* ∨ *b* ∈# *x1c*) = *False*› *clss-size-corr-simp*(*4*))
apply (*metis* (*no-types*, *opaque-lifting*) ‹⋀*x1c*. ∃ *b*>*0*. *b* ∉# *x1c*› *clss-size-corr-simp*(*4*))
done
done
qed

definition *already-propagated-unit-cls-conflict*
:: ‹*nat literal* ⇒ *nat twl-st-wl-init* ⇒ *nat twl-st-wl-init*›
where
‹*already-propagated-unit-cls-conflict* = (λ*L* ((*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*), *OC*).
((*M*, *N*, *D*, *add-mset* {#*L*#} *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, {#}), *OC*))›

definition *already-propagated-unit-cls-conflict-heur*
:: ‹*bool* ⇒ *nat literal* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*›
where
‹*already-propagated-unit-cls-conflict-heur* = (λ*unbdd L S*. *do* {

```
    ASSERT (isa-length-trail-pre (get-trail-init-wl-heur S));
    RETURN (set-literals-to-update-wl-init (isa-length-trail (get-trail-init-wl-heur S)) S)
  })›

lemma already-propagated-unit-cls-conflict-heur-already-propagated-unit-cls-conflict:
  ‹(uncurry (already-propagated-unit-cls-conflict-heur unbdd),
    uncurry (RETURN oo already-propagated-unit-cls-conflict)) ∈
   [λ(L, S). L ∈# \mathcal{L}_{all} \mathcal{A}]_f Id ×_r twl-st-heur-parsing-no-WL \mathcal{A} unbdd →
    ⟨twl-st-heur-parsing-no-WL \mathcal{A} unbdd⟩ nres-rel›
  by (intro frefI nres-relI)
   (auto simp: twl-st-heur-parsing-no-WL-def already-propagated-unit-cls-conflict-heur-def
     already-propagated-unit-cls-conflict-def all-lits-of-mm-add-mset
     all-lits-of-m-add-mset uminus-\mathcal{A}_{in}-iff isa-length-trail-length-u[THEN fref-to-Down-unRET-Id]
     intro: vmtf-consD
     intro!: ASSERT-leI isa-length-trail-pre)

definition (in −) set-conflict-empty :: ‹nat clause option ⇒ nat clause option› where
‹set-conflict-empty - = Some {#}›

definition (in −) lookup-set-conflict-empty :: ‹conflict-option-rel ⇒ conflict-option-rel› where
‹lookup-set-conflict-empty  = (λ(b, s) . (False, s))›

lemma lookup-set-conflict-empty-set-conflict-empty:
  ‹(RETURN o lookup-set-conflict-empty, RETURN o set-conflict-empty) ∈
    [λD. D = None]_f option-lookup-clause-rel \mathcal{A} → ⟨option-lookup-clause-rel \mathcal{A}⟩nres-rel›
  by (intro frefI nres-relI) (auto simp: set-conflict-empty-def
     lookup-set-conflict-empty-def option-lookup-clause-rel-def
     lookup-clause-rel-def)

definition set-empty-clause-as-conflict-heur
  :: ‹twl-st-wl-heur-init ⇒ twl-st-wl-heur-init nres› where
  ‹set-empty-clause-as-conflict-heur = (λS. do {
    let M = get-trail-init-wl-heur S;
    let (-, (n, xs)) = get-conflict-wl-heur-init S;
    ASSERT(isa-length-trail-pre M);
    let j = isa-length-trail M;
    RETURN (set-conflict-wl-heur-init ((False, (n, xs))) (set-literals-to-update-wl-init j S))
  })›

lemma set-empty-clause-as-conflict-heur-set-empty-clause-as-conflict:
  ‹(set-empty-clause-as-conflict-heur, RETURN o add-empty-conflict-init-wl) ∈
  [λS. get-conflict-init-wl S = None]_f
  twl-st-heur-parsing-no-WL \mathcal{A} unbdd → ⟨twl-st-heur-parsing-no-WL \mathcal{A} unbdd⟩ nres-rel›
  unfolding set-empty-clause-as-conflict-heur-def add-empty-conflict-init-wl-def
     twl-st-heur-parsing-no-WL-def Let-def
  by (intro frefI nres-relI)
   (auto simp: set-empty-clause-as-conflict-heur-def add-empty-conflict-init-wl-def
     twl-st-heur-parsing-no-WL-def set-conflict-empty-def option-lookup-clause-rel-def
    lookup-clause-rel-def isa-length-trail-length-u[THEN fref-to-Down-unRET-Id]
    all-lits-of-mm-add-mset all-atms-st-def clss-size-def
      intro!: isa-length-trail-pre ASSERT-leI)


definition (in −) add-clause-to-others-heur
  :: ‹nat clause-l ⇒ twl-st-wl-heur-init ⇒ twl-st-wl-heur-init nres› where
  ‹add-clause-to-others-heur = (λ - S. RETURN S)›
```

lemma *add-clause-to-others-heur-add-clause-to-others*:
‹(*uncurry add-clause-to-others-heur*, *uncurry* (*RETURN oo add-to-other-init*)) ∈
⟨*Id*⟩*list-rel* ×$_r$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* →$_f$ ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *add-clause-to-others-heur-def add-to-other-init.simps*
twl-st-heur-parsing-no-WL-def)

definition (**in** −) *add-tautology-to-clauses*
:: ‹*nat clause-l* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*add-tautology-to-clauses* = (λ - *S*. *RETURN S*)›

lemma *add-tautology-to-clauses-add-tautology-init-wl*:
‹(*uncurry add-tautology-to-clauses*, *uncurry* (*RETURN oo add-to-tautology-init-wl*)) ∈
[λ(*C*, -). *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*)]$_f$
⟨*Id*⟩*list-rel* ×$_r$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* →
⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩ *nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *add-to-other-init.simps all-lits-of-m-remdups-mset*
add-to-tautology-init-wl.simps add-tautology-to-clauses-def
twl-st-heur-parsing-no-WL-def all-lits-of-mm-add-mset
literals-are-in-$\mathcal{L}_{in}$*-def*)

definition (**in** −)*list-length-1* **where**
[*simp*]: ‹*list-length-1 C* ⟷ *length C* = *1*›

definition (**in** −)*list-length-1-code* **where**
‹*list-length-1-code C* ⟷ (*case C of* [-] ⇒ *True* | - ⇒ *False*)›

definition (**in** −) *get-conflict-wl-is-None-heur-init* :: ‹*twl-st-wl-heur-init* ⇒ *bool*› **where**
‹*get-conflict-wl-is-None-heur-init* = (λ*S*. *fst* (*get-conflict-wl-heur-init S*))›

definition *pre-simplify-clause-lookup-st*
:: ‹*nat clause-l* ⇒ *nat clause-l* ⇒ *twl-st-wl-heur-init* ⇒ (*bool* × - × *twl-st-wl-heur-init*) *nres*›
where
‹*pre-simplify-clause-lookup-st* = (λ*C E S*. *do* {
(*tauto*, *C*, *mark*) ← *pre-simplify-clause-lookup C E* (*get-mark-wl-heur-init S*);
RETURN (*tauto*, *C*, (*set-mark-wl-heur-init mark S*))
})›

definition *init-dt-step-wl-heur*
:: ‹*bool* ⇒ *nat clause-l* ⇒ *twl-st-wl-heur-init* × *nat clause-l* ⇒
(*twl-st-wl-heur-init* × *nat clause-l*) *nres*›
where
‹*init-dt-step-wl-heur unbdd* C_0 = (λ(*S*, *C*). *do* {
if get-conflict-wl-is-None-heur-init S
then do {
(*tauto*, *C*, *S*) ← *pre-simplify-clause-lookup-st* C_0 *C S*;
if tauto
then do {*T* ← *add-tautology-to-clauses* C_0 *S*; *RETURN* (*T*, *take 0 C*)}
else if is-Nil C
then do {*T* ← *set-empty-clause-as-conflict-heur S*; *RETURN* (*T*, *take 0 C*)}
else if list-length-1 C
then do {

```
      ASSERT (C ≠ []);
      let L = C ! 0;
      ASSERT(polarity-pol-pre (get-trail-init-wl-heur S) L);
      let val-L = polarity-pol (get-trail-init-wl-heur S) L;
      if val-L = None
     then do {T ← propagate-unit-cls-heur unbdd L S; RETURN (T, take 0 C)}
      else
        if val-L = Some True
      then do {T ← already-propagated-unit-cls-heur unbdd C S; RETURN (T, take 0 C)}
        else do {T ← conflict-propagated-unit-cls-heur unbdd L S; RETURN (T, take 0 C)}
     }
     else do {
      ASSERT(length C ≥ 2);
       T ← add-init-cls-heur unbdd C S;
       RETURN (T, take 0 C)
    }
   }
   else do {T ← add-clause-to-others-heur C0 S; RETURN (T, take 0 C)}
 })›

lemma init-dt-step-wl-heur-alt-def:
‹init-dt-step-wl-heur unbdd C0 = (λ(S,D). do {
  if get-conflict-wl-is-None-heur-init S
  then do {
    (tauto, C, S) ← pre-simplify-clause-lookup-st C0 D S;
    if tauto
   then do {T ← add-tautology-to-clauses C0 S; RETURN (T, take 0 C)}
    else do {
      C ← RETURN C;
      if is-Nil C
    then do {T ← set-empty-clause-as-conflict-heur S; RETURN (T, take 0 C)}
      else if list-length-1 C
    then do {
      ASSERT (C ≠ []);
      let L = C ! 0;
      ASSERT(polarity-pol-pre (get-trail-init-wl-heur S) L);
      let val-L = polarity-pol (get-trail-init-wl-heur S) L;
      if val-L = None
    then do {T ← propagate-unit-cls-heur unbdd L S; RETURN (T, take 0 C)}
      else
        if val-L = Some True
      then do {T ← already-propagated-unit-cls-heur unbdd C S; RETURN (T, take 0 C)}
        else do {T ← conflict-propagated-unit-cls-heur unbdd L S; RETURN (T, take 0 C)}
      }
   else do {
     ASSERT(length C ≥ 2);
       T ← add-init-cls-heur unbdd C S;
       RETURN (T, take 0 C)
    }
   } }
       else do {T ← add-clause-to-others-heur C0 S; RETURN (T, take 0 C)}
 })›
  unfolding nres-monad1 init-dt-step-wl-heur-def by auto

named-theorems twl-st-heur-parsing-no-WL
lemma [twl-st-heur-parsing-no-WL]:
```

assumes ‹$(S,\ T) \in$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*›
shows ‹(*get-trail-init-wl-heur* S, *get-trail-init-wl* T) $\in$ *trail-pol* $\mathcal{A}$›
using *assms*
by (*cases* S; *auto simp*: *twl-st-heur-parsing-no-WL-def*; *fail*)+

definition *get-conflict-wl-is-None-init* :: ‹*nat twl-st-wl-init* ⇒ *bool*› **where**
‹*get-conflict-wl-is-None-init* = (λ((M, N, D, NE, UE, Q), OC). *is-None* D)›

lemma *get-conflict-wl-is-None-init-alt-def*:
‹*get-conflict-wl-is-None-init* $S \longleftrightarrow$ *get-conflict-init-wl* S = *None*›
by (*cases* S) (*auto simp*: *get-conflict-wl-is-None-init-def split*: *option.splits*)

lemma *get-conflict-wl-is-None-heur-get-conflict-wl-is-None-init*:
‹(*RETURN o get-conflict-wl-is-None-heur-init*, *RETURN o get-conflict-wl-is-None-init*) $\in$
twl-st-heur-parsing-no-WL $\mathcal{A}$ *unbdd* $\rightarrow_f$ ⟨*Id*⟩*nres-rel*›
apply (*intro frefI nres-relI*)
apply (*rename-tac* x y, *case-tac* x, *case-tac* y)
by (*auto simp*: *twl-st-heur-parsing-no-WL-def get-conflict-wl-is-None-heur-init-def option-lookup-clause-rel-def*
get-conflict-wl-is-None-init-def split: *option.splits*)

definition (**in** −) *get-conflict-wl-is-None-init'* **where**
‹*get-conflict-wl-is-None-init'* = *get-conflict-wl-is-None*›

definition *pre-simplify-clause-lookup-st-rel* **where**
‹*pre-simplify-clause-lookup-st-rel* $\mathcal{A}$ *unbdd* C T = {((*tauto*, C', S'), *tauto'*).
tauto' = *tauto* ∧
(*tauto* $\longleftrightarrow$ *tautology* (*mset* C)) ∧
(¬*tauto* $\longrightarrow$ *remdups-mset* (*mset* C) = *mset* C') ∧
(S', T) $\in$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*}›

lemma *pre-simplify-clause-lookup-st-rel-alt-def*:
‹*pre-simplify-clause-lookup-st-rel* $\mathcal{A}$ *unbdd* C T = {((*tauto*, C', S'), *tauto'*).
tauto' = *tauto* ∧
(*tauto* $\longleftrightarrow$ *tautology* (*mset* C)) ∧
(¬*tauto* $\longrightarrow$ *remdups-mset* (*mset* C) = *mset* C' ∧ *distinct* C') ∧
(S', T) $\in$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*}›
by (*auto simp*: *pre-simplify-clause-lookup-st-rel-def eq-commute*[*of* - ‹*mset* -›]
simp del: *distinct-mset-mset-distinct*
simp flip: *distinct-mset-mset-distinct*)

lemma *pre-simplify-clause-lookup-st-remdups-clause*:
fixes D :: ‹*nat clause-l*›
assumes ‹(C, C') $\in$ *Id*› ‹(S, T) $\in$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*›
‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* C)› ‹*isasat-input-bounded* $\mathcal{A}$› **and** [*simp*]: ‹D = []›
shows
‹*pre-simplify-clause-lookup-st* C D S ≤ ⇓ (*pre-simplify-clause-lookup-st-rel* $\mathcal{A}$ *unbdd* C T)
(*RETURN* (*tautology* (*mset* C')))›
proof −
have ‹*mset* (*remdups* C') = *remdups-mset* (*mset* C')›
by *auto*
then have [*iff*]: ‹(∀ x. *mset* x ≠ *remdups-mset* (*mset* C')) $\longleftrightarrow$ *False*›
by *blast*
show *?thesis*
using *assms*

unfolding *pre-simplify-clause-lookup-st-def remdups-clause-def conc-fun-RES*
RETURN-def
apply (*cases S*)
apply *clarify*
apply (*subst bind-rule-complete-RES*)
apply (*refine-vcg bind-rule-complete-RES*
pre-simplify-clause-lookup-pre-simplify-clause[*of - $\mathcal{A}$, THEN order-trans*])
subgoal by (*auto simp*: *twl-st-heur-parsing-no-WL-def*)
subgoal by (*auto simp*: *literals-are-in-$\mathcal{L}_{in}$-alt-def atms-of-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$*)
subgoal by (*auto simp*: *twl-st-heur-parsing-no-WL-def*)
subgoal
by (*rule order-trans*, *rule ref-two-step'*, *rule pre-simplify-clause-spec*)
(*auto simp*: *pre-simplify-clause-spec-def conc-fun-RES*
twl-st-heur-parsing-no-WL-def pre-simplify-clause-lookup-st-rel-def)
done
qed

definition *RETURN2* **where** ‹*RETURN2 = RETURN*›

lemma *init-dt-step-wl-alt-def*:
‹*init-dt-step-wl C S =*
(*case get-conflict-init-wl S of*
None ⇒
let B = tautology (*mset C*) *in*
if B
then do {*T* ← *RETURN* (*add-to-tautology-init-wl C S*); *RETURN T*}
else
do {
C ← *remdups-clause C*;
if length C = 0
then do {*T* ← *RETURN* (*add-empty-conflict-init-wl S*); *RETURN T*}
else if length C = 1
then
let L = hd C in
if undefined-lit (*get-trail-init-wl S*) *L*
then do {*T* ← *propagate-unit-init-wl L S*; *RETURN T*}
else if L ∈ *lits-of-l* (*get-trail-init-wl S*)
then do {*T* ← *RETURN* (*already-propagated-unit-init-wl* (*mset C*) *S*); *RETURN T*}
else do {*T* ← *RETURN* (*set-conflict-init-wl L S*); *RETURN T*}
else do {*T* ← *add-to-clauses-init-wl C S*; *RETURN T*}
}
| *Some D* ⇒
do {*T* ← *RETURN* (*add-to-other-init C S*); *RETURN T*})›
unfolding *init-dt-step-wl-def Let-def nres-monad1 while.imonad2 RETURN2-def* **by** *auto*

lemma *init-dt-step-wl-heur-init-dt-step-wl*:
‹(*uncurry* (*init-dt-step-wl-heur unbdd*), *uncurry init-dt-step-wl*) ∈
[λ(*C*, *S*). *literals-are-in-$\mathcal{L}_{in}$ $\mathcal{A}$* (*mset C*)]$_f$
Id ×$_f$ {((*S*, *C*), *T*). *C* = [] ∧ (*S*,*T*)∈ *twl-st-heur-parsing-no-WL $\mathcal{A}$ unbdd*} →
⟨{((*S*, *C*), *T*). *C* = [] ∧ (*S*,*T*)∈*twl-st-heur-parsing-no-WL $\mathcal{A}$ unbdd*}⟩ *nres-rel*›
proof −
have *remdups*: ‹
(*x'*, *tautology* (*mset x1*))
∈ *pre-simplify-clause-lookup-st-rel $\mathcal{A}$ unbdd x1a T* ⟹
x2b = (*x1c*, *x2c*) ⟹

$x' = (x1b,\ x2b) \Longrightarrow$
$\neg\ x1b \Longrightarrow$
$\neg\ tautology\ (mset\ x1) \Longrightarrow (x1a,\ x1) \in Id \Longrightarrow$
$RETURN\ x1c \leq \Downarrow \{(a,b).\ a = b \wedge set\ b \subseteq set\ x1 \wedge x1c = b\}\ (remdups\text{-}clause\ x1)\rangle$
for *x1 x1a x1c x2 x2b x2c x' x1b T*
apply (*auto simp*: *remdups-clause-def pre-simplify-clause-lookup-st-rel-def*
intro!: *RETURN-RES-refine*)
by (*metis set-mset-mset set-mset-remdups-mset*)
show *?thesis*
supply [[*goals-limit=1*]]
unfolding *init-dt-step-wl-alt-def uncurry-def*
option.case-eq-if get-conflict-wl-is-None-init-alt-def[*symmetric*]
apply (*subst init-dt-step-wl-heur-alt-def*)
supply *RETURN-as-SPEC-refine*[*refine2 del*]
apply (*intro frefI nres-relI*)
apply (*refine-vcg*
set-empty-clause-as-conflict-heur-set-empty-clause-as-conflict[**where** $\mathcal{A} = \mathcal{A}$ **and** *unbdd* = *unbdd*,
THEN fref-to-Down, *unfolded comp-def*]
propagate-unit-cls-heur-propagate-unit-cls[**where** $\mathcal{A} = \mathcal{A}$ **and** *unbdd* = *unbdd*, *THEN fref-to-Down-curry*,
unfolded comp-def]
already-propagated-unit-cls-heur-already-propagated-unit-cls[**where** $\mathcal{A} = \mathcal{A}$ **and** *unbdd* = *unbdd*, *THEN*
fref-to-Down-curry,
unfolded comp-def]
conflict-propagated-unit-cls-heur-conflict-propagated-unit-cls[**where** $\mathcal{A} = \mathcal{A}$ **and** *unbdd* = *unbdd*, *THEN*
fref-to-Down-curry,
unfolded comp-def]
add-init-cls-heur-add-init-cls[**where** $\mathcal{A} = \mathcal{A}$ **and** *unbdd* = *unbdd*, *THEN fref-to-Down-curry*,
unfolded comp-def]
add-clause-to-others-heur-add-clause-to-others[**where** $\mathcal{A} = \mathcal{A}$ **and** *unbdd* = *unbdd*, *THEN fref-to-Down-curry*,
unfolded comp-def]
pre-simplify-clause-lookup-st-remdups-clause[**where** $\mathcal{A} = \mathcal{A}$ **and** *unbdd* = *unbdd*]
add-tautology-to-clauses-add-tautology-init-wl[**where** $\mathcal{A} = \mathcal{A}$ **and** *unbdd* = *unbdd*,
THEN fref-to-Down-curry, *unfolded comp-def*]
remdups)
subgoal by (*auto simp*: *get-conflict-wl-is-None-heur-get-conflict-wl-is-None-init*[*THEN fref-to-Down-unRET-Id*])
subgoal by (*auto simp*: *twl-st-heur-parsing-no-WL-def is-Nil-def split*: *list.splits*)
apply *auto*[]
subgoal by *simp*
subgoal by (*clarsimp simp add*: *twl-st-heur-parsing-no-WL-def*)
subgoal by *simp*
subgoal by (*simp only*: *prod.case in-pair-collect-simp pre-simplify-clause-lookup-st-rel-def*)
subgoal by *simp*
subgoal by (*auto simp*: *pre-simplify-clause-lookup-st-rel-def*)
subgoal by *auto*
apply *assumption*
apply *assumption*
apply *assumption*
apply *assumption*
subgoal by *auto*
subgoal by (*auto split*: *list.splits*)
subgoal by (*simp add*: *get-conflict-wl-is-None-init-alt-def*)
subgoal by (*auto simp*: *pre-simplify-clause-lookup-st-rel-def*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal for *x y x1 T x1a x2a x1b x2b x' x1c x1d x2d S'' D Ca*

by (*rule polarity-pol-pre*[*of* ‹*get-trail-init-wl-heur S''*› ‹*get-trail-init-wl T*› *A* ‹*D*!*0*›])
(*auto simp*: *pre-simplify-clause-lookup-st-rel-def twl-st-heur-parsing-no-WL*
literals-are-in-$\mathcal{L}_{in}$-add-mset dest!: *split-list*
split: *list.splits*)
subgoal for *x y x1 T x1a x2a x1b x2b x' x1c x1d x2d S'' D Ca*
by (*subst polarity-pol-polarity*[*of* $\mathcal{A}$, *unfolded option-rel-id-simp*,
THEN fref-to-Down-unRET-uncurry-Id, *of* ‹*get-trail-init-wl T*› ‹*hd D*›])
(*auto simp*: *pre-simplify-clause-lookup-st-rel-def twl-st-heur-parsing-no-WL*
polarity-pol-polarity[*of* $\mathcal{A}$, *unfolded option-rel-id-simp*, *THEN fref-to-Down-unRET-uncurry-Id*]
polarity-def
literals-are-in-$\mathcal{L}_{in}$-add-mset dest: *split-list split*: *list.splits*)
subgoal by *auto*
subgoal by (*auto simp*: *pre-simplify-clause-lookup-st-rel-def twl-st-heur-parsing-no-WL*
polarity-pol-polarity[*of* $\mathcal{A}$, *unfolded option-rel-id-simp*, *THEN fref-to-Down-unRET-uncurry-Id*]
polarity-def
literals-are-in-$\mathcal{L}_{in}$-add-mset dest: *split-list split*: *list.splits*)
subgoal by (*auto split*: *list.splits simp*: *pre-simplify-clause-lookup-st-rel-def*)
subgoal by *auto*
subgoal for *x y x1 T x1a x2a x1b x2b x' x1c x1d x2d S'' D Ca*
by (*subst polarity-pol-polarity*[*of* $\mathcal{A}$, *unfolded option-rel-id-simp*,
THEN fref-to-Down-unRET-uncurry-Id, *of* ‹*get-trail-init-wl T*› ‹*hd D*›])
(*auto simp*: *pre-simplify-clause-lookup-st-rel-def twl-st-heur-parsing-no-WL*
polarity-pol-polarity[*of* $\mathcal{A}$, *unfolded option-rel-id-simp*, *THEN fref-to-Down-unRET-uncurry-Id*]
polarity-def
literals-are-in-$\mathcal{L}_{in}$-add-mset dest: *split-list split*: *list.splits*)
subgoal by (*fastforce simp*: *literals-are-in-$\mathcal{L}_{in}$-alt-def*)
subgoal by (*auto simp*: *pre-simplify-clause-lookup-st-rel-def list-mset-rel-def br-def*)
subgoal by *auto*
subgoal by (*auto simp*: *literals-are-in-$\mathcal{L}_{in}$-alt-def*
in-$\mathcal{L}_{all}$-atm-of-in-atms-of-iff
split: *list.splits*)
subgoal by (*simp add*: *get-conflict-wl-is-None-init-alt-def*)
subgoal by (*simp add*: *hd-conv-nth pre-simplify-clause-lookup-st-rel-def*)
subgoal
by (*auto split*: *list.splits*)
subgoal
by (*auto split*: *list.splits*)
subgoal by *auto*
subgoal by (*fastforce simp*: *literals-are-in-$\mathcal{L}_{in}$-alt-def*)
subgoal by (*auto simp*: *pre-simplify-clause-lookup-st-rel-alt-def*)
subgoal by (*simp add*: *pre-simplify-clause-lookup-st-rel-def*)
subgoal by (*simp add*: *pre-simplify-clause-lookup-st-rel-def*)
subgoal by *fast*
subgoal by (*simp add*: *pre-simplify-clause-lookup-st-rel-def*)
subgoal by *auto*
done
qed

definition *polarity-st-init* :: ‹*'v twl-st-wl-init* ⇒ *'v literal* ⇒ *bool option*› **where**
‹*polarity-st-init S* = *polarity* (*get-trail-init-wl S*)›

lemma *get-conflict-wl-is-None-init*:
‹*get-conflict-init-wl S* = *None* ⟷ *get-conflict-wl-is-None-init S*›
by (*cases S*) (*auto simp*: *get-conflict-wl-is-None-init-def split*: *option.splits*)

definition *init-dt-wl-heur*

:: ‹*bool* ⇒ *nat clause-l list* ⇒ *twl-st-wl-heur-init* × - ⇒ (*twl-st-wl-heur-init* × -) *nres*›
where
‹*init-dt-wl-heur unbdd CS S* = *nfoldli CS* (λ-. *True*)
(λ*C S*. *do* {
init-dt-step-wl-heur unbdd C S}) *S*›

definition *init-dt-step-wl-heur-unb* :: ‹*nat clause-l* ⇒ *twl-st-wl-heur-init* × *nat clause-l* ⇒ (*twl-st-wl-heur-init* × *nat clause-l*) *nres*› **where**
‹*init-dt-step-wl-heur-unb* = *init-dt-step-wl-heur True*›

definition *init-dt-wl-heur-unb* :: ‹*nat clause-l list* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*›
where
‹*init-dt-wl-heur-unb CS S* = *do* { (*S*, -) ← *init-dt-wl-heur True CS* (*S*, []); *RETURN S*}›

definition *propagate-unit-cls-heur-b* :: ‹*nat literal* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*›
where
‹*propagate-unit-cls-heur-b* = *propagate-unit-cls-heur False*›

definition *already-propagated-unit-cls-conflict-heur-b* :: ‹*nat literal* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*already-propagated-unit-cls-conflict-heur-b* = *already-propagated-unit-cls-conflict-heur False*›

definition *init-dt-step-wl-heur-b* :: ‹*nat clause-l* ⇒ *twl-st-wl-heur-init* × *nat clause-l* ⇒
(*twl-st-wl-heur-init* × *nat clause-l*) *nres*› **where**
‹*init-dt-step-wl-heur-b* = *init-dt-step-wl-heur False*›

definition *init-dt-wl-heur-b* :: ‹*nat clause-l list* ⇒ *twl-st-wl-heur-init* ⇒
(*twl-st-wl-heur-init*) *nres*›
where
‹*init-dt-wl-heur-b CS S* = *do* { (*S*, -) ← *init-dt-wl-heur False CS* (*S*, []); *RETURN S*}›

17.1.3 Extractions of the atoms in the state

definition *init-valid-rep* :: ‹*nat list* ⇒ *nat set* ⇒ *bool*› **where**
‹*init-valid-rep xs l* ⟷
(∀ *L*∈*l*. *L* < *length xs*) ∧
(∀ *L* ∈ *l*. (*xs* ! *L*) *mod* 2 = 1) ∧
(∀ *L*. *L* < *length xs* ⟶ (*xs* ! *L*) *mod* 2 = 1 ⟶ *L* ∈ *l*)›

definition *isasat-atms-ext-rel* :: ‹((*nat list* × *nat* × *nat list*) × *nat set*) *set*› **where**
‹*isasat-atms-ext-rel* = {((*xs*, *n*, *atms*), *l*).
init-valid-rep xs l ∧
n = *Max* (*insert 0 l*) ∧
length xs < *unat32-max* ∧
(∀ *s*∈*set xs*. *s* ≤ *unat64-max*) ∧
finite l ∧
distinct atms ∧
set atms = *l* ∧
length xs ≠ *0*
}›

lemma *distinct-length-le-Suc-Max*:
assumes ‹*distinct* (*b* :: *nat list*)›
shows ‹*length b* ≤ *Suc* (*Max* (*insert 0* (*set b*)))›
proof −

```
  have ‹set b ⊆ {0 ..< Suc (Max (insert 0 (set b)))}›
    by (cases ‹set b = {}›)
     (auto simp add: le-imp-less-Suc)
  from card-mono[OF - this] show ?thesis
     using distinct-card[OF assms(1)] by auto
qed

lemma isasat-atms-ext-rel-alt-def:
  ‹isasat-atms-ext-rel = {((xs, n, atms), l).
      init-valid-rep xs l ∧
      n = Max (insert 0 l) ∧
      length xs < unat32-max ∧
      (∀ s∈set xs. s ≤ unat64-max) ∧
      finite l ∧
      distinct atms ∧
      set atms = l ∧
      length xs ≠ 0 ∧
      length atms ≤ Suc n
   }›
  by (auto simp: isasat-atms-ext-rel-def distinct-length-le-Suc-Max)


definition in-map-atm-of :: ‹'a ⇒ 'a list ⇒ bool› where
  ‹in-map-atm-of L N ⟷ L ∈ set N›

definition (in −) init-next-size where
  ‹init-next-size L = 2 * L›

lemma init-next-size: ‹L ≠ 0 ⟹ L + 1 ≤ unat32-max ⟹ L < init-next-size L›
  by (auto simp: init-next-size-def unat32-max-def)

definition add-to-atms-ext where
  ‹add-to-atms-ext = (λi (xs, n, atms). do {
    ASSERT(i ≤ unat32-max div 2);
    ASSERT(length xs ≤ unat32-max);
    ASSERT(length atms ≤ Suc n);
    let n = max i n;
    (if i < length-uint32-nat xs then do {
       ASSERT(xs!i ≤ unat64-max);
       let atms = (if xs!i AND 1 = 1 then atms else atms @ [i]);
       RETURN (xs[i := 1], n, atms)
     }
     else do {
       ASSERT(i + 1 ≤ unat32-max);
       ASSERT(length-uint32-nat xs ≠ 0);
       ASSERT(i < init-next-size i);
       RETURN ((list-grow xs (init-next-size i) 0)[i := 1], n,
           atms @ [i])
     })
    })›

lemma init-valid-rep-upd-OR:
  ‹init-valid-rep (x1b[x1a := a OR 1]) x2 ⟷
    init-valid-rep (x1b[x1a := 1]) x2 › (is ‹?A ⟷ ?B›)
proof
  assume ?A
```

```
  then have
    1: ‹∀ L∈x2. L < length (x1b[x1a := a OR 1])› and
    2: ‹∀ L∈x2. x1b[x1a := a OR 1] ! L mod 2 = 1› and
    3: ‹∀ L<length (x1b[x1a := a OR 1]).
        x1b[x1a := a OR 1] ! L mod 2 = 1 ⟶
        L ∈ x2›
    unfolding init-valid-rep-def by fast+
  have 1: ‹∀ L∈x2. L < length (x1b[x1a := 1])›
    using 1 by simp
  then have 2: ‹∀ L∈x2. x1b[x1a := 1] ! L mod 2 = 1›
    using 2 by (auto simp: nth-list-update')
  then have 3: ‹∀ L<length (x1b[x1a := 1]).
        x1b[x1a := 1] ! L mod 2 = 1 ⟶
        L ∈ x2›
    using 3 by (auto split: if-splits simp: bitOR-1-if-mod-2-nat)
  show ?B
    using 1 2 3
    unfolding init-valid-rep-def by fast+
next
  assume ?B
  then have
    1: ‹∀ L∈x2. L < length (x1b[x1a := 1])› and
    2: ‹∀ L∈x2. x1b[x1a := 1] ! L mod 2 = 1› and
    3: ‹∀ L<length (x1b[x1a := 1]).
        x1b[x1a := 1] ! L mod 2 = 1 ⟶
        L ∈ x2›
    unfolding init-valid-rep-def by fast+
  have 1: ‹∀ L∈x2. L < length (x1b[x1a :=  a OR 1])›
    using 1 by simp
  then have 2: ‹∀ L∈x2. x1b[x1a := a OR 1] ! L mod 2 = 1›
    using 2 by (auto simp: nth-list-update' bitOR-1-if-mod-2-nat)
  then have 3: ‹∀ L<length (x1b[x1a :=  a OR 1]).
        x1b[x1a :=  a OR 1] ! L mod 2 = 1 ⟶
        L ∈ x2›
    using 3 by (auto split: if-splits simp: bitOR-1-if-mod-2-nat)
  show ?A
    using 1 2 3
    unfolding init-valid-rep-def by fast+
qed

lemma init-valid-rep-insert:
  assumes val: ‹init-valid-rep x1b x2› and le: ‹x1a < length x1b›
  shows ‹init-valid-rep (x1b[x1a := Suc 0]) (insert x1a x2)›
proof −
  have
    1: ‹∀ L∈x2. L < length x1b› and
    2: ‹∀ L∈x2. x1b ! L mod 2 = 1› and
    3: ‹⋀L. L<length x1b ⟹ x1b ! L mod 2 = 1 ⟶ L ∈ x2›
    using val unfolding init-valid-rep-def by fast+
  have 1: ‹∀ L∈insert x1a x2. L < length (x1b[x1a := 1])›
    using 1 le by simp
  then have 2: ‹∀ L∈insert x1a x2. x1b[x1a := 1] ! L mod 2 = 1›
    using 2 by (auto simp: nth-list-update')
  then have 3: ‹∀ L<length (x1b[x1a := 1]).
        x1b[x1a := 1] ! L mod 2 = 1 ⟶
        L ∈ insert x1a x2›
```

```
    using 3 le by (auto split: if-splits simp: bitOR-1-if-mod-2-nat)
  show ?thesis
    using 1 2 3
    unfolding init-valid-rep-def by auto
qed

lemma init-valid-rep-extend:
  ‹init-valid-rep (x1b @ replicate n 0) x2 ⟷ init-valid-rep (x1b) x2›
   (is ‹?A ⟷ ?B› is ‹init-valid-rep ?x1b - ⟷ -›)
proof
  assume ?A
  then have
    1: ‹⋀L. L∈x2 ⟹ L < length ?x1b› and
    2: ‹⋀L. L∈x2 ⟹ ?x1b ! L mod 2 = 1› and
    3: ‹⋀L. L<length ?x1b ⟹ ?x1b ! L mod 2 = 1 ⟶ L ∈ x2›
    unfolding init-valid-rep-def by fast+
  have 1: ‹L∈x2 ⟹ L < length x1b› for L
    using 3[of L] 2[of L] 1[of L]
    by (auto simp: nth-append split: if-splits)
  then have 2: ‹∀ L∈x2. x1b ! L mod 2 = 1›
    using 2 by (auto simp: nth-list-update')
  then have 3: ‹∀ L<length x1b. x1b ! L mod 2 = 1 ⟶ L ∈ x2›
    using 3 by (auto split: if-splits simp: bitOR-1-if-mod-2-nat)
  show ?B
    using 1 2 3
    unfolding init-valid-rep-def by fast
next
  assume ?B
  then have
    1: ‹⋀L. L∈x2 ⟹ L < length x1b› and
    2: ‹⋀L. L∈x2 ⟹ x1b ! L mod 2 = 1› and
    3: ‹⋀L. L<length x1b ⟶ x1b ! L mod 2 = 1 ⟶ L ∈ x2›
    unfolding init-valid-rep-def by fast+
  have 10: ‹∀ L∈x2. L < length ?x1b›
    using 1 by fastforce
  then have 20: ‹L∈x2 ⟹ ?x1b ! L mod 2 = 1› for L
    using 1[of L] 2[of L] 3[of L] by (auto simp: nth-list-update' bitOR-1-if-mod-2-nat nth-append)
  then have 30: ‹L<length ?x1b ⟹ ?x1b ! L mod 2 = 1 ⟶ L ∈ x2›for L
    using 1[of L] 2[of L] 3[of L]
    by (auto split: if-splits simp: bitOR-1-if-mod-2-nat nth-append)
  show ?A
    using 10 20 30
    unfolding init-valid-rep-def by fast+
qed

lemma init-valid-rep-in-set-iff:
  ‹init-valid-rep x1b x2 ⟹ x ∈ x2 ⟷ (x < length x1b ∧ (x1b!x) mod 2 = 1)›
  unfolding init-valid-rep-def
  by auto

lemma add-to-atms-ext-op-set-insert:
  ‹(uncurry add-to-atms-ext, uncurry (RETURN oo Set.insert))
   ∈ [λ(n, l). n ≤ unat32-max div 2]f nat-rel ×f isasat-atms-ext-rel → ⟨isasat-atms-ext-rel⟩nres-rel›
proof −
  have H: ‹finite x2 ⟹ Max (insert x1 (insert 0 x2)) = Max (insert x1 x2)›
    ‹finite x2 ⟹ Max (insert 0 (insert x1 x2)) = Max (insert x1 x2)›
```

```
  for x1 and x2 :: ‹nat set›
  by (subst insert-commute) auto
have [simp]: ‹(a OR Suc 0) mod 2 = Suc 0› for a
  by (auto simp add: bitOR-1-if-mod-2-nat)
show ?thesis
  apply (intro frefI nres-relI)
  unfolding isasat-atms-ext-rel-def add-to-atms-ext-def uncurry-def
  apply (refine-vcg lhs-step-If)
  subgoal by auto
  subgoal by auto
  subgoal unfolding isasat-atms-ext-rel-def[symmetric] isasat-atms-ext-rel-alt-def by auto
  subgoal by auto
  subgoal for x y x1 x2 x1a x2a x1b x2b
    unfolding comp-def
    apply (rule RETURN-refine)
    apply (subst in-pair-collect-simp)
    apply (subst prod.case)+
    apply (intro conjI impI allI)
    subgoal by (simp add: init-valid-rep-upd-OR init-valid-rep-insert
        del: )
    subgoal by (auto simp: H Max-insert[symmetric] simp del: Max-insert)
    subgoal by auto
    subgoal
      unfolding bitOR-1-if-mod-2-nat
      by (auto simp del: simp: unat64-max-def
          elim!: in-set-upd-cases)
    subgoal
      unfolding bitAND-1-mod-2
      by (auto simp add: init-valid-rep-in-set-iff)
    subgoal
      unfolding bitAND-1-mod-2
      by (auto simp add: init-valid-rep-in-set-iff)
    subgoal
      unfolding bitAND-1-mod-2
      by (auto simp add: init-valid-rep-in-set-iff)
    subgoal
      by (auto simp add: init-valid-rep-in-set-iff)
    done
  subgoal by (auto simp: unat32-max-def)
  subgoal by (auto simp: unat32-max-def)
  subgoal by (auto simp: unat32-max-def init-next-size-def elim: neq-NilE)
  subgoal
    unfolding comp-def list-grow-def
    apply (rule RETURN-refine)
    apply (subst in-pair-collect-simp)
    apply (subst prod.case)+
    apply (intro conjI impI allI)
    subgoal
      unfolding init-next-size-def
      apply (simp del: )
      apply (subst init-valid-rep-insert)
      apply (auto elim: neq-NilE)
      apply (subst init-valid-rep-extend)
      apply (auto elim: neq-NilE)
      done
    subgoal by (auto simp: H Max-insert[symmetric] simp del: Max-insert)
```

```
    subgoal by (auto simp: init-next-size-def unat32-max-def)
    subgoal
      unfolding bitOR-1-if-mod-2-nat
      by (auto simp: unat64-max-def
          elim!: in-set-upd-cases)
    subgoal by (auto simp: init-valid-rep-in-set-iff)
    subgoal by (auto simp add: init-valid-rep-in-set-iff)
    subgoal by (auto simp add: init-valid-rep-in-set-iff)
    subgoal by (auto simp add: init-valid-rep-in-set-iff)
    done
  done
qed

definition extract-atms-cls :: ‹'a clause-l ⇒ 'a set ⇒ 'a set› where
  ‹extract-atms-cls C 𝒜in = fold (λL 𝒜in. insert (atm-of L) 𝒜in) C 𝒜in›

definition extract-atms-cls-i :: ‹nat clause-l ⇒ nat set ⇒ nat set nres› where
  ‹extract-atms-cls-i C 𝒜in = nfoldli C (λ-. True)
      (λL 𝒜in. do {
        ASSERT(atm-of L ≤ unat32-max div 2);
        RETURN(insert (atm-of L) 𝒜in)})
    𝒜in›

lemma fild-insert-insert-swap:
  ‹fold (λL. insert (f L)) C (insert a 𝒜in) = insert a (fold (λL. insert (f L)) C 𝒜in)›
  by (induction C arbitrary: a 𝒜in) (auto simp: extract-atms-cls-def)

lemma extract-atms-cls-alt-def: ‹extract-atms-cls C 𝒜in = 𝒜in ∪ atm-of ‘ set C›
  by (induction C) (auto simp: extract-atms-cls-def fild-insert-insert-swap)

lemma extract-atms-cls-i-extract-atms-cls:
  ‹(uncurry extract-atms-cls-i, uncurry (RETURN oo extract-atms-cls))
   ∈ [λ(C, 𝒜in). ∀ L∈set C. nat-of-lit L ≤ unat32-max]f
     ⟨Id⟩list-rel ×f Id → ⟨Id⟩nres-rel›
proof −
  have H1: ‹(x1a, x1) ∈ ⟨{(L, L'). L = L' ∧ nat-of-lit L ≤ unat32-max}⟩list-rel›
    if
      ‹case y of (C, 𝒜in) ⇒ ∀ L∈set C. nat-of-lit L ≤ unat32-max› and
      ‹(x, y) ∈ ⟨nat-lit-lit-rel⟩list-rel ×f Id› and
      ‹y = (x1, x2)› and
      ‹x = (x1a, x2a)›
    for x :: ‹nat literal list × nat set› and y :: ‹nat literal list × nat set› and
      x1 :: ‹nat literal list› and x2 :: ‹nat set› and x1a :: ‹nat literal list› and x2a :: ‹nat set›
    using that by (auto simp: list-rel-def list-all2-conj list.rel-eq list-all2-conv-all-nth)

  have atm-le: ‹nat-of-lit xa ≤ unat32-max ⟹ atm-of xa ≤ unat32-max div 2› for xa
    by (cases xa) (auto simp: unat32-max-def)

  show ?thesis
    supply RETURN-as-SPEC-refine[refine2 del]
    unfolding extract-atms-cls-i-def extract-atms-cls-def uncurry-def comp-def
      fold-eq-nfoldli
    apply (intro frefI nres-relI)
    apply (refine-rcg H1)
        apply assumption+
    subgoal by auto
```

subgoal by *auto*
subgoal by (*auto simp*: *atm-le*)
subgoal by *auto*
done
qed

definition *extract-atms-clss*:: ‹'*a clause-l list* ⇒ '*a set* ⇒ '*a set*› **where**
‹*extract-atms-clss N* $\mathcal{A}_{in}$ = *fold extract-atms-cls N* $\mathcal{A}_{in}$›

definition *extract-atms-clss-i* :: ‹*nat clause-l list* ⇒ *nat set* ⇒ *nat set nres*› **where**
‹*extract-atms-clss-i N* $\mathcal{A}_{in}$ = *nfoldli N* (λ-. *True*) *extract-atms-cls-i* $\mathcal{A}_{in}$›

lemma *extract-atms-clss-i-extract-atms-clss*:
‹(*uncurry extract-atms-clss-i*, *uncurry* (*RETURN oo extract-atms-clss*))
∈ [λ(*N*, $\mathcal{A}_{in}$). ∀ *C*∈*set N*. ∀ *L*∈*set C*. *nat-of-lit L* ≤ *unat32-max*]$_f$
⟨*Id*⟩*list-rel* ×$_f$ *Id* → ⟨*Id*⟩*nres-rel*›
proof −
have *H1*: ‹(*x1a*, *x1*) ∈ ⟨{(*C*, *C'*). *C* = *C'* ∧ (∀ *L*∈*set C*. *nat-of-lit L* ≤ *unat32-max*)}⟩*list-rel*›
if
‹*case y of* (*N*, $\mathcal{A}_{in}$) ⇒ ∀ *C*∈*set N*. ∀ *L*∈*set C*. *nat-of-lit L* ≤ *unat32-max*› **and**
‹(*x*, *y*) ∈ ⟨*Id*⟩*list-rel* ×$_f$ *Id*› **and**
‹*y* = (*x1*, *x2*)› **and**
‹*x* = (*x1a*, *x2a*)›
for *x* :: ‹*nat literal list list* × *nat set*› **and** *y* :: ‹*nat literal list list* × *nat set*› **and**
x1 :: ‹*nat literal list list*› **and** *x2* :: ‹*nat set*› **and** *x1a* :: ‹*nat literal list list*›
and *x2a* :: ‹*nat set*›
using *that* **by** (*auto simp*: *list-rel-def list-all2-conj list.rel-eq list-all2-conv-all-nth*)

show *?thesis*
supply *RETURN-as-SPEC-refine*[*refine2 del*]
unfolding *extract-atms-clss-i-def extract-atms-clss-def comp-def fold-eq-nfoldli uncurry-def*
apply (*intro frefI nres-relI*)
apply (*refine-vcg H1 extract-atms-cls-i-extract-atms-cls*[*THEN fref-to-Down-curry*,
unfolded comp-def])
apply *assumption*+
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

lemma *fold-extract-atms-cls-union-swap*:
‹*fold extract-atms-cls N* ($\mathcal{A}_{in}$ ∪ *a*) = *fold extract-atms-cls N* $\mathcal{A}_{in}$ ∪ *a*›
by (*induction N arbitrary*: *a* $\mathcal{A}_{in}$) (*auto simp*: *extract-atms-cls-alt-def*)

lemma *extract-atms-clss-alt-def*:
‹*extract-atms-clss N* $\mathcal{A}_{in}$ = $\mathcal{A}_{in}$ ∪ ((⋃ *C*∈*set N*. *atm-of* ' *set C*))›
by (*induction N*)
(*auto simp*: *extract-atms-clss-def extract-atms-cls-alt-def*
fold-extract-atms-cls-union-swap)

lemma *finite-extract-atms-clss*[*simp*]: ‹*finite* (*extract-atms-clss CS'* {})› **for** *CS'*

by (*auto simp*: *extract-atms-clss-alt-def*)

definition *op-extract-list-empty* **where**
‹*op-extract-list-empty* = {}›

definition *extract-atms-clss-imp-empty-rel* **where**
‹*extract-atms-clss-imp-empty-rel* = (*RETURN* (*replicate 1024 0*, *0*, []))›

lemma *extract-atms-clss-imp-empty-rel*:
‹(λ-. *extract-atms-clss-imp-empty-rel*, λ-. (*RETURN op-extract-list-empty*)) ∈
unit-rel →$_f$ ⟨*isasat-atms-ext-rel*⟩ *nres-rel*›
by (*intro frefI nres-relI*)
(*simp add*: *op-extract-list-empty-def unat32-max-def*
isasat-atms-ext-rel-def init-valid-rep-def extract-atms-clss-imp-empty-rel-def
del: *replicate-numeral*)

lemma *extract-atms-cls-Nil*[*simp*]:
‹*extract-atms-cls* [] $\mathcal{A}_{in}$ = $\mathcal{A}_{in}$›
unfolding *extract-atms-cls-def fold.simps* **by** *simp*

lemma *extract-atms-clss-Cons*[*simp*]:
‹*extract-atms-clss* (*C* # *Cs*) *N* = *extract-atms-clss Cs* (*extract-atms-cls C N*)›
by (*simp add*: *extract-atms-clss-def*)

definition (**in** −) *all-lits-of-atms-m* :: ‹'*a multiset* ⇒ '*a clause*› **where**
‹*all-lits-of-atms-m N* = *poss N* + *negs N*›

lemma (**in** −) *all-lits-of-atms-m-nil*[*simp*]: ‹*all-lits-of-atms-m* {#} = {#}›
unfolding *all-lits-of-atms-m-def* **by** *auto*

definition (**in** −) *all-lits-of-atms-mm* :: ‹'*a multiset multiset* ⇒ '*a clause*› **where**
‹*all-lits-of-atms-mm N* = *poss* (∑$_\#$ *N*) + *negs* (∑$_\#$ *N*)›

lemma *all-lits-of-atms-m-all-lits-of-m*:
‹*all-lits-of-atms-m N* = *all-lits-of-m* (*poss N*)›
unfolding *all-lits-of-atms-m-def all-lits-of-m-def*
by (*induction N*) *auto*

Creation of an initial state

definition *init-dt-wl-heur-spec*
:: ‹*bool* ⇒ *nat multiset* ⇒ *nat clause-l list* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init* ⇒ *bool*›
where
‹*init-dt-wl-heur-spec unbdd* $\mathcal{A}$ *CS T TOC* ⟷
(∃ *T' TOC'*. (*TOC*, *TOC'*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* ∧ (*T*, *T'*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* ∧
init-dt-wl-spec CS T' TOC')›

definition *init-state-wl* :: ‹*nat twl-st-wl-init'*› **where**
‹*init-state-wl* = ([], *fmempty*, *None*, {#}, {#}, {#}, {#}, {#}, {#}, {#}, {#})›

definition *init-state-wl-heur* :: ‹*nat multiset* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*init-state-wl-heur* $\mathcal{A}$ = *do* {
M ← *SPEC*(λ*M*. (*M*, []) ∈ *trail-pol* $\mathcal{A}$);

D ← *SPEC*(λ*D*. (*D*, *None*) ∈ *option-lookup-clause-rel* $\mathcal{A}$);
mark ← *SPEC*(λ*D*. (*D*, {#}) ∈ *lookup-clause-rel* $\mathcal{A}$);
W ← *SPEC* (λ*W*. (*W*, *empty-watched* $\mathcal{A}$) ∈ ⟨*Id*⟩*map-fun-rel* (*D*$_0$ $\mathcal{A}$));
vm ← *RES* (*bump-heur-init* $\mathcal{A}$ []);
φ ← *SPEC* (*phase-saving* $\mathcal{A}$);
cach ← *SPEC* (*cach-refinement-empty* $\mathcal{A}$);
let lbd = *empty-lbd*;
let vdom = [];
let ivdom = [];
let lcount = (0,0,0,0,0);
RETURN (*Tuple15 M* [] *D 0 W vm* φ *0 cach lbd vdom ivdom False lcount mark*)}›

definition *init-state-wl-heur-fast* **where**
‹*init-state-wl-heur-fast* = *init-state-wl-heur*›

lemma *clss-size-empty* [*simp*]: ‹*clss-size fmempty* {#} {#} {#} {#} {#} {#} {#} {#} = (0, 0 ,0, 0, 0)›
by (*auto simp*: *clss-size-def*)
lemma *clss-size-corr-empty* [*simp*]: ‹*clss-size-corr fmempty* {#} {#} {#} {#} {#} {#} {#} {#} (0, 0 ,0, 0, 0)›
by (*auto simp*: *clss-size-corr-def*)

lemma *init-state-wl-heur-init-state-wl*:
‹(λ-. (*init-state-wl-heur* $\mathcal{A}$), λ-. (*RETURN init-state-wl*)) ∈
[λ-. *isasat-input-bounded* $\mathcal{A}$]$_f$ *unit-rel* → ⟨*twl-st-heur-parsing-no-WL-wl* $\mathcal{A}$ *unbdd*⟩*nres-rel*›
by (*intro frefI nres-relI*)
(*auto simp*: *init-state-wl-heur-def init-state-wl-def*
RES-RETURN-RES bind-RES-RETURN-eq RES-RES-RETURN-RES RETURN-def
twl-st-heur-parsing-no-WL-wl-def vdom-m-def empty-watched-def valid-arena-empty
intro!: *RES-refine*)

definition (**in** −)*to-init-state* :: ‹*nat twl-st-wl-init'* ⇒ *nat twl-st-wl-init*› **where**
‹*to-init-state S* = (*S*, {#})›

definition (**in** −) *from-init-state* :: ‹*nat twl-st-wl-init-full* ⇒ *nat twl-st-wl*› **where**
‹*from-init-state* = *fst*›

definition (**in** −) *to-init-state-code* **where**
‹*to-init-state-code* = *id*›

definition *from-init-state-code* **where**
‹*from-init-state-code* = *id*›

definition (**in** −) *conflict-is-None-heur-wl* **where**
‹*conflict-is-None-heur-wl* = (λ(*M*, *N*, *U*, *D*, -). *is-None D*)›

definition (**in** −) *finalise-init* **where**
‹*finalise-init* = *id*›

17.1.4 Parsing

lemma *init-dt-wl-heur-init-dt-wl*:

‹(*uncurry* (*init-dt-wl-heur unbdd*), *uncurry init-dt-wl*) ∈
[λ(*CS*, *S*). (∀ *C* ∈ *set CS*. *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*))]$_f$
⟨*Id*⟩*list-rel* ×$_f$ {((*S*, *C*), *T*). *C* = [] ∧ (*S*,*T*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*} →
⟨{((*S*, *C*), *T*). *C* = [] ∧ (*S*,*T*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*}⟩ *nres-rel*›
proof −
have *H*: ‹⋀*x y x1 x2 x1a x2a*.
(∀ *C*∈*set x1*. *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*)) ⟹
(*x1a*, *x1*) ∈ ⟨*Id*⟩*list-rel* ⟹
(*x1a*, *x1*) ∈ ⟨{(*C*, *C'*). *C* = *C'* ∧ *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*)}⟩*list-rel*›
apply (*auto simp*: *list-rel-def list-all2-conj*)
apply (*auto simp*: *list-all2-conv-all-nth distinct-mset-set-def*)
done

show *?thesis*
unfolding *init-dt-wl-heur-def init-dt-wl-def uncurry-def*
apply (*intro frefI nres-relI*)
apply (*case-tac y rule*: *prod.exhaust*)
apply (*case-tac x rule*: *prod.exhaust*)
apply (*simp only*: *prod.case prod-rel-iff*)
apply (*refine-vcg init-dt-step-wl-heur-init-dt-step-wl*[*THEN fref-to-Down-curry*] *H*)
apply *normalize-goal*+
subgoal by *fast*
subgoal by *fast*
subgoal by *simp*
subgoal by *auto*
subgoal by *auto*
done
qed

Full Initialisation

definition *rewatch-heur-st-fast* **where**
‹*rewatch-heur-st-fast* = *rewatch-heur-st*›

definition *rewatch-heur-st-fast-pre* **where**
‹*rewatch-heur-st-fast-pre S* =
((∀ *x* ∈ *set* (*get-vdom-heur-init S*). *x* ≤ *snat64-max*) ∧ *length* (*get-clauses-wl-heur-init S*) ≤ *snat64-max*)›

definition *rewatch-heur-st-init*
:: ‹*twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*›
where
‹*rewatch-heur-st-init* = (λ*S*. *do* {
ASSERT(*length* ((*get-vdom-heur-init S*)) ≤ *length* (*get-clauses-wl-heur-init S*));
W ← *rewatch-heur* ((*get-vdom-heur-init S*)) (*get-clauses-wl-heur-init S*) (*get-watchlist-wl-heur-init S*);
RETURN (*set-watchlist-wl-heur-init W S*)
})›

definition *init-dt-wl-heur-full*
:: ‹*bool* ⇒ - ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*›
where
‹*init-dt-wl-heur-full unb CS S* = *do* {
(*S*, *C*) ← *init-dt-wl-heur unb CS* (*S*, []);
ASSERT(¬*is-failed-heur-init S*);
rewatch-heur-st-init S
}›

definition *init-dt-wl-heur-full-unb*
:: ‹- ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*›
where
‹*init-dt-wl-heur-full-unb* = *init-dt-wl-heur-full True*›

lemma *init-dt-wl-heur-full-init-dt-wl-full*:
assumes
‹*init-dt-wl-pre CS T*› **and**
‹∀ *C*∈*set CS*. *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*)›
‹(*S*, *T*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *True*›
shows ‹*init-dt-wl-heur-full True CS S*
≤ ⇓ (*twl-st-heur-parsing* $\mathcal{A}$ *True*) (*init-dt-wl-full CS T*)›
proof −
have *H*: ‹*valid-arena* (*get-clauses-wl-heur-init x*) *x1b* (*set* (*get-vdom-heur-init x*))›
‹*set* (*get-vdom-heur-init x*) ⊆ *set* (*get-vdom-heur-init x*)› ‹*set-mset* (*dom-m x1b*) ⊆ *set* (*get-vdom-heur-init x*)›
‹*distinct* (*get-vdom-heur-init x*)› ‹(*get-watchlist-wl-heur-init x*, λ-. []) ∈ ⟨*Id*⟩*map-fun-rel* (*D*$_0$ $\mathcal{A}$)›
if
xx': ‹(*xa*, *x'*) ∈ {((*S*, *C*), *T*). *C* = [] ∧ (*S*,*T*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *True*}› **and**
st: ‹*xa* = (*x*, *a*)›
‹*x2c* = (*x1e*, *x2d*)›
‹*x2b* = (*x1d*, *x2c*)›
‹*x2a* = (*x1c*, *x2b*)›
‹*x2* = (*x1b*, *x2a*)›
‹*x1* = (*x1a*, *x2*)›
‹*x'* = (*x1*, *x2e*)›
for *x x' x1 x1a x2 x1b x2a x1c x2b x1d x2c x1e x2d x2e x1f x2f x1g x2g x1h x2h*
x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p xa a
proof −
show ‹*valid-arena* (*get-clauses-wl-heur-init x*) *x1b* (*set* (*get-vdom-heur-init x*))›
‹*set* (*get-vdom-heur-init x*) ⊆ *set* (*get-vdom-heur-init x*)› ‹*set-mset* (*dom-m x1b*) ⊆ *set* (*get-vdom-heur-init x*)›
‹*distinct* (*get-vdom-heur-init x*)› ‹(*get-watchlist-wl-heur-init x*, λ-. []) ∈ ⟨*Id*⟩*map-fun-rel* (*D*$_0$ $\mathcal{A}$)›
using *xx' distinct-mset-dom*[*of x1b*] **unfolding** *st*
by (*auto simp*: *twl-st-heur-parsing-no-WL-def empty-watched-def*
simp flip: *set-mset-mset distinct-mset-mset-distinct*)
qed

show *?thesis*
unfolding *init-dt-wl-heur-full-def init-dt-wl-full-def rewatch-heur-st-init-def*
apply (*refine-rcg rewatch-heur-rewatch*[*of - - - - - - -* $\mathcal{A}$]
init-dt-wl-heur-init-dt-wl[*of True* $\mathcal{A}$, *THEN fref-to-Down-curry*])
subgoal using *assms* **by** *fast*
subgoal using *assms* **by** *auto*
subgoal by (*auto simp*: *twl-st-heur-parsing-def twl-st-heur-parsing-no-WL-def*)
subgoal by (*auto dest*: *valid-arena-vdom-subset simp*: *twl-st-heur-parsing-no-WL-def*)
apply ((*rule H*; *assumption*)+)[*5*]
subgoal
by (*auto simp*: *twl-st-heur-parsing-def twl-st-heur-parsing-no-WL-def*
literals-are-in-$\mathcal{L}_{in}$*-mm-def all-lits-of-mm-union*)
subgoal by (*auto simp*: *twl-st-heur-parsing-def twl-st-heur-parsing-no-WL-def*
empty-watched-def[*symmetric*] *map-fun-rel-def vdom-m-def*)
subgoal by (*auto simp*: *twl-st-heur-parsing-def twl-st-heur-parsing-no-WL-def*
empty-watched-def[*symmetric*] *ac-simps*)
done

qed

lemma *init-dt-wl-heur-full-init-dt-wl-spec-full*:
assumes
‹*init-dt-wl-pre CS T*› **and**
‹∀ *C*∈*set CS*. *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*)› **and**
‹(*S*, *T*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *True*›
shows ‹*init-dt-wl-heur-full True CS S*
≤ ⇓ (*twl-st-heur-parsing* $\mathcal{A}$ *True*) (*SPEC* (*init-dt-wl-spec-full CS T*))›
apply (*rule order.trans*)
apply (*rule init-dt-wl-heur-full-init-dt-wl-full*[*OF assms*])
apply (*rule ref-two-step′*)
apply (*rule init-dt-wl-full-init-dt-wl-spec-full*[*OF assms*(*1*)])
done

17.1.5 Conversion to normal state

definition *extract-lits-sorted* **where**
‹*extract-lits-sorted* = (λ(*xs*, *n*, *vars*). *do* {
vars ← — insert_sort_nth2 xs vars*RETURN vars*;
RETURN (*vars*, *n*)
})›

definition *lits-with-max-rel* **where**
‹*lits-with-max-rel* = {((*xs*, *n*), $\mathcal{A}_{in}$). *mset xs* = $\mathcal{A}_{in}$ ∧ *n* = *Max* (*insert 0* (*set xs*)) ∧
length xs < *unat32-max*}›

lemma *extract-lits-sorted-mset-set*:
‹(*extract-lits-sorted*, *RETURN o mset-set*)
∈ *isasat-atms-ext-rel* →$_f$ ⟨*lits-with-max-rel*⟩*nres-rel*›
proof −
have *K*: ‹*RETURN o mset-set* = (λ*v*. *do* {*v′* ← *SPEC*(λ*v′*. *v′* = *mset-set v*); *RETURN v′*})›
by *auto*
have *K′*: ‹*length x2a* < *unat32-max*› **if** ‹*distinct b*› ‹*init-valid-rep x1* (*set b*)›
‹*length x1* < *unat32-max*› ‹*mset x2a* = *mset b*›**for** *x1 x2a b*
proof −
have ‹*distinct x2a*›
by (*simp add*: *same-mset-distinct-iff that*(*1*) *that*(*4*))
have ‹*length x2a* = *length b*› ‹*set x2a* = *set b*›
using ‹*mset x2a* = *mset b*› **apply** (*metis size-mset*)
using ‹*mset x2a* = *mset b*› **by** (*rule mset-eq-setD*)
then have ‹*set x2a* ⊆ {*0*..<*unat32-max* − *1*}›
using *that* **by** (*auto simp*: *init-valid-rep-def*)
from *card-mono*[*OF - this*] **show** *?thesis*
using ‹*distinct x2a*› **by** (*auto simp*: *unat32-max-def distinct-card*)
qed
have *H-simple*: ‹*RETURN x2a*
≤ ⇓ (*list-mset-rel* ∩ {(*v*, *v′*). *length v* < *unat32-max*})
(*SPEC* (λ*v′*. *v′* = *mset-set y*))›
if
‹(*x*, *y*) ∈ *isasat-atms-ext-rel*› **and**
‹*x2* = (*x1a*, *x2a*)› **and**
‹*x* = (*x1*, *x2*)›
for *x* :: ‹*nat list* × *nat* × *nat list*› **and** *y* :: ‹*nat set*› **and** *x1* :: ‹*nat list*› **and**

x2 :: ‹*nat* × *nat list*› **and** *x1a* :: ‹*nat*› **and** *x2a* :: ‹*nat list*›
using *that mset-eq-length* **by** (*auto simp*: *isasat-atms-ext-rel-def list-mset-rel-def br-def mset-set-set RETURN-def intro*: *K′ intro*!: *RES-refine dest*: *mset-eq-length*)

show *?thesis*
unfolding *extract-lits-sorted-def reorder-list-def K*
apply (*intro frefI nres-relI*)
apply (*refine-vcg H-simple*)
apply *assumption+*
by (*auto simp*: *lits-with-max-rel-def isasat-atms-ext-rel-def mset-set-set list-mset-rel-def br-def dest*!: *mset-eq-setD*)
qed

TODO Move

definition *reduce-interval-init* :: ‹*64 word*› **where** ‹*reduce-interval-init* = *300*›

This value is taken from CaDiCaL.

definition *inprocessing-interval-init* :: ‹*64 word*› **where** ‹*inprocessing-interval-init* = *100000*›

definition *rephasing-initial-phase* :: ‹*64 word*› **where** ‹*rephasing-initial-phase* = *10*›
definition *rephasing-end-of-initial-phase* :: ‹*64 word*› **where** ‹*rephasing-end-of-initial-phase* = *10000*›

definition *subsuming-length-initial-phase* :: ‹*64 word*› **where** ‹*subsuming-length-initial-phase* = *10000*›

definition *finalize-vmtf-init* **where** ‹
finalize-vmtf-init = (λ(*ns*, *m*, *fst-As*, *lst-As*, *next-search*). *do* {
ASSERT (*fst-As* ≠ *None*);
ASSERT (*lst-As* ≠ *None*);
RETURN (*ns*, *m*, *the fst-As*, *the lst-As*, *next-search*)})›

definition *finalize-bump-init* :: ‹*bump-heuristics-init* ⇒ *bump-heuristics nres*› **where**
‹*finalize-bump-init* = (λ*x*. *case x of Tuple4 hstable focused foc to-remove* ⇒ *do* {
focused ← *finalize-vmtf-init focused*;
RETURN (*Tuple4 hstable focused foc to-remove*)
})›

The value 160 is random (but larger than the default 16 for array lists).

definition *finalise-init-code* :: ‹*opts* ⇒ *twl-st-wl-heur-init* ⇒ *isasat nres*› **where**
‹*finalise-init-code opts* =
(λ*S*. *case S of Tuple15 M′ N′ D′ Q′ W′ bmp φ clvls cach*
lbd vdom ivdom - lcount mark ⇒ *do* {
let init-stats = *empty-stats-clss* (*of-nat* (*length ivdom*));
let fema = *ema-init* (*opts-fema opts*);
let sema = *ema-init* (*opts-sema opts*);
let other-fema = *ema-init* (*opts-fema opts*);
let other-sema = *ema-init* (*opts-sema opts*);
let ccount = *restart-info-init*;
let heur = *Restart-Heuristics* ((*fema*, *sema*, *ccount*, *0*, (*φ*, *0*, *replicate* (*length φ*) *False*, *0*, *replicate* (*length φ*) *False*, *rephasing-end-of-initial-phase*, *0*, *rephasing-initial-phase*), *reluctant-init*, *False*, *replicate* (*length φ*) *False*, (*inprocessing-interval-init*, *reduce-interval-init*, *subsuming-length-initial-phase*), *other-fema*, *other-sema*));
let vdoms = *AIvdom-init vdom* [] *ivdom*;
let occs = *replicate* (*length W′*) [];
bmp ← *finalize-bump-init bmp*;
RETURN (*IsaSAT M′ N′ D′ Q′ W′ bmp*

clvls cach lbd (*take 1* (*replicate 160* (*Pos 0*))) *init-stats*
heur vdoms lcount opts [] *occs*)
})›

lemma *isa-vmtf-init-nemptyD*:
‹((*ak*, *al*, *am*, *an*, *bc*)) ∈ *isa-vmtf-init* $\mathcal{A}$ *au* ⟹ $\mathcal{A} \neq \{\#\}$ ⟹ ∃ *y*. *an* = *Some y*›
‹((*ak*, *al*, *am*, *an*, *bc*)) ∈ *isa-vmtf-init* $\mathcal{A}$ *au* ⟹ $\mathcal{A} \neq \{\#\}$ ⟹ ∃ *y*. *am* = *Some y*›
by (*auto simp*: *isa-vmtf-init-def bump-heur-init-def*)

lemma *isa-vmtf-init-isa-vmtf*: ‹$\mathcal{A} \neq \{\#\}$ ⟹ ((*ak*, *al*, *Some am*, *Some an*, *bc*))
∈ *isa-vmtf-init* $\mathcal{A}$ *au* ⟹ ((*ak*, *al*, *am*, *an*, *bc*))
∈ *vmtf* $\mathcal{A}$ *au*›
by (*auto simp*: *isa-vmtf-init-def Image-iff*)

lemma *bump-heur-init-isa-vmtf*: ‹$\mathcal{A} \neq \{\#\}$ ⟹ *x* ∈ *bump-heur-init* $\mathcal{A}$ *M* ⟹ *finalize-bump-init x* ≤
⇓*Id* (*SPEC* (λ*x*. *x* ∈ *bump-heur* $\mathcal{A}$ *M*))›
unfolding *finalize-bump-init-def bump-heur-init-def bump-heur-def finalize-vmtf-init-def*
nres-monad3
apply (*cases x*, *hypsubst*, *simp*)
apply *refine-vcg*
by (*auto dest*: *isa-vmtf-init-nemptyD intro*!: *isa-vmtf-init-isa-vmtf*)

lemma *heuristic-rel-initI*:
‹*phase-saving* $\mathcal{A}$ *φ* ⟹ *length φ′* = *length φ* ⟹ *length φ″* = *length φ* ⟹ *phase-saving* $\mathcal{A}$ *g* ⟹
heuristic-rel $\mathcal{A}$ (*Restart-Heuristics* ((*fema*, *sema*, *ccount*, *0*, (*φ*,*a*, *φ′*,*b*,*φ″*,*c*,*d*), *e*, *f*, *g*, *h*)))›
by (*auto simp*: *heuristic-rel-def phase-save-heur-rel-def phase-saving-def heuristic-rel-stats-def*)

lemma *init-empty-occ-list-from-WL-length*: ‹(*x5*, *m*) ∈ ⟨*Id*⟩*map-fun-rel* (*D*$_0$ *A*) ⟹
(*replicate* (*length x5*) [], *empty-occs-list A*) ∈ *occurrence-list-ref*›
by (*auto simp*: *occurrence-list-ref-def map-fun-rel-def empty-occs-list-def* $\mathcal{L}_{all}$*-add-mset*
dest!: *multi-member-split*)

lemma *finalise-init-finalise-init-full*:
‹*get-conflict-wl S* = *None* ⟹
all-atms-st S ≠ {#} ⟹ *size* (*learned-clss-l* (*get-clauses-wl S*)) = *0* ⟹
((*ops′*, *T*), *ops*, *S*) ∈ *Id* ×$_f$ *twl-st-heur-post-parsing-wl True* ⟹
finalise-init-code ops′ T ≤ ⇓ {(*S′*, *T′*). (*S′*, *T′*) ∈ *twl-st-heur* ∧
get-clauses-wl-heur-init T = *get-clauses-wl-heur S′* ∧
aivdom-inv-strong-dec (*get-aivdom S′*) (*dom-m* (*get-clauses-wl T′*)) ∧
get-learned-count-init T = *get-learned-count S′*} (*RETURN* (*finalise-init S*))›
apply (*cases S*; *cases T*)
apply (*simp add*: *finalise-init-code-def aivdom-inv-strong-dec-def2 split*:*prod.splits*)
apply (*auto simp*: *finalise-init-def twl-st-heur-def twl-st-heur-parsing-no-WL-def*
twl-st-heur-parsing-no-WL-wl-def distinct-mset-dom AIvdom-init-def
finalise-init-code-def out-learned-def all-lits-st-alt-def[*symmetric*]
twl-st-heur-post-parsing-wl-def all-atms-st-def aivdom-inv-dec-def
intro!: *ASSERT-leI intro*!: *heuristic-rel-initI*
specify-left-RES[*OF bump-heur-init-isa-vmtf*[**where** $\mathcal{A}$= ‹*all-atms-st S*› **and** *M*=‹*get-trail-wl S*›,
unfolded conc-fun-RES]]
intro:)
apply (*auto simp*: *finalise-init-def twl-st-heur-def twl-st-heur-parsing-no-WL-def*
twl-st-heur-parsing-no-WL-wl-def distinct-mset-dom AIvdom-init-def init-empty-occ-list-from-WL-length
finalise-init-code-def out-learned-def all-lits-st-alt-def[*symmetric*] *phase-saving-def*
twl-st-heur-post-parsing-wl-def all-atms-st-def aivdom-inv-dec-def ac-simps
intro!: *ASSERT-leI intro*!: *heuristic-rel-initI*
intro:)

done

lemma *finalise-init-finalise-init*:
‹(*uncurry finalise-init-code*, *uncurry* (*RETURN oo* (λ-. *finalise-init*))) ∈
[λ(-, *S*::*nat twl-st-wl*). *get-conflict-wl S* = *None* ∧ *all-atms-st S* ≠ {#} ∧
size (*learned-clss-l* (*get-clauses-wl S*)) = 0]$_f$ *Id* ×$_r$
twl-st-heur-post-parsing-wl True → ⟨*twl-st-heur*⟩*nres-rel*›
apply (*intro frefI nres-relI*)
subgoal for *x y*
using *finalise-init-finalise-init-full*[*of* ‹*snd y*› ‹*fst x*› ‹*snd x*› ‹*fst y*›]
by (*cases x*; *cases y*)
(*auto intro*: *weaken-⇓′*)
done

definition (**in** −) *init-rll* :: ‹*nat* ⇒ (*nat*, *′v clause-l* × *bool*) *fmap*› **where**
‹*init-rll n* = *fmempty*›

definition (**in** −) *init-aa* :: ‹*nat* ⇒ *′v list*› **where**
‹*init-aa n* = []›

definition (**in** −) *init-aa′* :: ‹*nat* ⇒ (*clause-status* × *nat* × *nat*) *list*› **where**
‹*init-aa′ n* = []›

definition *init-trail-D* :: ‹*nat list* ⇒ *nat* ⇒ *nat* ⇒ *trail-pol nres*› **where**
‹*init-trail-D* $\mathcal{A}_{in}$ *n m* = *do* {
let M0 = [];
let cs = [];
let M = *replicate m UNSET*;
let M′ = *replicate n 0*;
let M′′ = *replicate n 1*;
RETURN ((*M0*, *M*, *M′*, *M′′*, *0*, *cs*, *0*))
}›

definition *init-trail-D-fast* **where**
‹*init-trail-D-fast* = *init-trail-D*›

definition *init-state-wl-D′* :: ‹*nat list* × *nat* ⇒ (*twl-st-wl-heur-init*) *nres*› **where**
‹*init-state-wl-D′* = (λ(*A*$_{in}$, *n*). *do* {
ASSERT(*Suc* (*2* ∗ (*n*)) ≤ *unat32-max*);
let n = *Suc* (*n*);
let m = *2* ∗ *n*;
M ← *init-trail-D* $\mathcal{A}_{in}$ *n m*;
let N = [];
let D = (*True*, *0*, *replicate n NOTIN*);
let mark = (*0*, *replicate n None*);
let WS = *replicate m* [];
vm ← *initialize-Bump-Init* $\mathcal{A}_{in}$ *n*;
let φ = *replicate n False*;
let cach = (*replicate n SEEN-UNKNOWN*, []);
let lbd = *empty-lbd*;
let vdom = [];
let ivdom = [];
let lcount = (*0*, *0*, *0*, *0*, *0*);

```
    RETURN (Tuple15 M N D 0 WS vm φ 0 cach lbd vdom ivdom False lcount mark)
  })⟩

lemma init-trail-D-ref:
  ⟨(uncurry2 init-trail-D, uncurry2 (RETURN ooo (λ - - -. []))) ∈ [λ((N, n), m). mset N = 𝒜_in ∧
    distinct N ∧ (∀ L∈set N. L < n) ∧ m = 2 * n ∧ isasat-input-bounded 𝒜_in]_f
    ⟨Id⟩list-rel ×_f nat-rel ×_f nat-rel →
   ⟨trail-pol 𝒜_in⟩ nres-rel⟩
proof −
  have K: ⟨(∀ L∈set N. L < n) ⟷
    (∀ L ∈# (𝓛_all (mset N)). atm-of L < n)⟩ for N n
    apply (rule iffI)
    subgoal by (auto simp: in-𝓛_all-atm-of-𝒜_in)
    subgoal by (metis (full-types) image-eqI in-𝓛_all-atm-of-𝒜_in literal.sel(1)
        set-image-mset set-mset-mset)
    done
  have K': ⟨(∀ L∈set N. L < n) ⟹
    (∀ L ∈# (𝓛_all (mset N)). nat-of-lit L < 2 * n)⟩
    (is ⟨?A ⟹ ?B⟩) for N n

  proof −
    assume ?A
    then show ?B
      apply (auto simp: in-𝓛_all-atm-of-𝒜_in)
      apply (case-tac L)
      apply auto
      done
  qed
  show ?thesis
    unfolding init-trail-D-def
    apply (intro frefI nres-relI)
    unfolding uncurry-def Let-def comp-def trail-pol-def
    apply clarify
    unfolding RETURN-refine-iff
    apply clarify
    apply (intro conjI)
    subgoal
      by (auto simp: ann-lits-split-reasons-def
          list-mset-rel-def Collect-eq-comp list-rel-def
          list-all2-op-eq-map-right-iff' Id-def
          br-def in-𝓛_all-atm-of-in-atms-of-iff atms-of-𝓛_all-𝒜_in
        dest: multi-member-split)
    subgoal
      by auto
    subgoal using K' by (auto simp: polarity-def)
    subgoal
      by (auto simp:
        nat-shiftr-div2 in-𝓛_all-atm-of-in-atms-of-iff
        polarity-atm-def trail-pol-def K
        phase-saving-def list-rel-mset-rel-def  atms-of-𝓛_all-𝒜_in
        list-rel-def Id-def br-def list-all2-op-eq-map-right-iff'
        ann-lits-split-reasons-def
      list-mset-rel-def Collect-eq-comp)
    subgoal
      by auto
    subgoal
```

by *auto*
subgoal
by (*auto simp*: *control-stack.empty*)
subgoal by (*auto simp*: *zeroed-trail-def*)
subgoal by *auto*
done
qed

fun *to-tuple* **where**
‹*to-tuple* (*Tuple15 a b c d e f g h i j k l m n ko*) = (*a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*, *i*, *j*, *k*, *l*, *m*, *n*, *ko*)›

definition *tuple15-rel* **where** *tuple15-rel-internal-def*:
‹*tuple15-rel A B C D E F G H I J K L M N KO* = {(*S*,*T*).
(*case* (*S*, *T*) *of*
(*Tuple15 a b c d e f g h i j k l m n ko*,
Tuple15 a′ b′ c′ d′ e′ f′ g′ h′ i′ j′ k′ l′ m′ n′ ko′) ⇒
(*a*, *a′*) ∈ *A* ∧ (*b*,*b′*) ∈ *B* ∧ (*c*,*c′*) ∈ *C* ∧ (*d*,*d′*)∈*D* ∧ (*e*,*e′*)∈*E* ∧
(*f*,*f′*)∈*F* ∧ (*g*,*g′*) ∈ *G* ∧ (*h*,*h′*)∈*H* ∧ (*i*,*i′*)∈*I* ∧ (*j*,*j′*) ∈ *J* ∧ (*k*,*k′*)∈*K* ∧
(*l*,*l′*)∈*L* ∧ (*m*,*m′*)∈*M* ∧ (*n*,*n′*)∈*N* ∧ (*ko*,*ko′*)∈*KO*)}›

lemma *tuple15-rel-def*:
‹⟨*A*,*B*,*C*,*D*,*E*,*F*,*G*,*H*,*I*,*J*,*K*,*L*,*M*,*N*,*KO*⟩*tuple15-rel* ≡ {(*a*,*b*). *case* (*a*,*b*) *of*
(*Tuple15 a b c d e f g h i j k l m n ko*,
Tuple15 a′ b′ c′ d′ e′ f′ g′ h′ i′ j′ k′ l′ m′ n′ ko′) ⇒
(*a*, *a′*) ∈ *A* ∧ (*b*,*b′*) ∈ *B* ∧ (*c*,*c′*) ∈ *C* ∧ (*d*,*d′*)∈*D* ∧ (*e*,*e′*)∈*E* ∧
(*f*,*f′*)∈*F* ∧ (*g*,*g′*) ∈ *G* ∧ (*h*,*h′*)∈*H* ∧ (*i*,*i′*)∈*I* ∧ (*j*,*j′*) ∈ *J* ∧ (*k*,*k′*)∈*K* ∧
(*l*,*l′*)∈*L* ∧ (*m*,*m′*)∈*M* ∧ (*n*,*n′*)∈*N* ∧ (*ko*,*ko′*)∈*KO*}›
by (*simp add*: *tuple15-rel-internal-def relAPP-def*)

lemma *to-tuple-eq-iff*[*iff*]: ‹*to-tuple S* = *to-tuple T* ⟷ *S* = *T*›
by (*cases S*; *cases T*) *auto*

lemma *init-state-wl-D0*:
‹(*init-state-wl-D′*, *init-state-wl-heur*) ∈
[λ*N*. *N* = $\mathcal{A}_{in}$ ∧ *distinct-mset* $\mathcal{A}_{in}$ ∧ *isasat-input-bounded* $\mathcal{A}_{in}$]$_f$
lits-with-max-rel O ⟨*Id*⟩*mset-rel* →
⟨⟨*Id*, *Id*, *Id*, *nat-rel*, ⟨⟨*Id*⟩*list-rel*⟩*list-rel*,
Id, ⟨*bool-rel*⟩*list-rel*, *Id*, *Id*, *Id*, *Id*, *Id*, *Id*, *Id*⟩*tuple15-rel*⟩*nres-rel*›
(**is** ‹*?C* ∈ [*?Pre*]$_f$ *?arg* → ⟨*?im*⟩*nres-rel*›)
proof −
have *init-state-wl-heur-alt-def*: ‹*init-state-wl-heur* $\mathcal{A}_{in}$ = *do* {
M ← *SPEC* (λ*M*. (*M*, []) ∈ *trail-pol* $\mathcal{A}_{in}$);
N ← *RETURN* [];
D ← *SPEC* (λ*D*. (*D*, *None*) ∈ *option-lookup-clause-rel* $\mathcal{A}_{in}$);
mark ← *SPEC* (λ*D*. (*D*, {#}) ∈ *lookup-clause-rel* $\mathcal{A}_{in}$);
W ← *SPEC* (λ*W*. (*W*, *empty-watched* $\mathcal{A}_{in}$) ∈ ⟨*Id*⟩*map-fun-rel* (*D*$_0$ $\mathcal{A}_{in}$));
vm ← *RES* (*bump-heur-init* $\mathcal{A}_{in}$ []);
φ ← *SPEC* (*phase-saving* $\mathcal{A}_{in}$);
cach ← *SPEC* (*cach-refinement-empty* $\mathcal{A}_{in}$);
let lbd = *empty-lbd*;
let vdom = [];
let ivdom = [];
let lcount = (*0*, *0*, *0*, *0*, *0*);
RETURN (*Tuple15 M N D 0 W vm* φ *0 cach lbd vdom ivdom False lcount mark*)}› **for** $\mathcal{A}_{in}$
unfolding *init-state-wl-heur-def Let-def* **by** *auto*

have *tr*: ‹*distinct-mset* $\mathcal{A}_{in}$ ∧ (∀ *L*∈#$\mathcal{A}_{in}$. *L* < *b*) ⟹
(*A*$_{in}$′, $\mathcal{A}_{in}$) ∈ ⟨*Id*⟩*list-rel-mset-rel* ⟹ *isasat-input-bounded* $\mathcal{A}_{in}$ ⟹
b′ = 2 ∗ *b* ⟹
init-trail-D $\mathcal{A}_{in}$′ *b* (2 ∗ *b*) ≤ ⇓ (*trail-pol* $\mathcal{A}_{in}$) (*RETURN* [])› **for** *b*′ *b* $\mathcal{A}_{in}$ *A*$_{in}$′ *x*
by (*rule init-trail-D-ref*[*unfolded fref-def nres-rel-def*, *simplified*, *rule-format*])
(*auto simp*: *list-rel-mset-rel-def list-mset-rel-def br-def*)

have [*simp*]: ‹*comp-fun-idem* (*max* :: 'b :: {*zero*,*linorder*} ⇒ -)›
unfolding *comp-fun-idem-def comp-fun-commute-def comp-fun-idem-axioms-def*
by (*auto simp*: *max-def*[*abs-def*] *intro*!: *ext*)
have [*simp*]: ‹*fold max x a* = *Max* (*insert a* (*set x*))› **for** *x* **and** *a* :: ‹'*b* :: {*zero*,*linorder*}›
by (*auto simp flip*: *Max.eq-fold Max.set-eq-fold*)
have *in-N0*: ‹*L* ∈ *set* $\mathcal{A}_{in}$ ⟹ *L* < *Suc* ((*Max* (*insert 0* (*set* $\mathcal{A}_{in}$))))›
for *L* $\mathcal{A}_{in}$
using *Max-ge*[*of* ‹*insert 0* (*set* $\mathcal{A}_{in}$)› *L*]
by (*auto simp del*: *Max-ge simp*: *nat-shiftr-div2*)
define *P* **where** ‹*P x* = {(*a*, *b*). *b* = [] ∧ (*a*, *b*) ∈ *trail-pol x*}› **for** *x*
have *P*: ‹(*c*, []) ∈ *P x* ⟷ (*c*, []) ∈ *trail-pol x*› **for** *c x*
unfolding *P-def* **by** *auto*
have [*simp*]: ‹{*p*. ∃ *x*. *p* = (*x*, *x*)} = {(*y*, *x*). *x* = *y*}›
by *auto*
have [*simp*]: ‹⋀*a* $\mathcal{A}_{in}$. (*a*, $\mathcal{A}_{in}$) ∈ ⟨*nat-rel*⟩*mset-rel* ⟷ $\mathcal{A}_{in}$ = *a*›
by (*auto simp*: *Id-def br-def in-mset-rel-eq-f-iff list-rel-mset-rel-def*
in-mset-rel-eq-f-iff)

have [*simp*]: ‹(*a*, *mset a*) ∈ ⟨*Id*⟩*list-rel-mset-rel*› **for** *a*
unfolding *list-rel-mset-rel-def*
by (*rule relcompI* [*of* - ‹*a*›])
(*auto simp*: *list-rel-def Id-def br-def list-all2-op-eq-map-right-iff*′
list-mset-rel-def)
have *init*: ‹*init-trail-D x1* (*Suc* (*x2*))
(2 ∗ *Suc* (*x2*)) ≤
SPEC (λ*c*. (*c*, []) ∈ *trail-pol* $\mathcal{A}_{in}$)›
if ‹*distinct-mset* $\mathcal{A}_{in}$› **and** *x*: ‹(*A*$_{in}$′, $\mathcal{A}_{in}$) ∈ *?arg*› **and**
‹*A*$_{in}$′ = (*x1*, *x2*)› **and** ‹*isasat-input-bounded* $\mathcal{A}_{in}$›
for $\mathcal{A}_{in}$ *A*$_{in}$′ *x1 x2*
unfolding *x P*
by (*rule tr*[*unfolded conc-fun-RETURN*])
(*use this* **in** ‹*auto simp*: *lits-with-max-rel-def dest*: *in-N0*›)

have *H*:
‹(*replicate* (2 ∗ *Suc* (*b*)) [], *empty-watched* $\mathcal{A}_{in}$)
∈ ⟨*Id*⟩*map-fun-rel* ((λ*L*. (*nat-of-lit L*, *L*)) ‘ *set-mset* ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$))›
if ‹(*x*, $\mathcal{A}_{in}$) ∈ *?arg*› **and**
‹*x* = (*a*, *b*)›
for $\mathcal{A}_{in}$ *x a b*
using *this* **unfolding** *map-fun-rel-def*
by (*auto simp*: *empty-watched-def* $\mathcal{L}_{all}$*-def*
lits-with-max-rel-def
intro!: *nth-replicate dest*!: *in-N0*
simp del: *replicate.simps*)
have [*simp*]: ‹(*x*, *y*) ∈ ⟨*Id*⟩*list-rel-mset-rel* ⟹ *L* ∈# *y* ⟹
L < *Suc* ((*Max* (*insert 0* (*set x*))))›
for *x y L*
by (*auto simp*: *list-rel-mset-rel-def br-def list-rel-def Id-def*
list-all2-op-eq-map-right-iff′ *list-mset-rel-def dest*: *in-N0*)

have *cach*: ‹*RETURN* (*replicate* (*Suc* (*b*)) *SEEN-UNKNOWN*, [])
  ≤ ⇓ *Id*
    (*SPEC* (*cach-refinement-empty y*))›
 if
  ‹$y = \mathcal{A}_{in} \wedge$ *distinct-mset* $\mathcal{A}_{in}$› **and**
  ‹(*x*, *y*) ∈ *?arg*› **and**
  ‹*x* = (*a*, *b*)›
 for *M W vm vma* φ *x y a b*
proof −
 show *?thesis*
  unfolding *cach-refinement-empty-def RETURN-RES-refine-iff*
   cach-refinement-alt-def Bex-def
  by (*rule exI*[*of* - ‹(*replicate* (*Suc* (*b*)) *SEEN-UNKNOWN*, [])›]) (*use that* **in**
    ‹*auto simp*: *map-fun-rel-def empty-watched-def* $\mathcal{L}_{all}$*-def*
      list-mset-rel-def lits-with-max-rel-def
     simp del: *replicate-Suc*
     dest!: *in-N0 intro*: ›)
qed
have *conflict*: ‹*RETURN* (*True*, *0*, *replicate* (*Suc* (*b*)) *NOTIN*)
  ≤ *SPEC* (λ*D*. (*D*, *None*) ∈ *option-lookup-clause-rel* $\mathcal{A}_{in}$)›
if
 ‹$y = \mathcal{A}_{in} \wedge$ *distinct-mset* $\mathcal{A}_{in}$ $\wedge$ *isasat-input-bounded* $\mathcal{A}_{in}$› **and**
 ‹((*a*, *b*), $\mathcal{A}_{in}$) ∈ *lits-with-max-rel O* ⟨*Id*⟩*mset-rel*› **and**
 ‹*x* = (*a*, *b*)›
for *a b x y*
proof −
 have ‹*L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$) ⟹
   L < *Suc* (*b*)› **for** *L*
  using *that in-N0* **by** (*auto simp*: *atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$
    lits-with-max-rel-def)
 then show *?thesis*
  by (*auto simp*: *option-lookup-clause-rel-def*
  lookup-clause-rel-def simp del: *replicate-Suc*
  intro: *mset-as-position.intros*)
qed
have *mark*: ‹*RETURN* (*0*, *replicate* (*Suc* (*b*)) *None*)
 ≤ *SPEC* (λ*D*. (*D*, {#}) ∈ *lookup-clause-rel* $\mathcal{A}_{in}$)›
 if
  ‹$y = \mathcal{A}_{in} \wedge$ *distinct-mset* $\mathcal{A}_{in}$ $\wedge$ *isasat-input-bounded* $\mathcal{A}_{in}$› **and**
  ‹((*a*, *b*), $\mathcal{A}_{in}$) ∈ *lits-with-max-rel O* ⟨*Id*⟩*mset-rel*› **and**
  ‹*x* = (*a*, *b*)›
 for *a b x y*
proof −
 have ‹*L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$) ⟹
   L < *Suc* (*b*)› **for** *L*
  using *that in-N0* **by** (*auto simp*: *atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$
   lits-with-max-rel-def)
 then show *?thesis*
  by (*auto simp*: *option-lookup-clause-rel-def*
   lookup-clause-rel-def simp del: *replicate-Suc*
   intro: *mset-as-position.intros*)
qed
have [*simp*]:
  ‹*NO-MATCH 0 a1* ⟹ *max 0* (*Max* (*insert a1* (*set a2*))) = *max a1* (*Max* (*insert 0* (*set a2*)))›
 for *a1* :: *nat* **and** *a2*

by (*metis* (*mono-tags*, *lifting*) *List.finite-set Max-insert all-not-in-conv finite-insert insertI1 insert-commute*)
have *le-uint32*: ‹$\forall$ L∈#$\mathcal{L}_{all}$ (*mset a*). *nat-of-lit* $L \leq$ *unat32-max* $\Longrightarrow$
Suc (*2* $*$ (*Max* (*insert 0* (*set a*)))) $\leq$ *unat32-max*› **for** *a*
apply (*induction a*)
apply (*auto simp*: *unat32-max-def*)
apply (*auto simp*: *max-def* $\mathcal{L}_{all}$*-add-mset*)
done
have *K*[*simp*]: ‹(*x*, $\mathcal{A}_{in}$) ∈ ⟨*Id*⟩*list-rel-mset-rel* $\Longrightarrow$
$L \in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}_{in}$) $\Longrightarrow$ $L <$ *Suc* ((*Max* (*insert 0* (*set x*))))›
for *x L* $\mathcal{A}_{in}$
unfolding *atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$
by (*auto simp*: *list-rel-mset-rel-def br-def list-rel-def Id-def*
list-all2-op-eq-map-right-iff' *list-mset-rel-def*)

show *?thesis*
apply (*intro frefI nres-relI*)
subgoal for *x y*
unfolding *init-state-wl-heur-alt-def init-state-wl-D'-def*
apply (*rewrite* **in** ‹*let* - = *Suc* -*in* -› *Let-def*)
apply (*rewrite* **in** ‹*let* - = *2* $*$ -*in* -› *Let-def*)
apply (*cases x*; *simp only*: *prod.case*)
apply (*refine-rcg init*[*of y x*] *initialize-Bump-Init*[**where** $\mathcal{A}$=$\mathcal{A}_{in}$, *THEN fref-to-Down-curry*, *of* -
‹*Suc* (*snd x*)›] *cach*)
subgoal for *a b* **by** (*auto simp*: *lits-with-max-rel-def intro*: *le-uint32*)
subgoal by (*auto intro*!: *simp*: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$
lits-with-max-rel-def atms-of-$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*rule conflict*)
subgoal by (*rule mark*)
subgoal by (*rule RETURN-rule*) (*rule H*; *simp only*:)
subgoal using *in-N0* **by** (*auto simp*: *lits-with-max-rel-def P-def*)
subgoal by *simp*
subgoal by (*auto simp*: *lits-with-max-rel-def*)
subgoal by (*auto simp*: *lits-with-max-rel-def*)
subgoal by (*auto simp*: *lits-with-max-rel-def*)
subgoal unfolding *phase-saving-def lits-with-max-rel-def* **by** (*auto intro*!: *K*)
subgoal by *fast*
subgoal by (*auto simp*: *lits-with-max-rel-def*)
apply *assumption*
apply (*rule refl*)
subgoal by (*auto simp*: *P-def init-rll-def option-lookup-clause-rel-def*
lookup-clause-rel-def lits-with-max-rel-def tuple15-rel-def
simp del: *replicate.simps*
intro!: *mset-as-position.intros K*)
done
done
qed

lemma *init-state-wl-D'*:
‹(*init-state-wl-D'*, *init-state-wl-heur*) ∈
[$\lambda\mathcal{A}_{in}$. *distinct-mset* $\mathcal{A}_{in}$ ∧ *isasat-input-bounded* $\mathcal{A}_{in}$]$_f$
lits-with-max-rel O ⟨*Id*⟩*mset-rel* →
⟨⟨*Id*, *Id*, *Id*, *nat-rel*, ⟨⟨*Id*⟩*list-rel*⟩*list-rel*,

Id, ⟨*bool-rel*⟩*list-rel*, *Id*, *Id*, *Id*, *Id*, *Id*, *Id*, *Id*⟩*tuple15-rel*⟩*nres-rel*›
apply −
apply (*intro frefI nres-relI*)
by (*rule init-state-wl-D0*[*THEN fref-to-Down*, *THEN order-trans*]) *auto*

lemma *init-state-wl-heur-init-state-wl'*:
‹(*init-state-wl-heur*, *RETURN o* (λ-. *init-state-wl*))
∈ [λ*N*. *N* = $\mathcal{A}_{in}$ ∧ *isasat-input-bounded* $\mathcal{A}_{in}$]$_f$ *Id* → ⟨*twl-st-heur-parsing-no-WL-wl* $\mathcal{A}_{in}$ *True*⟩*nres-rel*›
apply (*intro frefI nres-relI*)
unfolding *comp-def*
using *init-state-wl-heur-init-state-wl*[*THEN fref-to-Down*, *of* $\mathcal{A}_{in}$ ‹()› ‹()›]
by *auto*

lemma *all-blits-are-in-problem-init-blits-in*: ‹*all-blits-are-in-problem-init S* ⟹ *blits-in-*$\mathcal{L}_{in}$ *S*›
unfolding *blits-in-*$\mathcal{L}_{in}$*-def*
by (*cases S*)
(*auto simp*: *all-blits-are-in-problem-init.simps ac-simps*
$\mathcal{L}_{all}$*-atm-of-all-lits-of-mm all-lits-def all-lits-st-def*)

lemma *correct-watching-init-blits-in-*$\mathcal{L}_{in}$:
assumes ‹*correct-watching-init S*›
shows ‹*blits-in-*$\mathcal{L}_{in}$ *S*›
proof −
show *?thesis*
using *assms*
by (*cases S*)
(*auto simp*: *all-blits-are-in-problem-init-blits-in*
correct-watching-init.simps)
qed

fun *append-empty-watched* **where**
‹*append-empty-watched* ((*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*), *OC*) = ((*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, (λ-. [])), *OC*)›

fun *remove-watched* :: ‹'*v twl-st-wl-init-full* ⇒ '*v twl-st-wl-init*› **where**
‹*remove-watched* ((*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NNS*, *US*, *N0*, *U0*, *Q*, -), *OC*) = ((*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NNS*, *US*, *N0*, *U0*, *Q*), *OC*)›

definition *init-dt-wl'* :: ‹'*v clause-l list* ⇒ '*v twl-st-wl-init* ⇒ '*v twl-st-wl-init-full nres*› **where**
‹*init-dt-wl' CS S* = *do*{
S ← *init-dt-wl CS S*;
RETURN (*append-empty-watched S*)
}›

lemma *init-dt-wl'-spec*: ‹*init-dt-wl-pre CS S* ⟹ *init-dt-wl' CS S* ≤ ⇓
({(*S* :: '*v twl-st-wl-init-full*, *S'* :: '*v twl-st-wl-init*).
remove-watched S = *S'*}) (*SPEC* (*init-dt-wl-spec CS S*))›
unfolding *init-dt-wl'-def*
by (*refine-vcg bind-refine-spec*[*OF* - *init-dt-wl-init-dt-wl-spec*])
(*auto intro*!: *RETURN-RES-refine*)

lemma *init-dt-wl'-init-dt*:
‹*init-dt-wl-pre CS S* ⟹ (*S*, *S'*) ∈ *state-wl-l-init* ⟹
init-dt-wl' CS S ≤ ⇓

```
  ({(S :: 'v twl-st-wl-init-full, S' :: 'v twl-st-wl-init).
     remove-watched S = S'} O state-wl-l-init) (init-dt CS S')›
  unfolding init-dt-wl'-def
  apply (refine-vcg bind-refine[of - - - - - ‹RETURN›, OF init-dt-wl-init-dt, simplified])
  subgoal for S T
    by (cases S; cases T)
      auto
  done

definition isasat-init-fast-slow :: ‹twl-st-wl-heur-init ⇒ twl-st-wl-heur-init nres› where
  ‹isasat-init-fast-slow =
    (λS::twl-st-wl-heur-init. case S of Tuple15 M' N' D' j W' vm φ clvls cach lbd vdom failed x y z ⇒
      RETURN (Tuple15 (trail-pol-slow-of-fast M') N' D' j (convert-wlists-to-nat-conv W') vm φ
        clvls cach lbd vdom failed x y z))›

lemma isasat-init-fast-slow-alt-def:
  ‹isasat-init-fast-slow S = RETURN S›
  unfolding isasat-init-fast-slow-def trail-pol-slow-of-fast-alt-def
    convert-wlists-to-nat-conv-def
  by (cases S) auto

end
theory Tuple15-LLVM
  imports Tuple15 IsaSAT-Literals-LLVM
begin

hide-const (open) NEMonad.ASSERT NEMonad.RETURN
```

This is the setup for accessing and modifying the state as an abstract tuple of 15 elements. The construction is kept generic (even if still targetting only our state). There is a lot of copy-paste that would be nice to automate at some point.

We define 3 sort of operations:

1. extracting an element, replacing it by an default element. Modifies the state. The name starts with *exctr*
2. reinserting an element, freeing the current one. Modifies the state. The name starts with *update*
3. in-place reading a value, possibly with pure parameters. Does not modify the state. The name starts with *read*

```
instantiation tuple15 ::
  (llvm-rep,llvm-rep,llvm-rep,llvm-rep,
  llvm-rep,llvm-rep,llvm-rep,llvm-rep,
  llvm-rep,llvm-rep,llvm-rep,llvm-rep,
  llvm-rep,llvm-rep,llvm-rep) llvm-rep
begin
  definition to-val-tuple15 where
    ‹to-val-tuple15 ≡ (λS. case S of
     Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts ⇒ LL-STRUCT [to-val
M, to-val N, to-val D, to-val i, to-val W, to-val ivmtf,
       to-val icount, to-val ccach, to-val lbd,
       to-val outl, to-val stats, to-val heur, to-val aivdom,  to-val clss, to-val opts])›
```

definition *from-val-tuple15* :: ‹*llvm-val* ⇒ ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*) *tuple15*›
where
‹*from-val-tuple15* ≡ (λ*p*. *case llvm-val.the-fields p of*
[*M*, *N*, *D*, *i*, *W*, *ivmtf*, *icount*, *ccach*, *lbd*, *outl*, *stats*, *heur*, *aivdom*, *clss*, *opts*] ⇒
Tuple15 (*from-val M*) (*from-val N*) (*from-val D*) (*from-val i*) (*from-val W*) (*from-val ivmtf*) (*from-val icount*) (*from-val ccach*) (*from-val lbd*)
(*from-val outl*) (*from-val stats*) (*from-val heur*) (*from-val aivdom*) (*from-val clss*) (*from-val opts*))›

definition [*simp*]: *struct-of-tuple15* (- :: ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*) *tuple15 itself*) ≡
VS-STRUCT [*struct-of TYPE*('*a*), *struct-of TYPE*('*b*), *struct-of TYPE*('*c*),
struct-of TYPE('*d*), *struct-of TYPE*('*e*), *struct-of TYPE*('*f*), *struct-of TYPE*('*g*), *struct-of TYPE*('*h*),
struct-of TYPE('*i*), *struct-of TYPE*('*j*), *struct-of TYPE*('*k*), *struct-of TYPE*('*l*),
struct-of TYPE('*m*), *struct-of TYPE*('*n*), *struct-of TYPE*('*o*)]

definition [*simp*]: *init-tuple15* :: ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*) *tuple15* ≡ *Tuple15 init init init init init init init init init init init init init init init*

instance
apply *standard*
unfolding *from-val-tuple15-def to-val-tuple15-def struct-of-tuple15-def init-tuple15-def comp-def tuple15.case-distrib*
subgoal
by (*auto simp*: *init-zero fun-eq-iff from-val-tuple15-def split*: *tuple15.splits*)
subgoal for *v*
by (*cases v*) (*auto split*: *list.splits tuple15.splits*)
subgoal for *v*
by (*cases v*)
(*simp add*: *LLVM-Shallow.null-def to-val-ptr-def split*: *tuple15.splits*)
subgoal
by (*simp add*: *LLVM-Shallow.null-def to-val-ptr-def to-val-word-def init-zero split*: *tuple15.splits*)
done
end

Setup for LLVM code export

Declare structure to code generator.

lemma *to-val-tuple17*[*ll-struct-of*]: *struct-of TYPE*(('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*, '*k*, '*l*, '*m*, '*n*, '*o*) *tuple15*) = *VS-STRUCT* [
struct-of TYPE('*a*::*llvm-rep*),
struct-of TYPE('*b*::*llvm-rep*),
struct-of TYPE('*c*::*llvm-rep*),
struct-of TYPE('*d*::*llvm-rep*),
struct-of TYPE('*e*::*llvm-rep*),
struct-of TYPE('*f*::*llvm-rep*),
struct-of TYPE('*g*::*llvm-rep*),
struct-of TYPE('*h*::*llvm-rep*),
struct-of TYPE('*i*::*llvm-rep*),
struct-of TYPE('*j*::*llvm-rep*),
struct-of TYPE('*k*::*llvm-rep*),
struct-of TYPE('*l*::*llvm-rep*),
struct-of TYPE('*m*::*llvm-rep*),
struct-of TYPE('*n*::*llvm-rep*),
struct-of TYPE('*o*::*llvm-rep*)]
by (*auto*)

lemma *node-insert-value*:
ll-insert-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *M′ 0* = *Mreturn* (*Tuple15 M′ N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*)
ll-insert-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *N′* (*Suc 0*) = *Mreturn* (*Tuple15 M N′ D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*)
ll-insert-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *D′ 2* = *Mreturn* (*Tuple15 M N D′ i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*)
ll-insert-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *i′ 3* = *Mreturn* (*Tuple15 M N D i′ W ivmtf icount ccach lbd outl stats heur aivdom clss opts*)
ll-insert-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *W′ 4* = *Mreturn* (*Tuple15 M N D i W′ ivmtf icount ccach lbd outl stats heur aivdom clss opts*)
ll-insert-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *ivmtf′ 5* = *Mreturn* (*Tuple15 M N D i W ivmtf′ icount ccach lbd outl stats heur aivdom clss opts*)
ll-insert-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *icount′ 6* = *Mreturn* (*Tuple15 M N D i W ivmtf icount′ ccach lbd outl stats heur aivdom clss opts*)
ll-insert-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *ccach′ 7* = *Mreturn* (*Tuple15 M N D i W ivmtf icount ccach′ lbd outl stats heur aivdom clss opts*)
ll-insert-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *lbd′ 8* = *Mreturn* (*Tuple15 M N D i W ivmtf icount ccach lbd′ outl stats heur aivdom clss opts*)
ll-insert-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *outl′ 9* = *Mreturn* (*Tuple15 M N D i W ivmtf icount ccach lbd outl′ stats heur aivdom clss opts*)
ll-insert-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *stats′ 10* = *Mreturn* (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats′ heur aivdom clss opts*)
ll-insert-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *heur′ 11* = *Mreturn* (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur′ aivdom clss opts*)
ll-insert-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *aivdom′ 12* = *Mreturn* (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom′ clss opts*)
ll-insert-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *clss′ 13* = *Mreturn* (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss′ opts*)
ll-insert-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *opts′ 14* = *Mreturn* (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts′*)
by (*simp-all add*: *ll-insert-value-def llvm-insert-value-def Let-def checked-from-val-def to-val-tuple15-def from-val-tuple15-def*)

lemma *node-extract-value*:
ll-extract-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *0* = *Mreturn M*
ll-extract-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) (*Suc 0*) = *Mreturn N*
ll-extract-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *2* = *Mreturn D*
ll-extract-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *3* = *Mreturn i*
ll-extract-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *4* = *Mreturn W*
ll-extract-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *5* = *Mreturn ivmtf*
ll-extract-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *6* = *Mreturn icount*
ll-extract-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *7* = *Mreturn ccach*
ll-extract-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *8* = *Mreturn lbd*
ll-extract-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *9* = *Mreturn outl*

ll-extract-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *10* = *Mreturn stats*
ll-extract-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *11* = *Mreturn heur*
ll-extract-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *12* = *Mreturn aivdom*
ll-extract-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *13* = *Mreturn clss*
ll-extract-value (*Tuple15 M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts*) *14* = *Mreturn opts*
apply (*simp-all add*: *ll-extract-value-def llvm-extract-value-def Let-def checked-from-val-def to-val-tuple15-def from-val-tuple15-def*)
done

Lemmas to translate node construction and destruction

lemma *inline-return-node*[*llvm-pre-simp*]: *Mreturn* (*Tuple15 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*) = *doM* {
r ← *ll-insert-value init M 0*;
r ← *ll-insert-value r N 1*;
r ← *ll-insert-value r D 2*;
r ← *ll-insert-value r i 3*;
r ← *ll-insert-value r W 4*;
r ← *ll-insert-value r ivmtf 5*;
r ← *ll-insert-value r icount 6*;
r ← *ll-insert-value r ccach 7*;
r ← *ll-insert-value r lbd 8*;
r ← *ll-insert-value r outl 9*;
r ← *ll-insert-value r heur 10*;
r ← *ll-insert-value r stats 11*;
r ← *ll-insert-value r aivdom 12*;
r ← *ll-insert-value r clss 13*;
r ← *ll-insert-value r opts 14*;
Mreturn r
}
apply (*auto simp*: *node-insert-value*)
done

lemma *inline-node-case*[*llvm-pre-simp*]: (*case r of* (*Tuple15 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*) ⇒ *f M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*) = *doM* {
M ← *ll-extract-value r 0*;
N ← *ll-extract-value r 1*;
D ← *ll-extract-value r 2*;
i ← *ll-extract-value r 3*;
W ← *ll-extract-value r 4*;
ivmtf ← *ll-extract-value r 5*;
icount ← *ll-extract-value r 6*;
ccach ← *ll-extract-value r 7*;
lbd ← *ll-extract-value r 8*;
outl ← *ll-extract-value r 9*;
heur ← *ll-extract-value r 10*;
stats ← *ll-extract-value r 11*;
aivdom ← *ll-extract-value r 12*;
clss ← *ll-extract-value r 13*;
opts ← *ll-extract-value r 14*;
f M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts

}
apply (*cases r*)
apply (*auto simp*: *node-extract-value*)
done

lemma *inline-return-node-case*[*llvm-pre-simp*]: *doM* {*Mreturn* (*case r of* (*Tuple15 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*) ⇒ *f M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*)} = *doM* {
M ← *ll-extract-value r 0*;
N ← *ll-extract-value r 1*;
D ← *ll-extract-value r 2*;
i ← *ll-extract-value r 3*;
W ← *ll-extract-value r 4*;
ivmtf ← *ll-extract-value r 5*;
icount ← *ll-extract-value r 6*;
ccach ← *ll-extract-value r 7*;
lbd ← *ll-extract-value r 8*;
outl ← *ll-extract-value r 9*;
heur ← *ll-extract-value r 10*;
stats ← *ll-extract-value r 11*;
aivdom ← *ll-extract-value r 12*;
clss ← *ll-extract-value r 13*;
opts ← *ll-extract-value r 14*;
Mreturn (*f M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*)
}
apply (*cases r*)
apply (*auto simp*: *node-extract-value*)
done
lemma *inline-direct-return-node-case*[*llvm-pre-simp*]: *doM* {(*case r of* (*Tuple15 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*) ⇒ *f M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*)} = *doM* {
M ← *ll-extract-value r 0*;
N ← *ll-extract-value r 1*;
D ← *ll-extract-value r 2*;
i ← *ll-extract-value r 3*;
W ← *ll-extract-value r 4*;
ivmtf ← *ll-extract-value r 5*;
icount ← *ll-extract-value r 6*;
ccach ← *ll-extract-value r 7*;
lbd ← *ll-extract-value r 8*;
outl ← *ll-extract-value r 9*;
heur ← *ll-extract-value r 10*;
stats ← *ll-extract-value r 11*;
aivdom ← *ll-extract-value r 12*;
clss ← *ll-extract-value r 13*;
opts ← *ll-extract-value r 14*;
(*f M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*)
}
apply (*cases r*)
apply (*auto simp*: *node-extract-value*)
done

lemmas [*llvm-inline*] =
tuple15.Tuple15-a-def
tuple15.Tuple15-b-def
tuple15.Tuple15-c-def

tuple15.Tuple15-d-def
tuple15.Tuple15-e-def
tuple15.Tuple15-f-def
tuple15.Tuple15-g-def
tuple15.Tuple15-h-def
tuple15.Tuple15-i-def
tuple15.Tuple15-j-def
tuple15.Tuple15-k-def
tuple15.Tuple15-l-def
tuple15.Tuple15-m-def
tuple15.Tuple15-n-def
tuple15.Tuple15-o-def

fun *tuple15-assn* :: ‹
('a ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('b ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('c ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('d ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('e ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('f ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('g ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('h ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('i ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('j ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('k ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('l ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('m ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('n ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('o ⇒ - ⇒ *llvm-amemory* ⇒ *bool*) ⇒
('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o) *tuple15* ⇒ - ⇒ *assn*› **where**
‹*tuple15-assn a-assn b-assn' c-assn d-assn e-assn f-assn g-assn h-assn i-assn j-assn k-assn l-assn m-assn n-assn o-assn S T* =
(*case* (*S*, *T*) *of*
(*Tuple15 M N D i W ivmtf icount ccach lbd outl heur stats aivdom clss opts*,
Tuple15 M' N' D' i' W' ivmtf' icount' ccach' lbd' outl' heur' stats' aivdom' clss' opts')
⇒
(*a-assn M* (*M'*) ∧* *b-assn' N* (*N'*) ∧* *c-assn D* (*D'*) ∧* *d-assn i* (*i'*) ∧*
e-assn W (*W'*) ∧* *f-assn ivmtf* (*ivmtf'*) ∧* *g-assn icount* (*icount'*) ∧* *h-assn ccach* (*ccach'*) ∧*
i-assn lbd (*lbd'*) ∧* *j-assn outl* (*outl'*) ∧* *k-assn heur* (*heur'*) ∧* *l-assn stats* (*stats'*) ∧*
m-assn aivdom (*aivdom'*) ∧* *n-assn clss* (*clss'*) ∧* *o-assn opts* (*opts'*)))
›

locale *tuple15-state-ops* =
fixes
a-assn :: ‹'a ⇒ 'xa ⇒ *assn*› **and**
b-assn :: ‹'b ⇒ 'xb ⇒ *assn*› **and**
c-assn :: ‹'c ⇒ 'xc ⇒ *assn*› **and**
d-assn :: ‹'d ⇒ 'xd ⇒ *assn*› **and**
e-assn :: ‹'e ⇒ 'xe ⇒ *assn*› **and**
f-assn :: ‹'f ⇒ 'xf ⇒ *assn*› **and**
g-assn :: ‹'g ⇒ 'xg ⇒ *assn*› **and**
h-assn :: ‹'h ⇒ 'xh ⇒ *assn*› **and**
i-assn :: ‹'i ⇒ 'xi ⇒ *assn*› **and**
j-assn :: ‹'j ⇒ 'xj ⇒ *assn*› **and**
k-assn :: ‹'k ⇒ 'xk ⇒ *assn*› **and**

l-assn :: ‹*'l* ⇒ *'xl* ⇒ *assn*› **and**
m-assn :: ‹*'m* ⇒ *'xm* ⇒ *assn*› **and**
n-assn :: ‹*'n* ⇒ *'xn* ⇒ *assn*› **and**
o-assn :: ‹*'o* ⇒ *'xo* ⇒ *assn*› **and**
a-default :: *'a* **and**
a :: ‹*'xa llM*› **and**
b-default :: *'b* **and**
b :: ‹*'xb llM*› **and**
c-default :: *'c* **and**
c :: ‹*'xc llM*› **and**
d-default :: *'d* **and**
d :: ‹*'xd llM*› **and**
e-default :: *'e* **and**
e :: ‹*'xe llM*› **and**
f-default :: *'f* **and**
f :: ‹*'xf llM*› **and**
g-default :: *'g* **and**
g :: ‹*'xg llM*› **and**
h-default :: *'h* **and**
h :: ‹*'xh llM*› **and**
i-default :: *'i* **and**
i :: ‹*'xi llM*› **and**
j-default :: *'j* **and**
j :: ‹*'xj llM*› **and**
k-default :: *'k* **and**
k :: ‹*'xk llM*› **and**
l-default :: *'l* **and**
l :: ‹*'xl llM*› **and**
m-default :: *'m* **and**
m :: ‹*'xm llM*› **and**
n-default :: *'n* **and**
n :: ‹*'xn llM*› **and**
ko-default :: *'o* **and**
ko :: ‹*'xo llM*›
begin

definition *tuple15-int-assn* :: ‹- ⇒ - ⇒ *assn*› **where**
‹*tuple15-int-assn* = *tuple15-assn*
a-assn b-assn c-assn d-assn
e-assn f-assn g-assn h-assn
i-assn j-assn k-assn l-assn
m-assn n-assn o-assn›

definition *remove-a* :: ‹(*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*) *tuple15* ⇒ *'a* × (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*) *tuple15*› **where**
‹*remove-a tuple15* = (*case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15* ⇒
(*x1*, *Tuple15 a-default x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15*))›

definition *remove-b* :: ‹(*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*) *tuple15* ⇒ *'b* × (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*) *tuple15*› **where**
‹*remove-b tuple15* = (*case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15* ⇒
(*x2*, *Tuple15 x1 b-default x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15*))›

definition *remove-c* :: ‹(*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,

'k, 'l, 'm, 'n, 'o) tuple15 ⇒ - × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o) tuple15› **where**
‹remove-c tuple15 = (case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 ⇒
(x3, Tuple15 x1 x2 c-default x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15))›

definition remove-d :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o) tuple15 ⇒ - × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o) tuple15› **where**
‹remove-d tuple15 = (case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 ⇒
(x4, Tuple15 x1 x2 x3 d-default x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15))›

definition remove-e :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o) tuple15 ⇒ 'e × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o) tuple15› **where**
‹remove-e tuple15 = (case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 ⇒
(x5, Tuple15 x1 x2 x3 x4 e-default x6 x7 x8 x9 x10 x11 x12 x13 x14 x15))›

definition remove-f :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o) tuple15 ⇒ 'f × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o) tuple15› **where**
‹remove-f tuple15 = (case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 ⇒
(x6, Tuple15 x1 x2 x3 x4 x5 f-default x7 x8 x9 x10 x11 x12 x13 x14 x15))›

definition remove-g :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o) tuple15 ⇒ 'g × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o) tuple15› **where**
‹remove-g tuple15 = (case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 ⇒
(x7, Tuple15 x1 x2 x3 x4 x5 x6 g-default x8 x9 x10 x11 x12 x13 x14 x15))›

definition remove-h :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o) tuple15 ⇒ 'h × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o) tuple15› **where**
‹remove-h tuple15 = (case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 ⇒
(x8, Tuple15 x1 x2 x3 x4 x5 x6 x7 h-default x9 x10 x11 x12 x13 x14 x15))›

definition remove-i :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o) tuple15 ⇒ 'i × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o) tuple15› **where**
‹remove-i tuple15 = (case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 ⇒
(x9, Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 i-default x10 x11 x12 x13 x14 x15))›

definition remove-j :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o) tuple15 ⇒ 'j × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o) tuple15› **where**
‹remove-j tuple15 = (case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 ⇒
(x10, Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 j-default x11 x12 x13 x14 x15))›

definition remove-k :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o) tuple15 ⇒ 'k × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o) tuple15› **where**
‹remove-k tuple15 = (case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 ⇒
(x11, Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 k-default x12 x13 x14 x15))›

definition remove-l :: ‹('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o) tuple15 ⇒ 'l × ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
'k, 'l, 'm, 'n, 'o) tuple15› **where**

‹*remove-l tuple15* = (*case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15* ⇒
(*x12*, *Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 l-default x13 x14 x15*))›

definition *remove-m* :: ‹('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*,
'*k*, '*l*, '*m*, '*n*, '*o*) *tuple15* ⇒ '*m* × ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*,
'*k*, '*l*, '*m*, '*n*, '*o*) *tuple15*› **where**
‹*remove-m tuple15* = (*case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15* ⇒
(*x13*, *Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 m-default x14 x15*))›

definition *remove-n* :: ‹('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*,
'*k*, '*l*, '*m*, '*n*, '*o*) *tuple15* ⇒ '*n* × ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*,
'*k*, '*l*, '*m*, '*n*, '*o*) *tuple15*› **where**
‹*remove-n tuple15* = (*case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15* ⇒
(*x14*, *Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 n-default x15*))›

definition *remove-o* :: ‹('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*,
'*k*, '*l*, '*m*, '*n*, '*o*) *tuple15* ⇒ '*o* × ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*,
'*k*, '*l*, '*m*, '*n*, '*o*) *tuple15*› **where**
‹*remove-o tuple15* = (*case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15* ⇒
(*x15*, *Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 ko-default*))›

definition *update-a* :: ‹'*a* ⇒ ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*,
'*k*, '*l*, '*m*, '*n*, '*o*) *tuple15* ⇒ ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*,
'*k*, '*l*, '*m*, '*n*, '*o*) *tuple15*› **where**
‹*update-a x1 tuple15* = (*case tuple15 of Tuple15 M x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15*
⇒
let - = M in
Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15)›

definition *update-b* :: ‹'*b* ⇒ ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*,
'*k*, '*l*, '*m*, '*n*, '*o*) *tuple15* ⇒ ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*,
'*k*, '*l*, '*m*, '*n*, '*o*) *tuple15*› **where**
‹*update-b x2 tuple15* = (*case tuple15 of Tuple15 x1 M x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15*
⇒
let - = M in
Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15)›

definition *update-c* :: ‹'*c* ⇒ ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*,
'*k*, '*l*, '*m*, '*n*, '*o*) *tuple15* ⇒ ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*,
'*k*, '*l*, '*m*, '*n*, '*o*) *tuple15*› **where**
‹*update-c x3 tuple15* = (*case tuple15 of Tuple15 x1 x2 M x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15*
⇒
let - = M in
Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15)›

definition *update-d* :: ‹'*d* ⇒ ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*,
'*k*, '*l*, '*m*, '*n*, '*o*) *tuple15* ⇒ ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*,
'*k*, '*l*, '*m*, '*n*, '*o*) *tuple15*› **where**
‹*update-d x4 tuple15* = (*case tuple15 of Tuple15 x1 x2 x3 M x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15*
⇒
let - = M in
Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15)›

definition *update-e* :: ‹'*e* ⇒ ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*,
'*k*, '*l*, '*m*, '*n*, '*o*) *tuple15* ⇒ ('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*,
'*k*, '*l*, '*m*, '*n*, '*o*) *tuple15*› **where**

‹update-e x5 tuple15 = (case tuple15 of Tuple15 x1 x2 x3 x4 M x6 x7 x8 x9 x10 x11 x12 x13 x14 x15
⇒
  let - = M in
  Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15)›

definition update-f :: ‹'f ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
  'k, 'l, 'm, 'n, 'o) tuple15 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
  'k, 'l, 'm, 'n, 'o) tuple15› **where**
‹update-f x6 tuple15 = (case tuple15 of Tuple15 x1 x2 x3 x4 x5 M x7 x8 x9 x10 x11 x12 x13 x14 x15 ⇒
  let - = M in
  Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15)›

definition update-g :: ‹'g ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
  'k, 'l, 'm, 'n, 'o) tuple15 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
  'k, 'l, 'm, 'n, 'o) tuple15› **where**
‹update-g x7 tuple15 = (case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 M x8 x9 x10 x11 x12 x13 x14 x15
⇒
  let - = M in
  Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15)›

definition update-h :: ‹'h ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
  'k, 'l, 'm, 'n, 'o) tuple15 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
  'k, 'l, 'm, 'n, 'o) tuple15› **where**
‹update-h x8 tuple15 = (case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 x7 M x9 x10 x11 x12 x13 x14 x15
⇒
  let - = M in
  Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15)›

definition update-i :: ‹'i ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
  'k, 'l, 'm, 'n, 'o) tuple15 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
  'k, 'l, 'm, 'n, 'o) tuple15› **where**
‹update-i x9 tuple15 = (case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 M x10 x11 x12 x13 x14 x15 ⇒
  let - = M in
  Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15)›

definition update-j :: ‹'j ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
  'k, 'l, 'm, 'n, 'o) tuple15 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
  'k, 'l, 'm, 'n, 'o) tuple15› **where**
‹update-j x10 tuple15 = (case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 M x11 x12 x13 x14 x15 ⇒
  let - = M in
  Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15)›

definition update-k :: ‹'k ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
  'k, 'l, 'm, 'n, 'o) tuple15 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
  'k, 'l, 'm, 'n, 'o) tuple15› **where**
‹update-k x11 tuple15 = (case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 M x12 x13 x14 x15
⇒
  let - = M in
  Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15)›

definition update-l :: ‹'l ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
  'k, 'l, 'm, 'n, 'o) tuple15 ⇒ ('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,
  'k, 'l, 'm, 'n, 'o) tuple15› **where**
‹update-l x12 tuple15 = (case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 M x13 x14 x15 ⇒
  let - = M in
  Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15)›

definition *update-m* :: ‹*'m* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*) *tuple15* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*) *tuple15*› **where**
‹*update-m x13 tuple15* = (*case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 M x14 x15*
⇒
let - = M in
Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15)›

definition *update-n* :: ‹*'n* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*) *tuple15* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*) *tuple15*› **where**
‹*update-n x14 tuple15* = (*case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 M x15*
⇒
let - = M in
Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15)›

definition *update-o* :: ‹*'o* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*) *tuple15* ⇒ (*'a*, *'b*, *'c*, *'d*, *'e*, *'f*, *'g*, *'h*, *'i*, *'j*,
'k, *'l*, *'m*, *'n*, *'o*) *tuple15*› **where**
‹*update-o x15 tuple15* = (*case tuple15 of Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 M*
⇒
let - = M in
Tuple15 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15)›

end

lemma *tuple15-assn-conv*[*simp*]:
tuple15-assn P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13 P14 P15 (*Tuple15 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15*)
(*Tuple15 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15'*) =
(*P1 a1 a1'* ∧∗
P2 a2 a2' ∧∗
P3 a3 a3' ∧∗
P4 a4 a4' ∧∗
P5 a5 a5' ∧∗
P6 a6 a6' ∧∗
P7 a7 a7' ∧∗
P8 a8 a8' ∧∗ *P9 a9 a9'* ∧∗ *P10 a10 a10'* ∧∗ *P11 a11 a11'* ∧∗ *P12 a12 a12'* ∧∗ *P13 a13 a13'* ∧∗ *P14 a14 a14'* ∧∗ *P15 a15 a15'*)
unfolding *tuple15-assn.simps* **by** *auto*
lemma *tuple15-assn-ctxt*:
‹*tuple15-assn P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13 P14 P15* (*Tuple15 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15*)
(*Tuple15 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15'*) = *z* ⟹
hn-ctxt (*tuple15-assn P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13 P14 P15*) (*Tuple15 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15*)
(*Tuple15 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15'*)= *z*›
by (*simp add*: *hn-ctxt-def*)

lemma *hn-case-tuple15'*[*sepref-comb-rules*]:
assumes *FR*: ‹Γ ⊢ *hn-ctxt* (*tuple15-assn P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13 P14 P15*) *p' p* ∗∗ Γ*1*›
assumes *Pair*: ⋀*a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15'*.

⟦p'=Tuple15 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15'⟧
⟹ hn-refine (hn-ctxt P1 a1' a1 ∧* hn-ctxt P2 a2' a2 ∧* hn-ctxt P3 a3' a3 ∧* hn-ctxt P4 a4' a4 ∧*
hn-ctxt P5 a5' a5 ∧* hn-ctxt P6 a6' a6 ∧* hn-ctxt P7 a7' a7 ∧* hn-ctxt P8 a8' a8 ∧*
hn-ctxt P9 a9' a9 ∧* hn-ctxt P10 a10' a10 ∧* hn-ctxt P11 a11' a11 ∧* hn-ctxt P12 a12' a12 ∧*
hn-ctxt P13 a13' a13 ∧* hn-ctxt P14 a14' a14 ∧* hn-ctxt P15 a15' a15 ∧* Γ1)
(f a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15)
(Γ2 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15') R
(CP a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15)
(f' a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15')
assumes FR2: ‹⋀a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15'.
Γ2 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a1' a2' a3' a4' a5' a6' a7' a8' a9' a10' a11' a12' a13' a14' a15' ⊢
hn-ctxt P1' a1' a1 ** hn-ctxt P2' a2' a2 ** hn-ctxt P3' a3' a3 ** hn-ctxt P4' a4' a4 **
hn-ctxt P5' a5' a5 ** hn-ctxt P6' a6' a6 ** hn-ctxt P7' a7' a7 ** hn-ctxt P8' a8' a8 **
hn-ctxt P9' a9' a9 ** hn-ctxt P10' a10' a10 ** hn-ctxt P11' a11' a11 ** hn-ctxt P12' a12' a12 **
hn-ctxt P13' a13' a13 ** hn-ctxt P14' a14' a14 ** hn-ctxt P15' a15' a15 ** Γ1'›
shows ‹hn-refine Γ (case-tuple15 f p) (hn-ctxt (tuple15-assn P1' P2' P3' P4' P5' P6' P7' P8' P9' P10' P11' P12' P13' P14' P15') p' p ** Γ1')
R (case-tuple15 CP p) (case-tuple15$(λ2a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15. f' a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15)$p')› (**is** ‹?G Γ›)
unfolding autoref-tag-defs PROTECT2-def
apply1 (rule hn-refine-cons-pre[OF FR])
apply1 (cases p; cases p'; simp add: tuple15-assn-conv[THEN tuple15-assn-ctxt])
unfolding CP-SPLIT-def prod.simps
apply (rule hn-refine-cons[OF - Pair - entails-refl])
applyS (simp add: hn-ctxt-def)
applyS simp **using** FR2
by (simp add: hn-ctxt-def)

lemma case-tuple15-arity[sepref-monadify-arity]:
‹case-tuple15 ≡ λ2fp p. SP case-tuple15$(λ2a b. fp$a$b)$p›
by (simp-all only: SP-def APP-def PROTECT2-def RCALL-def)

lemma case-tuple15-comb[sepref-monadify-comb]:
‹⋀fp p. case-tuple15fpp ≡ Refine-Basic.bind$(EVAL$p)$(λ2p. (SP case-tuple15$fp$p))›
by (simp-all)

lemma case-tuple15-plain-comb[sepref-monadify-comb]:
EVAL$(case-tuple15$(λ2a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15. fp a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15)$p) ≡
Refine-Basic.bind$(EVAL$p)$(λ2p. case-tuple15$(λ2a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15. EVAL$(fp a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15))$p)
apply (rule eq-reflection, simp split: list.split prod.split option.split tuple15.split)+
done

lemma ho-tuple15-move[sepref-preproc]: ‹case-tuple15 (λa1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 x. f x a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15) =
(λp x. case-tuple15 (f x) p)›
by (auto split: tuple15.splits)

locale tuple15-state =

tuple15-state-ops a-assn b-assn c-assn d-assn e-assn
f-assn g-assn h-assn i-assn j-assn
k-assn l-assn m-assn n-assn o-assn
a-default a
b-default b
c-default c
d-default d
e-default e
f-default f
g-default g
h-default h
i-default i
j-default j
k-default k
l-default l
m-default m
n-default n
ko-default ko
for
a-assn :: ‹*'a* ⇒ *'xa* ⇒ *assn*› **and**
b-assn :: ‹*'b* ⇒ *'xb* ⇒ *assn*› **and**
c-assn :: ‹*'c* ⇒ *'xc* ⇒ *assn*› **and**
d-assn :: ‹*'d* ⇒ *'xd* ⇒ *assn*› **and**
e-assn :: ‹*'e* ⇒ *'xe* ⇒ *assn*› **and**
f-assn :: ‹*'f* ⇒ *'xf* ⇒ *assn*› **and**
g-assn :: ‹*'g* ⇒ *'xg* ⇒ *assn*› **and**
h-assn :: ‹*'h* ⇒ *'xh* ⇒ *assn*› **and**
i-assn :: ‹*'i* ⇒ *'xi* ⇒ *assn*› **and**
j-assn :: ‹*'j* ⇒ *'xj* ⇒ *assn*› **and**
k-assn :: ‹*'k* ⇒ *'xk* ⇒ *assn*› **and**
l-assn :: ‹*'l* ⇒ *'xl* ⇒ *assn*› **and**
m-assn :: ‹*'m* ⇒ *'xm* ⇒ *assn*› **and**
n-assn :: ‹*'n* ⇒ *'xn* ⇒ *assn*› **and**
o-assn :: ‹*'o* ⇒ *'xo* ⇒ *assn*› **and**
a-default :: *'a* **and**
a :: ‹*'xa llM*› **and**
b-default :: *'b* **and**
b :: ‹*'xb llM*› **and**
c-default :: *'c* **and**
c :: ‹*'xc llM*› **and**
d-default :: *'d* **and**
d :: ‹*'xd llM*› **and**
e-default :: *'e* **and**
e :: ‹*'xe llM*› **and**
f-default :: *'f* **and**
f :: ‹*'xf llM*› **and**
g-default :: *'g* **and**
g :: ‹*'xg llM*› **and**
h-default :: *'h* **and**
h :: ‹*'xh llM*› **and**
i-default :: *'i* **and**
i :: ‹*'xi llM*› **and**
j-default :: *'j* **and**
j :: ‹*'xj llM*› **and**
k-default :: *'k* **and**
k :: ‹*'xk llM*› **and**

l-default :: *'l* **and**
l :: ‹*'xl llM*› **and**
m-default :: *'m* **and**
m :: ‹*'xm llM*› **and**
n-default :: *'n* **and**
n :: ‹*'xn llM*› **and**
ko-default :: *'o* **and**
ko :: ‹*'xo llM*› **and**
a-free :: ‹*'xa* ⇒ *unit llM*› **and**
b-free :: ‹*'xb* ⇒ *unit llM*› **and**
c-free :: ‹*'xc* ⇒ *unit llM*› **and**
d-free :: ‹*'xd* ⇒ *unit llM*› **and**
e-free :: ‹*'xe* ⇒ *unit llM*› **and**
f-free :: ‹*'xf* ⇒ *unit llM*› **and**
g-free :: ‹*'xg* ⇒ *unit llM*› **and**
h-free :: ‹*'xh* ⇒ *unit llM*› **and**
i-free :: ‹*'xi* ⇒ *unit llM*› **and**
j-free :: ‹*'xj* ⇒ *unit llM*› **and**
k-free :: ‹*'xk* ⇒ *unit llM*› **and**
l-free :: ‹*'xl* ⇒ *unit llM*› **and**
m-free :: ‹*'xm* ⇒ *unit llM*› **and**
n-free :: ‹*'xn* ⇒ *unit llM*› **and**
o-free :: ‹*'xo* ⇒ *unit llM*› +
assumes
a: ‹(*uncurry0 a*, *uncurry0* (*RETURN a-default*)) ∈ *unit-assn*^{k} →$_a$ *a-assn*› **and**
b: ‹(*uncurry0 b*, *uncurry0* (*RETURN b-default*)) ∈ *unit-assn*^{k} →$_a$ *b-assn*› **and**
c: ‹(*uncurry0 c*, *uncurry0* (*RETURN c-default*)) ∈ *unit-assn*^{k} →$_a$ *c-assn*› **and**
d: ‹(*uncurry0 d*, *uncurry0* (*RETURN d-default*)) ∈ *unit-assn*^{k} →$_a$ *d-assn*› **and**
e: ‹(*uncurry0 e*, *uncurry0* (*RETURN e-default*)) ∈ *unit-assn*^{k} →$_a$ *e-assn*› **and**
f: ‹(*uncurry0 f*, *uncurry0* (*RETURN f-default*)) ∈ *unit-assn*^{k} →$_a$ *f-assn*› **and**
g: ‹(*uncurry0 g*, *uncurry0* (*RETURN g-default*)) ∈ *unit-assn*^{k} →$_a$ *g-assn*› **and**
h: ‹(*uncurry0 h*, *uncurry0* (*RETURN h-default*)) ∈ *unit-assn*^{k} →$_a$ *h-assn*› **and**
i: ‹(*uncurry0 i*, *uncurry0* (*RETURN i-default*)) ∈ *unit-assn*^{k} →$_a$ *i-assn*› **and**
j: ‹(*uncurry0 j*, *uncurry0* (*RETURN j-default*)) ∈ *unit-assn*^{k} →$_a$ *j-assn*› **and**
k: ‹(*uncurry0 k*, *uncurry0* (*RETURN k-default*)) ∈ *unit-assn*^{k} →$_a$ *k-assn*› **and**
l: ‹(*uncurry0 l*, *uncurry0* (*RETURN l-default*)) ∈ *unit-assn*^{k} →$_a$ *l-assn*› **and**
m: ‹(*uncurry0 m*, *uncurry0* (*RETURN m-default*)) ∈ *unit-assn*^{k} →$_a$ *m-assn*› **and**
n: ‹(*uncurry0 n*, *uncurry0* (*RETURN n-default*)) ∈ *unit-assn*^{k} →$_a$ *n-assn*› **and**
o: ‹(*uncurry0 ko*, *uncurry0* (*RETURN ko-default*)) ∈ *unit-assn*^{k} →$_a$ *o-assn*›**and**
a-free: ‹*MK-FREE a-assn a-free*› **and**
b-free: ‹*MK-FREE b-assn b-free*›**and**
c-free: ‹*MK-FREE c-assn c-free*›**and**
d-free: ‹*MK-FREE d-assn d-free*›**and**
e-free: ‹*MK-FREE e-assn e-free*›**and**
f-free: ‹*MK-FREE f-assn f-free*›**and**
g-free: ‹*MK-FREE g-assn g-free*›**and**
h-free: ‹*MK-FREE h-assn h-free*›**and**
i-free: ‹*MK-FREE i-assn i-free*›**and**
j-free: ‹*MK-FREE j-assn j-free*›**and**
k-free: ‹*MK-FREE k-assn k-free*›**and**
l-free: ‹*MK-FREE l-assn l-free*›**and**
m-free: ‹*MK-FREE m-assn m-free*›**and**
n-free: ‹*MK-FREE n-assn n-free*›**and**
o-free: ‹*MK-FREE o-assn o-free*›
notes [[*sepref-register-adhoc a-default b-default c-default d-default e-default f-default g-default h-default i-default j-default k-default l-default m-default n-default ko-default p-default*]]

begin

lemmas [*sepref-comb-rules*] = *hn-case-tuple15'*[*of* - *a-assn b-assn c-assn d-assn e-assn f-assn g-assn h-assn i-assn j-assn k-assn l-assn m-assn n-assn o-assn*, *unfolded tuple15-int-assn-def*[*symmetric*]]

lemma *ex-tuple15-iff*: (∃ *b* :: (-,-,-,-,-,-,-,-,-,-,-,-,-,-,-)*tuple15*. *P b*) ⟷
(∃ *a b c d e f g h i j k l m n ko*. *P* (*Tuple15 a b c d e f g h i j k l m n ko*))
apply *auto*
apply (*case-tac b*)
by *force*

lemmas [*sepref-frame-free-rules*] = *a-free b-free c-free d-free e-free f-free g-free h-free i-free j-free k-free l-free m-free n-free o-free*
sepref-register
‹*Tuple15*›
lemma [*sepref-fr-rules*]: ‹(*uncurry14* (*Mreturn* o_{15} *Tuple15*), *uncurry14* (*RETURN* o_{15} *Tuple15*))
∈ *a-assn*^{d} $*_a$ *b-assn*^{d} $*_a$ *c-assn*^{d} $*_a$ *d-assn*^{d} $*_a$
e-assn^d $*_a$ *f-assn*^{d} $*_a$ *g-assn*^{d} $*_a$ *h-assn*^{d} $*_a$
i-assn^d $*_a$ *j-assn*^{d} $*_a$ *k-assn*^{d} $*_a$ *l-assn*^{d} $*_a$
m-assn^d $*_a$ *n-assn*^{d} $*_a$ *o-assn*^{d}
$\to_a$ *tuple15-int-assn*›
unfolding *tuple15-int-assn-def*
apply *sepref-to-hoare*
apply *vcg*
unfolding *ex-tuple15-iff*
apply *vcg'*
done

lemma [*sepref-frame-match-rules*]:
‹ *hn-ctxt*
(*tuple15-assn* (*invalid-assn a-assn*) (*invalid-assn b-assn*) (*invalid-assn c-assn*) (*invalid-assn d-assn*) (*invalid-assn e-assn*)
(*invalid-assn f-assn*) (*invalid-assn g-assn*) (*invalid-assn h-assn*) (*invalid-assn i-assn*) (*invalid-assn j-assn*) (*invalid-assn k-assn*)
(*invalid-assn l-assn*) (*invalid-assn m-assn*) (*invalid-assn n-assn*) (*invalid-assn o-assn*)) *ax bx* ⊢ *hn-val UNIV ax bx*›
unfolding *hn-ctxt-def invalid-assn-def tuple15-int-assn-def entails-def*
apply (*auto split*: *prod.split tuple15.splits elim*: *is-pureE*
simp: *sep-algebra-simps pure-part-pure-conj-eq*)
apply (*auto simp*: *pure-part-def*)
apply (*auto simp*: *pure-def pure-true-conv*)
done

lemma *RETURN-case-tuple15-inverse*: ‹*RETURN*
(*let* - = *M*
in ff) =
(*do* {- ← *mop-free M*;
RETURN (*ff*)})›
by (*auto intro*!: *ext simp*: *mop-free-def split*: *tuple15.splits*)

sepref-definition *update-a-code*
is ‹*uncurry* (*RETURN oo update-a*)›
:: ‹*a-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\to_a$ *tuple15-int-assn*›
supply [[*goals-limit*=5]]
unfolding *update-a-def tuple15.case-distrib comp-def RETURN-case-tuple15-inverse*

by *sepref*

sepref-definition *update-b-code*
is ‹*uncurry* (*RETURN oo update-b*)›
:: ‹*b-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
supply [[*goals-limit=1*]]
unfolding *update-b-def tuple15.case-distrib comp-def RETURN-case-tuple15-inverse*
by *sepref*

sepref-definition *update-c-code*
is ‹*uncurry* (*RETURN oo update-c*)›
:: ‹*c-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
supply [[*goals-limit=1*]]
unfolding *update-c-def tuple15.case-distrib comp-def RETURN-case-tuple15-inverse*
by *sepref*

sepref-definition *update-d-code*
is ‹*uncurry* (*RETURN oo update-d*)›
:: ‹*d-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
supply [[*goals-limit=1*]]
unfolding *update-d-def tuple15.case-distrib comp-def RETURN-case-tuple15-inverse*
by *sepref*

sepref-definition *update-e-code*
is ‹*uncurry* (*RETURN oo update-e*)›
:: ‹*e-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
supply [[*goals-limit=1*]]
unfolding *update-e-def tuple15.case-distrib comp-def RETURN-case-tuple15-inverse*
by *sepref*

sepref-definition *update-f-code*
is ‹*uncurry* (*RETURN oo update-f*)›
:: ‹*f-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
supply [[*goals-limit=1*]]
unfolding *update-f-def tuple15.case-distrib comp-def RETURN-case-tuple15-inverse*
by *sepref*

sepref-definition *update-g-code*
is ‹*uncurry* (*RETURN oo update-g*)›
:: ‹*g-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
supply [[*goals-limit=1*]]
unfolding *update-g-def tuple15.case-distrib comp-def RETURN-case-tuple15-inverse*
by *sepref*

sepref-definition *update-h-code*
is ‹*uncurry* (*RETURN oo update-h*)›
:: ‹*h-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
supply [[*goals-limit=1*]]
unfolding *update-h-def tuple15.case-distrib comp-def RETURN-case-tuple15-inverse*
by *sepref*

sepref-definition *update-i-code*
is ‹*uncurry* (*RETURN oo update-i*)›
:: ‹*i-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
supply [[*goals-limit=1*]]
unfolding *update-i-def tuple15.case-distrib comp-def RETURN-case-tuple15-inverse*

by *sepref*

sepref-definition *update-j-code*
is ‹*uncurry* (*RETURN oo update-j*)›
:: ‹*j-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
supply [[*goals-limit=1*]]
unfolding *update-j-def tuple15.case-distrib comp-def RETURN-case-tuple15-inverse*
by *sepref*

sepref-definition *update-k-code*
is ‹*uncurry* (*RETURN oo update-k*)›
:: ‹*k-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
supply [[*goals-limit=1*]]
unfolding *update-k-def tuple15.case-distrib comp-def RETURN-case-tuple15-inverse*
by *sepref*

sepref-definition *update-l-code*
is ‹*uncurry* (*RETURN oo update-l*)›
:: ‹*l-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
supply [[*goals-limit=1*]]
unfolding *update-l-def tuple15.case-distrib comp-def RETURN-case-tuple15-inverse*
by *sepref*

sepref-definition *update-m-code*
is ‹*uncurry* (*RETURN oo update-m*)›
:: ‹*m-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
supply [[*goals-limit=1*]]
unfolding *update-m-def tuple15.case-distrib comp-def RETURN-case-tuple15-inverse*
by *sepref*

sepref-definition *update-n-code*
is ‹*uncurry* (*RETURN oo update-n*)›
:: ‹*n-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
supply [[*goals-limit=1*]]
unfolding *update-n-def tuple15.case-distrib comp-def RETURN-case-tuple15-inverse*
by *sepref*

sepref-definition *update-o-code*
is ‹*uncurry* (*RETURN oo update-o*)›
:: ‹*o-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
supply [[*goals-limit=1*]]
unfolding *update-o-def tuple15.case-distrib comp-def RETURN-case-tuple15-inverse*
by *sepref*

lemma *RETURN-case-tuple15-invers*: ‹(*RETURN* $\circ\circ$ *case-tuple15*)
(λ*x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15*.
ff x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15) =
case-tuple15
(λ*x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15*.
RETURN (*ff x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15*))›
by (*auto intro*!: *ext split*: *tuple15.splits*)

lemmas [*sepref-fr-rules*] = *a b c d e f g h i j k l m n o*

sepref-definition *remove-a-code*
is ‹*RETURN o remove-a*›

:: ‹ *tuple15-int-assn*^{d} $\rightarrow_a$ *a-assn* $\times_a$ *tuple15-int-assn*›
unfolding *remove-a-def RETURN-case-tuple15-invers*
by *sepref*

sepref-definition *remove-b-code*
is ‹*RETURN o remove-b*›
:: ‹ *tuple15-int-assn*^{d} $\rightarrow_a$ *b-assn* $\times_a$ *tuple15-int-assn*›
unfolding *remove-b-def RETURN-case-tuple15-invers*
by *sepref*

sepref-definition *remove-c-code*
is ‹*RETURN o remove-c*›
:: ‹ *tuple15-int-assn*^{d} $\rightarrow_a$ *c-assn* $\times_a$ *tuple15-int-assn*›
unfolding *remove-c-def RETURN-case-tuple15-invers*
by *sepref*

sepref-definition *remove-d-code*
is ‹*RETURN o remove-d*›
:: ‹ *tuple15-int-assn*^{d} $\rightarrow_a$ *d-assn* $\times_a$ *tuple15-int-assn*›
unfolding *remove-d-def RETURN-case-tuple15-invers*
by *sepref*

sepref-definition *remove-e-code*
is ‹*RETURN o remove-e*›
:: ‹ *tuple15-int-assn*^{d} $\rightarrow_a$ *e-assn* $\times_a$ *tuple15-int-assn*›
unfolding *remove-e-def RETURN-case-tuple15-invers*
by *sepref*

sepref-definition *remove-f-code*
is ‹*RETURN o remove-f*›
:: ‹ *tuple15-int-assn*^{d} $\rightarrow_a$ *f-assn* $\times_a$ *tuple15-int-assn*›
unfolding *remove-f-def RETURN-case-tuple15-invers*
by *sepref*

sepref-definition *remove-g-code*
is ‹*RETURN o remove-g*›
:: ‹ *tuple15-int-assn*^{d} $\rightarrow_a$ *g-assn* $\times_a$ *tuple15-int-assn*›
unfolding *remove-g-def RETURN-case-tuple15-invers*
by *sepref*

sepref-definition *remove-h-code*
is ‹*RETURN o remove-h*›
:: ‹ *tuple15-int-assn*^{d} $\rightarrow_a$ *h-assn* $\times_a$ *tuple15-int-assn*›
unfolding *remove-h-def RETURN-case-tuple15-invers*
by *sepref*

sepref-definition *remove-i-code*
is ‹*RETURN o remove-i*›
:: ‹ *tuple15-int-assn*^{d} $\rightarrow_a$ *i-assn* $\times_a$ *tuple15-int-assn*›
unfolding *remove-i-def RETURN-case-tuple15-invers*
by *sepref*

sepref-definition *remove-j-code*
is ‹*RETURN o remove-j*›
:: ‹ *tuple15-int-assn*^{d} $\rightarrow_a$ *j-assn* $\times_a$ *tuple15-int-assn*›
unfolding *remove-j-def RETURN-case-tuple15-invers*

by *sepref*

sepref-definition *remove-k-code*
is ‹*RETURN o remove-k*›
:: ‹ *tuple15-int-assn*^{d} →$_a$ *k-assn* ×$_a$ *tuple15-int-assn*›
unfolding *remove-k-def RETURN-case-tuple15-invers*
by *sepref*

sepref-definition *remove-l-code*
is ‹*RETURN o remove-l*›
:: ‹ *tuple15-int-assn*^{d} →$_a$ *l-assn* ×$_a$ *tuple15-int-assn*›
unfolding *remove-l-def RETURN-case-tuple15-invers*
by *sepref*

sepref-definition *remove-m-code*
is ‹*RETURN o remove-m*›
:: ‹ *tuple15-int-assn*^{d} →$_a$ *m-assn* ×$_a$ *tuple15-int-assn*›
unfolding *remove-m-def RETURN-case-tuple15-invers*
by *sepref*

sepref-definition *remove-n-code*
is ‹*RETURN o remove-n*›
:: ‹ *tuple15-int-assn*^{d} →$_a$ *n-assn* ×$_a$ *tuple15-int-assn*›
unfolding *remove-n-def RETURN-case-tuple15-invers*
by *sepref*

sepref-definition *remove-o-code*
is ‹*RETURN o remove-o*›
:: ‹ *tuple15-int-assn*^{d} →$_a$ *o-assn* ×$_a$ *tuple15-int-assn*›
unfolding *remove-o-def RETURN-case-tuple15-invers*
by *sepref*

lemmas *separation-rules* =
remove-a-code.refine
remove-b-code.refine
remove-c-code.refine
remove-d-code.refine
remove-e-code.refine
remove-f-code.refine
remove-g-code.refine
remove-h-code.refine
remove-i-code.refine
remove-j-code.refine
remove-k-code.refine
remove-l-code.refine
remove-m-code.refine
remove-n-code.refine
remove-o-code.refine
update-a-code.refine
update-b-code.refine
update-c-code.refine
update-d-code.refine
update-e-code.refine
update-f-code.refine
update-g-code.refine
update-h-code.refine

update-i-code.refine
update-j-code.refine
update-k-code.refine
update-l-code.refine
update-m-code.refine
update-n-code.refine
update-o-code.refine

lemmas *code-rules* =
remove-a-code-def
remove-b-code-def
remove-c-code-def
remove-d-code-def
remove-e-code-def
remove-f-code-def
remove-g-code-def
remove-h-code-def
remove-i-code-def
remove-j-code-def
remove-k-code-def
remove-l-code-def
remove-m-code-def
remove-n-code-def
remove-o-code-def
update-a-code-def
update-b-code-def
update-c-code-def
update-d-code-def
update-e-code-def
update-f-code-def
update-g-code-def
update-h-code-def
update-i-code-def
update-j-code-def
update-k-code-def
update-l-code-def
update-m-code-def
update-n-code-def
update-o-code-def

lemmas *setter-and-getters-def* =
update-a-def remove-a-def
update-b-def remove-b-def
update-c-def remove-c-def
update-d-def remove-d-def
update-e-def remove-e-def
update-f-def remove-f-def
update-g-def remove-g-def
update-h-def remove-h-def
update-i-def remove-i-def
update-j-def remove-j-def
update-k-def remove-k-def
update-l-def remove-l-def
update-m-def remove-m-def
update-n-def remove-n-def
update-o-def remove-o-def

end

context *tuple15-state*
begin
lemma *reconstruct-isasat*[*sepref-frame-match-rules*]:
‹*hn-ctxt*
(*tuple15-assn* (*a-assn*) (*b-assn*) (*c-assn*) (*d-assn*) (*e-assn*)
(*f-assn*) (*g-assn*) (*h-assn*) (*i-assn*) (*j-assn*) (*k-assn*)
(*l-assn*) (*m-assn*) (*n-assn*) (*o-assn*)) *ax bx* ⊢ *hn-ctxt tuple15-int-assn ax bx*›
unfolding *tuple15-int-assn-def*
apply (*auto split*: *prod.split tuple15.splits elim*: *is-pureE*
simp: *sep-algebra-simps pure-part-pure-conj-eq*)
done

context
fixes *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and**
read-all-code :: ‹'*xa* ⇒ '*xb* ⇒ '*xc* ⇒ '*xd* ⇒ '*xe* ⇒ '*xf* ⇒ '*xg* ⇒ '*xh* ⇒ '*xi* ⇒ '*xj* ⇒ '*xk* ⇒ '*xl* ⇒ '*xm* ⇒ '*xn* ⇒ '*xo* ⇒ '*q llM*› **and**
read-all :: ‹'*a* ⇒ '*b* ⇒ '*c* ⇒ '*d* ⇒ '*e* ⇒ '*f* ⇒ '*g* ⇒ '*h* ⇒ '*i* ⇒ '*j* ⇒ '*k* ⇒ '*l* ⇒ '*m* ⇒ '*n* ⇒ '*o* ⇒ '*r nres*›
begin

definition *read-all-st-code* :: ‹-› **where**
‹*read-all-st-code xi* = (*case xi of*
Tuple15 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 ⇒
read-all-code a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15)›

definition *read-all-st* :: ‹('*a*, '*b*, '*c*, '*d*, '*e*, '*f*, '*g*, '*h*, '*i*, '*j*,
'*k*, '*l*, '*m*, '*n*, '*o*) *tuple15* ⇒ -› **where**
‹*read-all-st tuple15* = (*case tuple15 of Tuple15 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15a16* ⇒
read-all a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15a16)›

context
fixes *P*
assumes *trail-read*[*sepref-fr-rules*]: ‹(*uncurry14 read-all-code*, *uncurry14 read-all*) ∈
[*uncurry14 P*]$_a$ *a-assn*^{k} *$*_a$ *b-assn*^{k} *$*_a$ *c-assn*^{k} *$*_a$ *d-assn*^{k} *$*_a$ *e-assn*^{k} *$*_a$ *f-assn*^{k} *$*_a$
g-assn^k *$*_a$ *h-assn*^{k} *$*_a$ *i-assn*^{k} *$*_a$ *j-assn*^{k} *$*_a$ *k-assn*^{k} *$*_a$ *l-assn*^{k} *$*_a$
m-assn^k *$*_a$ *n-assn*^{k} *$*_a$ *o-assn*^{k} → *x-assn*›
notes [[*sepref-register-adhoc read-all*]]
begin
sepref-definition *read-all-st-code-tmp*
is *read-all-st*
:: ‹[*case-tuple15 P*]$_a$ *tuple15-int-assn*^{k} → *x-assn*›
unfolding *read-all-st-def*
by *sepref*

lemmas *read-all-st-code-refine* =
read-all-st-code-tmp.refine[*unfolded read-all-st-code-tmp-def*
read-all-st-code-def[*symmetric*]]
end

end

end

lemma *Mreturn-comp-Tuple15*:
‹(*Mreturn* o_{15} *Tuple15*) *a b c d e f g h i j k l m n ko* =
Mreturn (*Tuple15 a b c d e f g h i j k l m n ko*)›
by *auto*

lemma *tuple15-free*[*sepref-frame-free-rules*]:
assumes
‹*MK-FREE A freea*› ‹*MK-FREE B freeb*› ‹*MK-FREE C freec*› ‹*MK-FREE D freed*›
‹*MK-FREE E freee*› ‹*MK-FREE F freef*› ‹*MK-FREE G freeg*› ‹*MK-FREE H freeh*›
‹*MK-FREE I freei*› ‹*MK-FREE J freej*› ‹*MK-FREE K freek*› ‹*MK-FREE L freel*›
‹*MK-FREE M freem*› ‹*MK-FREE N freen*› ‹*MK-FREE KO freeko*›
shows
‹
MK-FREE (*tuple15-assn A B C D E F G H I J K L M N KO*) (λ*S*. *case S of Tuple15 a b c d e f g h i j k l m n ko* ⇒ do_M {
freea a; *freeb b*; *freec c*; *freed d*; *freee e*; *freef f*; *freeg g*; *freeh h*; *freei i*; *freej j*;
freek k; *freel l*; *freem m*; *freen n*; *freeko ko*
})›
supply [*vcg-rules*] = *assms*[*THEN MK-FREED*]
apply (*rule*)+
apply (*clarsimp split*: *tuple15.splits*)
by *vcg'*

end
theory *IsaSAT-Initialisation-State-LLVM*
imports *IsaSAT-Initialisation Tuple15-LLVM IsaSAT-Setup-LLVM IsaSAT-Mark-LLVM*
IsaSAT-VMTF-LLVM Watched-Literals.Watched-Literals-Watch-List-Initialisation
IsaSAT-Mark-LLVM
begin
hide-const (**open**) *NEMonad.RETURN NEMonad.ASSERT*

type-synonym *bump-heuristics-init-assn* = ‹
((*32 word ptr* × *32 word ptr* × *32 word ptr* × *32 word ptr* × *64 word ptr* × *32 word*) × *64 word*,
(*64 word* × *32 word* × *32 word*) *ptr* × *64 word* × *32 word* × *32 word* × *32 word*,
1 word, (*64 word* × *64 word* × *32 word ptr*) × *1 word ptr*) *tuple4*›

type-synonym *vmtf-init* = ‹(*nat*, *nat*) *vmtf-node list* × *nat* × *nat option* × *nat option* × *nat option*›
definition (**in** −) *vmtf-init-assn* :: ‹*vmtf-init* ⇒ - ⇒ *llvm-amemory* ⇒ *bool*› **where**
‹*vmtf-init-assn* ≡ (*array-assn vmtf-node-assn* ×$_a$ *uint64-nat-assn* ×$_a$
atom.option-assn ×$_a$ *atom.option-assn* ×$_a$ *atom.option-assn*)›

type-synonym *bump-heuristics-init* = ‹((*nat*,*nat*)*acids*, *vmtf-init*, *bool*, *nat list* × *bool list*) *tuple4*›

abbreviation *Bump-Heuristics-Init* :: ‹- ⇒ - ⇒ - ⇒ - ⇒ *bump-heuristics-init*› **where**
‹*Bump-Heuristics-Init a b c d* ≡ *Tuple4 a b c d*›

definition *heuristic-bump-init-assn* :: ‹*bump-heuristics-init* ⇒ *bump-heuristics-init-assn* ⇒ -› **where**
‹*heuristic-bump-init-assn* = *tuple4-assn acids-assn2 vmtf-init-assn bool1-assn distinct-atoms-assn*›

definition *bottom-atom* **where**
‹*bottom-atom* = *0*›

definition *bottom-init-vmtf* :: ‹*vmtf-init*› **where**
‹*bottom-init-vmtf* = (*replicate 0* (*VMTF-Node 0 None None*), *0*, *None*, *None*, *None*)›

definition *extract-bump-stable* **where**
‹*extract-bump-stable = tuple4-state-ops.remove-a empty-acids*›
definition *extract-bump-focused* **where**
‹*extract-bump-focused = tuple4-state-ops.remove-b bottom-init-vmtf*›

lemma [*sepref-fr-rules*]: ‹(*uncurry0* (*Mreturn 0*), *uncurry0* (*RETURN bottom-atom*)) ∈ *unit-assn*^{k} →$_a$ *atom-assn*›
unfolding *bottom-atom-def*
apply *sepref-to-hoare*
apply *vcg*
apply (*auto simp*: *atom-rel-def unat-rel-def unat.rel-def br-def entails-def ENTAILS-def*)
by (*smt* (*verit*, *best*) *pure-true-conv rel-simps*(*51*) *sep.add-0*)

sepref-def *bottom-init-vmtf-code*
is ‹*uncurry0* (*RETURN bottom-init-vmtf*)›
:: ‹*unit-assn*^{k} →$_a$ *vmtf-init-assn*›
unfolding *bottom-init-vmtf-def*
apply (*rewrite* **in** ‹((¤, -, -, -, -))› *array-fold-custom-replicate*)
unfolding
atom.fold-option array-fold-custom-replicate vmtf-init-assn-def
al-fold-custom-empty[**where** *'l=64*]
apply (*rewrite at 0* **in** ‹*VMTF-Node* ¤› *unat-const-fold*[**where** *'a=64*])
apply (*rewrite at* ‹(-, ¤, -, -)› *unat-const-fold*[**where** *'a=64*])
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

schematic-goal *free-vmtf-remove-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE vmtf-init-assn ?a*›
unfolding *vmtf-init-assn-def*
by *synthesize-free*

sepref-def *free-vmtf-remove*
is ‹*mop-free*›
:: ‹*vmtf-init-assn*^{d} →$_a$ *unit-assn*›
by *sepref*

definition *extract-bump-is-focused* **where**
‹*extract-bump-is-focused = tuple4-state-ops.remove-c False*›

definition *bottom-atms-hash* **where**
‹*bottom-atms-hash* = ([], *replicate 0 False*)›

definition *extract-bump-atms-to-bump* **where**
‹*extract-bump-atms-to-bump = tuple4-state-ops.remove-d bottom-atms-hash*›

sepref-def *bottom-atms-hash-code*
is ‹*uncurry0* (*RETURN bottom-atms-hash*)›
:: ‹*unit-assn*^{k} →$_a$ *distinct-atoms-assn*›
unfolding *bottom-atms-hash-def*
unfolding
atom.fold-option array-fold-custom-replicate
al-fold-custom-empty[**where** *'l=64*]
apply (*rewrite at* ‹(¤, -)› *annotate-assn*[**where** *A* =‹*al-assn atom-assn*›])
apply (*rewrite at* ‹(-, ¤)› *annotate-assn*[**where** *A* =‹*atoms-hash-assn*›])
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *free-vmtf-init-assn-assn2*: ‹*MK-FREE vmtf-init-assn free-vmtf-remove*›
unfolding *free-vmtf-remove-def*
by (*rule back-subst*[*of* ‹*MK-FREE vmtf-init-assn*›, *OF free-vmtf-remove-assn*])
(*auto intro*!: *ext*)

global-interpretation *Bump-Heur-Init*: *tuple4-state* **where**
a-assn = *acids-assn2* **and**
b-assn = *vmtf-init-assn* **and**
c-assn = *bool1-assn* **and**
d-assn = *distinct-atoms-assn* **and**
a-default = *empty-acids* **and**
a = ‹*empty-acids-code*› **and**
a-free = *free-acids* **and**
b-default = *bottom-init-vmtf* **and**
b = ‹*bottom-init-vmtf-code*› **and**
b-free = *free-vmtf-remove* **and**
c-default = *False* **and**
c = ‹*bottom-focused*› **and**
c-free = *free-focused* **and**
d-default = ‹*bottom-atms-hash*› **and**
d = ‹*bottom-atms-hash-code*› **and**
d-free = ‹*free-atms-hash-code*›
rewrites
‹*Bump-Heur-Init.tuple4-int-assn* ≡ *heuristic-bump-init-assn*›**and**
‹*Bump-Heur-Init.remove-a* ≡ *extract-bump-stable*› **and**
‹*Bump-Heur-Init.remove-b* ≡ *extract-bump-focused*› **and**
‹*Bump-Heur-Init.remove-c* ≡ *extract-bump-is-focused*› **and**
‹*Bump-Heur-Init.remove-d* ≡ *extract-bump-atms-to-bump*›
apply *unfold-locales*
apply (*rule bottom-init-vmtf-code.refine bottom-focused.refine free-acids-assn2*
bottom-atms-hash-code.refine free-vmtf-init-assn-assn2 free-focused-assn2
free-distinct-atoms-assn2 empty-acids-code.refine free-acids-assn2')+
subgoal unfolding *heuristic-bump-init-assn-def tuple4-state-ops.tuple4-int-assn-def* **by** *auto*
subgoal unfolding *extract-bump-stable-def* **by** *auto*
subgoal unfolding *extract-bump-focused-def* **by** *auto*
subgoal unfolding *extract-bump-is-focused-def* **by** *auto*
subgoal unfolding *extract-bump-atms-to-bump-def* **by** *auto*
done

lemmas [*unfolded Tuple4-LLVM.inline-direct-return-node-case*, *llvm-code*] =
Bump-Heur-Init.code-rules[*unfolded Mreturn-comp-Tuple4*]

lemmas [*sepref-fr-rules*] =
Bump-Heur-Init.separation-rules

type-synonym (**in** −)*twl-st-wll-trail-init* =
‹(*trail-pol-fast-assn*, *arena-assn*, *option-lookup-clause-assn*,
64 word, *watched-wl-uint32*, *bump-heuristics-init-assn*, *phase-saver-assn*,
32 word, *cach-refinement-l-assn*, *lbd-assn*, *vdom-fast-assn*, *vdom-fast-assn*, *1 word*,
(*64 word* × *64 word* × *64 word* × *64 word* × *64 word*), *mark-assn*) *tuple15*›

definition *isasat-init-assn*
:: ‹*twl-st-wl-heur-init* ⇒ *twl-st-wll-trail-init* ⇒ *assn*›
where

‹*isasat-init-assn* = *tuple15-assn*
trail-pol-fast-assn arena-fast-assn
conflict-option-rel-assn
sint64-nat-assn
watchlist-fast-assn
heuristic-bump-init-assn phase-saver-assn
uint32-nat-assn
cach-refinement-l-assn
lbd-assn
vdom-fast-assn
vdom-fast-assn
bool1-assn lcount-assn
marked-struct-assn›

definition *bottom-vdom* :: ‹-› **where**
‹*bottom-vdom* = []›

sepref-def *bottom-vdom-code*
is ‹*uncurry0* (*RETURN bottom-vdom*)›
:: ‹*unit-assn*^{k} →$_a$ *vdom-fast-assn*›
supply [[*goals-limit=1*]]
unfolding *bottom-vdom-def*
unfolding *al-fold-custom-empty*[**where** *'l=64*]
by *sepref*

sepref-def *free-vdom*
is ‹*mop-free*›
:: ‹*vdom-fast-assn*^{d} →$_a$ *unit-assn*›
by *sepref*

schematic-goal *free-vdom*[*sepref-frame-free-rules*]: ‹*MK-FREE vdom-fast-assn ?a*›
by *synthesize-free*

lemma *free-vdom2*: ‹*MK-FREE vdom-fast-assn free-vdom*›
unfolding *free-arena-fast-def*
by (*rule back-subst*[*of* ‹*MK-FREE vdom-fast-assn*›, *OF free-vdom*])
(*auto intro*!: *ext simp*: *free-vdom-def*)

sepref-def *free-phase-saver*
is ‹*mop-free*›
:: ‹*phase-saver-assn*^{d} →$_a$ *unit-assn*›
by *sepref*

definition *bottom-phase-saver* :: ‹*phase-saver*› **where**
‹*bottom-phase-saver* = *op-larray-custom-replicate 0 False*›

sepref-def *bottom-phase-saver-code*
is ‹*uncurry0* (*RETURN bottom-phase-saver*)›
:: ‹*unit-assn*^{k} →$_a$ *phase-saver-assn*›
supply [[*goals-limit=1*]]
unfolding *bottom-phase-saver-def*
unfolding *larray-fold-custom-replicate*

apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

schematic-goal *free-phase-saver*[*sepref-frame-free-rules*]: ‹*MK-FREE phase-saver-assn ?a*›
by *synthesize-free*

lemma *free-phase-saver2*: ‹*MK-FREE phase-saver-assn free-phase-saver*›
unfolding *free-arena-fast-def*
by (*rule back-subst*[*of* ‹*MK-FREE phase-saver-assn*›, *OF free-phase-saver*])
(*auto intro*!: *ext simp*: *free-phase-saver-def*)

definition *bottom-init-bump* :: ‹*bump-heuristics-init*› **where**
‹*bottom-init-bump* = *Bump-Heuristics-Init empty-acids bottom-init-vmtf False bottom-atms-hash*›

schematic-goal *free-vmtf-init-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE heuristic-bump-init-assn ?a*›
unfolding *heuristic-bump-init-assn-def*
by *synthesize-free*

sepref-def *free-bottom-init-bump-code*
is ‹*mop-free*›
:: ‹*heuristic-bump-init-assn*^{d} →$_a$ *unit-assn*›
by *sepref*

lemma *free-vmtf-remove2*: ‹*MK-FREE heuristic-bump-init-assn free-bottom-init-bump-code*›
unfolding *free-bottom-init-bump-code-def*
apply (*rule back-subst*[*of* ‹*MK-FREE heuristic-bump-init-assn*›, *OF free-vmtf-init-assn*])
apply (*auto intro*!: *ext simp*: *M-monad-laws*)
by (*metis M-monad-laws*(*1*))

definition *op-empty-array* **where**
‹*op-empty-array* ≡ []›

lemma [*sepref-fr-rules*]: ‹(*uncurry0* (*Mreturn null*), *uncurry0* (*RETURN op-empty-array*)) ∈ *unit-assn*^{k} →$_a$ *array-assn R*›
unfolding *array-assn-def op-empty-array-def*
apply *sepref-to-hoare*
apply *vcg*
apply (*auto simp*: *array-assn-def ENTAILS-def hr-comp-def*[*abs-def*] *Exists-eq-simp*
pure-true-conv narray-assn-null-init)
done

sepref-def *bottom-init-vmtf2-code*
is ‹*uncurry0* (*RETURN bottom-init-bump*)›
:: ‹*unit-assn*^{k} →$_a$ *heuristic-bump-init-assn*›
unfolding *bottom-init-bump-def atom.fold-None*
unfolding *al-fold-custom-empty*[**where** *'l*=*64*]
by *sepref*

definition *bottom-bool* **where**
‹*bottom-bool* = *False*›

sepref-def *bottom-bool-code*
is ‹*uncurry0* (*RETURN bottom-bool*)›
:: ‹*unit-assn*^{k} →$_a$ *bool1-assn*›

unfolding *bottom-bool-def*
by *sepref*

sepref-def *free-bool*
is ‹*mop-free*›
:: ‹*bool1-assn*^{d} →$_a$ *unit-assn*›
by *sepref*

schematic-goal *free-bool*[*sepref-frame-free-rules*]: ‹*MK-FREE bool1-assn ?a*›
by *synthesize-free*

lemma *free-bool2*: ‹*MK-FREE bool1-assn free-bool*›
unfolding *free-focused-def*
by (*rule back-subst*[*of* ‹*MK-FREE bool1-assn*›, *OF free-bool*])
(*auto intro*!: *ext simp*: *free-bool-def free-focused-def*)

definition *bottom-marked-struct* :: ‹-› **where**
‹*bottom-marked-struct* = (*0*, [])›

sepref-def *bottom-marked-struct-code*
is ‹*uncurry0* (*RETURN bottom-marked-struct*)›
:: ‹*unit-assn*^{k} →$_a$ *marked-struct-assn*›
unfolding *bottom-marked-struct-def marked-struct-assn-def op-empty-array-def*[*symmetric*]
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

sepref-def *free-marked-struct-code*
is ‹*mop-free*›
:: ‹*marked-struct-assn*^{d} →$_a$ *unit-assn*›
unfolding *bottom-marked-struct-def marked-struct-assn-def*
by *sepref*

schematic-goal *free-marked-struct*[*sepref-frame-free-rules*]: ‹*MK-FREE marked-struct-assn ?a*›
unfolding *marked-struct-assn-def*
by *synthesize-free*

lemma *free-marked-struct2*: ‹*MK-FREE marked-struct-assn free-marked-struct-code*›
unfolding *free-arena-fast-def*
by (*rule back-subst*[*of* ‹*MK-FREE marked-struct-assn*›, *OF free-marked-struct*])
(*auto intro*!: *ext simp*: *free-marked-struct-code-def*)

global-interpretation *IsaSAT-Init*: *tuple15-state-ops* **where**
a-assn = *trail-pol-fast-assn* **and**
b-assn = *arena-fast-assn* **and**
c-assn = *conflict-option-rel-assn* **and**
d-assn = *sint64-nat-assn* **and**
e-assn = *watchlist-fast-assn* **and**
f-assn = *heuristic-bump-init-assn* **and**
g-assn = *phase-saver-assn* **and**
h-assn = *uint32-nat-assn* **and**
i-assn = *cach-refinement-l-assn* **and**
j-assn = *lbd-assn* **and**
k-assn = *vdom-fast-assn* **and**
l-assn = *vdom-fast-assn* **and**
m-assn = *bool1-assn* **and**
n-assn = *lcount-assn* **and**

o-assn = *marked-struct-assn* **and**
a-default = *bottom-trail* **and**
a = ‹*bottom-trail-code*› **and**
b-default = *bottom-arena* **and**
b = ‹*bottom-arena-code*› **and**
c-default = *bottom-conflict* **and**
c = ‹*bottom-conflict-code*› **and**
d-default = ‹*bottom-decision-level*› **and**
d = ‹(*bottom-decision-level-code*)› **and**
e-default = *bottom-watchlist* **and**
e = ‹*bottom-watchlist-code*› **and**
f-default = *bottom-init-bump* **and**
f = ‹*bottom-init-vmtf2-code*› **and**
g-default = *bottom-phase-saver* **and**
g = ‹*bottom-phase-saver-code*› **and**
h-default = *bottom-clvls* **and**
h = ‹*bottom-clvls-code*›**and**
i-default = *bottom-ccach* **and**
i = ‹*bottom-ccach-code*› **and**
j-default = *empty-lbd* **and**
j = ‹*empty-lbd-code*› **and**
k-default = *bottom-vdom* **and**
k = ‹*bottom-vdom-code*› **and**
l-default = *bottom-vdom* **and**
l = ‹*bottom-vdom-code*› **and**
m-default = *bottom-bool* **and**
m = ‹*bottom-bool-code*› **and**
n-default = *bottom-lcount* **and**
n = ‹*bottom-lcount-code*› **and**
ko-default = *bottom-marked-struct* **and**
ko = ‹*bottom-marked-struct-code*›
.

definition *extract-trail-wl-heur-init* **where**
‹*extract-trail-wl-heur-init* = *IsaSAT-Init.remove-a*›

definition *extract-arena-wl-heur-init* **where**
‹*extract-arena-wl-heur-init* = *IsaSAT-Init.remove-b*›

definition *extract-conflict-wl-heur-init* **where**
‹*extract-conflict-wl-heur-init* = *IsaSAT-Init.remove-c*›

definition *extract-literals-to-update-wl-heur-init* **where**
‹*extract-literals-to-update-wl-heur-init* = *IsaSAT-Init.remove-d*›

definition *extract-watchlist-wl-heur-init* **where**
‹*extract-watchlist-wl-heur-init* = *IsaSAT-Init.remove-e*›

definition *extract-vmtf-wl-heur-init* **where**
‹*extract-vmtf-wl-heur-init* = *IsaSAT-Init.remove-f*›

definition *extract-phase-saver-wl-heur-init* **where**
‹*extract-phase-saver-wl-heur-init* = *IsaSAT-Init.remove-g*›

definition *extract-clvls-wl-heur-init* **where**
‹*extract-clvls-wl-heur-init* = *IsaSAT-Init.remove-h*›

definition *extract-ccach-wl-heur-init* **where**
‹*extract-ccach-wl-heur-init* = *IsaSAT-Init.remove-i*›

definition *extract-lbd-wl-heur-init* **where**
‹*extract-lbd-wl-heur-init* = *IsaSAT-Init.remove-j*›

definition *extract-vdom-wl-heur-init* **where**
‹*extract-vdom-wl-heur-init* = *IsaSAT-Init.remove-k*›

definition *extract-ivdom-wl-heur-init* **where**
‹*extract-ivdom-wl-heur-init* = *IsaSAT-Init.remove-l*›

definition *extract-failed-wl-heur-init* **where**
‹*extract-failed-wl-heur-init* = *IsaSAT-Init.remove-m*›

definition *extract-lcount-wl-heur-init* **where**
‹*extract-lcount-wl-heur-init* = *IsaSAT-Init.remove-n*›

definition *extract-marked-wl-heur-init* **where**
‹*extract-marked-wl-heur-init* = *IsaSAT-Init.remove-o*›

global-interpretation *IsaSAT-Init*: *tuple15-state* **where**
a-assn = *trail-pol-fast-assn* **and**
b-assn = *arena-fast-assn* **and**
c-assn = *conflict-option-rel-assn* **and**
d-assn = *sint64-nat-assn* **and**
e-assn = *watchlist-fast-assn* **and**
f-assn = *heuristic-bump-init-assn* **and**
g-assn = *phase-saver-assn* **and**
h-assn = *uint32-nat-assn* **and**
i-assn = *cach-refinement-l-assn* **and**
j-assn = *lbd-assn* **and**
k-assn = *vdom-fast-assn* **and**
l-assn = *vdom-fast-assn* **and**
m-assn = *bool1-assn* **and**
n-assn = *lcount-assn* **and**
o-assn = *marked-struct-assn* **and**
a-default = *bottom-trail* **and**
a = ‹*bottom-trail-code*› **and**
a-free = *free-trail-pol-fast* **and**
b-default = *bottom-arena* **and**
b = ‹*bottom-arena-code*› **and**
b-free = *free-arena-fast* **and**
c-default = *bottom-conflict* **and**
c = ‹*bottom-conflict-code*› **and**
c-free = *free-conflict-option-rel* **and**
d-default = ‹*bottom-decision-level*› **and**
d = ‹*bottom-decision-level-code*› **and**
d-free = ‹*free-sint64-nat*› **and**
e-default = *bottom-watchlist* **and**
e = ‹*bottom-watchlist-code*› **and**
e-free = *free-watchlist-fast* **and**
f-default = *bottom-init-bump* **and**
f = ‹*bottom-init-vmtf2-code*› **and**
f-free = *free-bottom-init-bump-code* **and**

g-default = *bottom-phase-saver* **and**
g = ‹*bottom-phase-saver-code*› **and**
g-free = *free-phase-saver* **and**
h-default = *bottom-clvls* **and**
h = ‹*bottom-clvls-code*›**and**
h-free = *free-uint32-nat* **and**
i-default = *bottom-ccach* **and**
i = ‹*bottom-ccach-code*› **and**
i-free = *free-cach-refinement-l* **and**
j-default = *empty-lbd* **and**
j = ‹*empty-lbd-code*› **and**
j-free = *free-lbd* **and**
k-default = *bottom-vdom* **and**
k = ‹*bottom-vdom-code*› **and**
k-free = *free-vdom* **and**
l-default = *bottom-vdom* **and**
l = ‹*bottom-vdom-code*› **and**
l-free = *free-vdom* **and**
m-default = *bottom-bool* **and**
m = ‹*bottom-bool-code*› **and**
m-free = *free-bool* **and**
n-default = *bottom-lcount* **and**
n = ‹*bottom-lcount-code*› **and**
n-free = *free-lcount* **and**
ko-default = *bottom-marked-struct* **and**
ko = ‹*bottom-marked-struct-code*› **and**
o-free = *free-marked-struct-code*
rewrites
‹*IsaSAT-Init.tuple15-int-assn* = *isasat-init-assn*› **and**
‹*IsaSAT-Init.remove-a* = *extract-trail-wl-heur-init*› **and**
‹*IsaSAT-Init.remove-b* = *extract-arena-wl-heur-init*› **and**
‹*IsaSAT-Init.remove-c* = *extract-conflict-wl-heur-init*› **and**
‹*IsaSAT-Init.remove-d* = *extract-literals-to-update-wl-heur-init*› **and**
‹*IsaSAT-Init.remove-e* = *extract-watchlist-wl-heur-init*› **and**
‹*IsaSAT-Init.remove-f* = *extract-vmtf-wl-heur-init*› **and**
‹*IsaSAT-Init.remove-g* = *extract-phase-saver-wl-heur-init*› **and**
‹*IsaSAT-Init.remove-h* = *extract-clvls-wl-heur-init*› **and**
‹*IsaSAT-Init.remove-i* = *extract-ccach-wl-heur-init*› **and**
‹*IsaSAT-Init.remove-j* = *extract-lbd-wl-heur-init*› **and**
‹*IsaSAT-Init.remove-k* = *extract-vdom-wl-heur-init*› **and**
‹*IsaSAT-Init.remove-l* = *extract-ivdom-wl-heur-init*› **and**
‹*IsaSAT-Init.remove-m* = *extract-failed-wl-heur-init*› **and**
‹*IsaSAT-Init.remove-n* = *extract-lcount-wl-heur-init*› **and**
‹*IsaSAT-Init.remove-o* = *extract-marked-wl-heur-init*›
apply *unfold-locales*
subgoal by (*rule bottom-trail-code.refine*)
subgoal by (*rule bottom-arena-code.refine*)
subgoal by (*rule bottom-conflict-code.refine*)
subgoal by (*rule bottom-decision-level-code.refine*)
subgoal by (*rule bottom-watchlist-code.refine*)
subgoal by (*rule bottom-init-vmtf2-code.refine*)
subgoal by (*rule bottom-phase-saver-code.refine*)
subgoal by (*rule bottom-clvls-code.refine*)
subgoal by (*rule bottom-ccach-code.refine*)
subgoal by (*rule empty-lbd-hnr*)
subgoal by (*rule bottom-vdom-code.refine*)

subgoal by (*rule bottom-bool-code.refine*)
subgoal by (*rule bottom-lcount-code.refine*)
subgoal by (*rule bottom-marked-struct-code.refine*)
subgoal by (*rule free-trail-pol-fast-assn2*)
subgoal by (*rule free-arena-fast-assn2*)
subgoal by (*rule free-conflict-option-rel-assn2*)
subgoal by (*synthesize-free*)
subgoal by (*synthesize-free*)
subgoal by (*rule free-vmtf-remove2*)
subgoal by (*rule free-phase-saver2*)
subgoal by (*synthesize-free*)
subgoal by (*synthesize-free*)
subgoal by (*synthesize-free*)
subgoal by (*rule free-vdom2*)
subgoal by (*rule free-bool2*)
subgoal by (*synthesize-free*)
subgoal by (*rule free-marked-struct2*)
subgoal unfolding *isasat-init-assn-def tuple15-state-ops.tuple15-int-assn-def* **..**
subgoal unfolding *extract-trail-wl-heur-init-def* **..**
subgoal unfolding *extract-arena-wl-heur-init-def* **..**
subgoal unfolding *extract-conflict-wl-heur-init-def* **..**
subgoal unfolding *extract-literals-to-update-wl-heur-init-def* **..**
subgoal unfolding *extract-watchlist-wl-heur-init-def* **..**
subgoal unfolding *extract-vmtf-wl-heur-init-def* **..**
subgoal unfolding *extract-phase-saver-wl-heur-init-def* **..**
subgoal unfolding *extract-clvls-wl-heur-init-def* **..**
subgoal unfolding *extract-ccach-wl-heur-init-def* **..**
subgoal unfolding *extract-lbd-wl-heur-init-def* **..**
subgoal unfolding *extract-vdom-wl-heur-init-def* **..**
subgoal unfolding *extract-ivdom-wl-heur-init-def* **..**
subgoal unfolding *extract-failed-wl-heur-init-def* **..**
subgoal unfolding *extract-lcount-wl-heur-init-def* **..**
subgoal unfolding *extract-marked-wl-heur-init-def* **..**
done

lemmas [*unfolded Tuple15-LLVM.inline-direct-return-node-case*, *llvm-code*] =
IsaSAT-Init.code-rules[*unfolded Mreturn-comp-Tuple15*]

lemmas [*sepref-fr-rules*] =
IsaSAT-Init.separation-rules

lemmas *isasat-init-getters-and-setters-def* =
IsaSAT-Init.setter-and-getters-def
extract-trail-wl-heur-init-def
extract-arena-wl-heur-init-def
extract-conflict-wl-heur-init-def
extract-literals-to-update-wl-heur-init-def
extract-marked-wl-heur-init-def
extract-lcount-wl-heur-init-def
extract-failed-wl-heur-init-def
extract-ivdom-wl-heur-init-def
extract-vdom-wl-heur-init-def
extract-lbd-wl-heur-init-def
extract-ccach-wl-heur-init-def
extract-clvls-wl-heur-init-def
extract-phase-saver-wl-heur-init-def

extract-vmtf-wl-heur-init-def
extract-watchlist-wl-heur-init-def
IsaSAT-Init.remove-a-def
IsaSAT-Init.remove-b-def
IsaSAT-Init.remove-c-def
IsaSAT-Init.remove-d-def
IsaSAT-Init.remove-e-def
IsaSAT-Init.remove-f-def
IsaSAT-Init.remove-g-def
IsaSAT-Init.remove-h-def
IsaSAT-Init.remove-i-def
IsaSAT-Init.remove-j-def
IsaSAT-Init.remove-k-def
IsaSAT-Init.remove-l-def
IsaSAT-Init.remove-m-def
IsaSAT-Init.remove-n-def
IsaSAT-Init.remove-o-def

lemma (**in** −) *case-isasat-int-split-getter*: ‹*P*
(*Tuple15-a S*)
(*Tuple15-b S*)
(*Tuple15-c S*)
(*Tuple15-d S*)
(*Tuple15-e S*)
(*Tuple15-f S*)
(*Tuple15-g S*)
(*Tuple15-h S*)
(*Tuple15-i S*)
(*Tuple15-j S*)
(*Tuple15-k S*)
(*Tuple15-l S*)
(*Tuple15-m S*)
(*Tuple15-n S*)
(*Tuple15-o S*) = *case-tuple15 P S*›
by (*auto split*: *tuple15.splits*)

context *tuple15-state*
begin
context
fixes
f' :: ‹'*s* ⇒ '*a* ⇒ '*b* ⇒ '*c* ⇒ '*d* ⇒ '*e* ⇒ '*f* ⇒ '*g* ⇒ '*h* ⇒ '*i* ⇒ '*j* ⇒ '*k* ⇒ '*l* ⇒ '*m* ⇒ '*n* ⇒ '*o* ⇒ '*x*
nres› **and**
P :: ‹'*s* ⇒ '*a* ⇒ '*b* ⇒ '*c* ⇒ '*d* ⇒ '*e* ⇒ '*f* ⇒ '*g* ⇒ '*h* ⇒ '*i* ⇒ '*j* ⇒ '*k* ⇒ '*l* ⇒ '*m* ⇒ '*n* ⇒ '*o* ⇒ *bool*›
begin
definition *mop* **where**
‹*mop S C* = *do* {
ASSERT (*P C*
(*Tuple15-a S*)
(*Tuple15-b S*)
(*Tuple15-c S*)
(*Tuple15-d S*)
(*Tuple15-e S*)
(*Tuple15-f S*)

```
  (Tuple15-g S)
  (Tuple15-h S)
  (Tuple15-i S)
  (Tuple15-j S)
  (Tuple15-k S)
  (Tuple15-l S)
  (Tuple15-m S)
  (Tuple15-n S)
  (Tuple15-o S));
   read-all-st (f' C) S
  }›

context
  fixes R and kf and x-assn :: ‹'x ⇒ 'q ⇒ assn›
  assumes not-deleted-code-refine:
    ‹(uncurry15 (λa b c d e f g h i j k l m n ko C. kf C a b c d e f g h i j k l m n ko),
    uncurry15 (λa b c d e f g h i j k l m n ko C. f' C a b c d e f g h i j k l m n ko))
    ∈ [uncurry15 (λa b c d e f g h i j k l m n ko C. P C a b c d e f g h i j k l m n ko)]a
    a-assn^k *a b-assn^k *a c-assn^k *a d-assn^k *a
  e-assn^k *a f-assn^k *a g-assn^k *a h-assn^k *a i-assn^k *a
  j-assn^k *a k-assn^k *a l-assn^k *a m-assn^k *a
  n-assn^k *a o-assn^k *a (pure R)^k → x-assn›
begin

context
begin
lemma not-deleted-code-refine-tmp:
  ‹⋀C C'. (C, C') ∈ R ⟹ (uncurry14 (kf C), uncurry14 (f' C')) ∈ [uncurry14 (P C')]a
    a-assn^k *a b-assn^k *a c-assn^k *a d-assn^k *a
  e-assn^k *a f-assn^k *a g-assn^k *a h-assn^k *a i-assn^k *a
  j-assn^k *a k-assn^k *a l-assn^k *a m-assn^k *a
  n-assn^k *a o-assn^k → x-assn›
  apply (rule remove-pure-parameter2'[where R=R])
  apply (rule hfref-cong[OF not-deleted-code-refine])
  apply (auto simp add: uncurry-def)
  done
end
lemma read-all-refine:
  ‹(uncurry (λN C. read-all-st-code (kf C) N),
    uncurry (λN C'. read-all-st (f' C') N))
  ∈ [uncurry (λS C. P C
  (Tuple15-a S)
  (Tuple15-b S)
  (Tuple15-c S)
  (Tuple15-d S)
  (Tuple15-e S)
  (Tuple15-f S)
  (Tuple15-g S)
  (Tuple15-h S)
  (Tuple15-i S)
  (Tuple15-j S)
  (Tuple15-k S)
  (Tuple15-l S)
  (Tuple15-m S)
  (Tuple15-n S)
  (Tuple15-o S))]a tuple15-int-assn^k *a (pure R)^k→ x-assn›
```

apply (*rule add-pure-parameter2*)
unfolding *tuple15.case-distrib case-isasat-int-split-getter*
apply (*rule read-all-st-code-refine*)
apply (*rule not-deleted-code-refine-tmp*)
apply *assumption*
done

lemma *read-all-mop-refine*:
‹(*uncurry* (λ*N C*. *read-all-st-code* (*kf C*) *N*),
uncurry mop)
∈ *tuple15-int-assn*^{k} $*_a$ (*pure R*)^k→$_a$ *x-assn*›
unfolding *mop-def*
apply (*rule refine-ASSERT-move-to-pre*)
apply (*rule read-all-refine*)
done
end
end

abbreviation *read-trail-wl-heur-code* :: ‹-› **where**
‹*read-trail-wl-heur-code kf* ≡ *IsaSAT-Init.read-all-st-code* (λ*M* - - - - - - - - - - - - - - -. *kf M*)›
abbreviation *read-trail-wl-heur* :: ‹-› **where**
‹*read-trail-wl-heur kf* ≡ *IsaSAT-Init.read-all-st* (λ*M* - - - - - - - - - - - - - - -. *kf M*)›

context
fixes *R* **and** *kf* **and** *f'* **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** *P*
assumes *not-deleted-code-refine*: ‹(*uncurry* (λ*S C*. *kf C S*), *uncurry* (λ*S C*. *f' C S*)) ∈ [*uncurry* (λ*S C*. *P C S*)]$_a$ *a-assn*^{k} $*_a$ (*pure R*)^k → *x-assn*›
begin
private lemma *not-deleted-code-refine'*:
‹(*uncurry15* (λ*M* - - - - - - - - - - - - - - - *C*. *kf C M*), *uncurry15* (λ*M* - - - - - - - - - - - - - - - *C'*. *f' C' M*)) ∈ [*uncurry15* (λ*M* - - - - - - - - - - - - - - - *C*. *P C M*)]$_a$
a-assn^k $*_a$ *b-assn*^{k} $*_a$ *c-assn*^{k} $*_a$ *d-assn*^{k} $*_a$
e-assn^k $*_a$ *f-assn*^{k} $*_a$ *g-assn*^{k} $*_a$ *h-assn*^{k} $*_a$ *i-assn*^{k} $*_a$
j-assn^k $*_a$ *k-assn*^{k} $*_a$ *l-assn*^{k} $*_a$ *m-assn*^{k} $*_a$
n-assn^k $*_a$ *o-assn*^{k} $*_a$ (*pure R*)^k → *x-assn*›
apply (*insert not-deleted-code-refine*)
apply (*drule remove-component-middle*[**where** *X* = *b-assn*])
apply (*drule remove-component-middle*[**where** *X* = *c-assn*])
apply (*drule remove-component-middle*[**where** *X* = *d-assn*])
apply (*drule remove-component-middle*[**where** *X* = *e-assn*])
apply (*drule remove-component-middle*[**where** *X* = *f-assn*])
apply (*drule remove-component-middle*[**where** *X* = *g-assn*])
apply (*drule remove-component-middle*[**where** *X* = *h-assn*])
apply (*drule remove-component-middle*[**where** *X* = *i-assn*])
apply (*drule remove-component-middle*[**where** *X* = *j-assn*])
apply (*drule remove-component-middle*[**where** *X* = *k-assn*])
apply (*drule remove-component-middle*[**where** *X* = *l-assn*])
apply (*drule remove-component-middle*[**where** *X* = *m-assn*])
apply (*drule remove-component-middle*[**where** *X* = *n-assn*])
apply (*drule remove-component-middle*[**where** *X* = *o-assn*])
apply (*rule hfref-cong*, *assumption*)
apply (*auto simp add*: *uncurry-def*)
done

lemmas *read-trail-refine* = *read-all-refine*[*OF not-deleted-code-refine'*]

lemmas *mop-read-trail-refine* = *read-all-mop-refine*[*OF not-deleted-code-refine'*]
end

abbreviation *read-conflict-wl-heur-code* :: ‹-› **where**
‹*read-conflict-wl-heur-code kf* ≡ *IsaSAT-Init.read-all-st-code* (λ*M* - - - - - - - - - - - - - - -. *kf M*)›
abbreviation *read-conflict-wl-heur* :: ‹-› **where**
‹*read-conflict-wl-heur kf* ≡ *IsaSAT-Init.read-all-st* (λ*M* - - - - - - - - - - - - - - -. *kf M*)›

context
fixes *R* **and** *kf* **and** *f'* **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** *P*
assumes *not-deleted-code-refine*: ‹(*uncurry* (λ*S C*. *kf C S*), *uncurry* (λ*S C*. *f' C S*)) ∈ [*uncurry* (λ*S C*. *P C S*)]$_a$ *c-assn*^{k} ∗$_a$ (*pure R*)^k → *x-assn*›
begin
private lemma *not-deleted-code-refine-conflict*:
‹(*uncurry15* (λ- - *M* - - - - - - - - - - - - - *C*. *kf C M*), *uncurry15* (λ- - *M* - - - - - - - - - - - - - *C'*. *f' C' M*)) ∈ [*uncurry15* (λ- - *M* - - - - - - - - - - - - - *C*. *P C M*)]$_a$
a-assn^k ∗$_a$ *b-assn*^{k} ∗$_a$ *c-assn*^{k} ∗$_a$ *d-assn*^{k} ∗$_a$
e-assn^k ∗$_a$ *f-assn*^{k} ∗$_a$ *g-assn*^{k} ∗$_a$ *h-assn*^{k} ∗$_a$ *i-assn*^{k} ∗$_a$
j-assn^k ∗$_a$ *k-assn*^{k} ∗$_a$ *l-assn*^{k} ∗$_a$ *m-assn*^{k} ∗$_a$
n-assn^k ∗$_a$ *o-assn*^{k} ∗$_a$ (*pure R*)^k → *x-assn*›
apply (*rule add-pure-parameter2*)
apply (*drule remove-pure-parameter2'*[**where** *f* =*kf* **and** *f'*=*f'*, *OF not-deleted-code-refine*])
apply (*drule remove-component-left*[**where** *X* = *a-assn*])
apply (*drule remove-component-middle*[**where** *X* = *b-assn*])
apply (*drule remove-component-right*[**where** *X* = *d-assn*])
apply (*drule remove-component-right*[**where** *X* = *e-assn*])
apply (*drule remove-component-right*[**where** *X* = *f-assn*])
apply (*drule remove-component-right*[**where** *X* = *g-assn*])
apply (*drule remove-component-right*[**where** *X* = *h-assn*])
apply (*drule remove-component-right*[**where** *X* = *i-assn*])
apply (*drule remove-component-right*[**where** *X* = *j-assn*])
apply (*drule remove-component-right*[**where** *X* = *k-assn*])
apply (*drule remove-component-right*[**where** *X* = *l-assn*])
apply (*drule remove-component-right*[**where** *X* = *m-assn*])
apply (*drule remove-component-right*[**where** *X* = *n-assn*])
apply (*drule remove-component-right*[**where** *X* = *o-assn*])
apply (*rule hfref-cong*, *assumption*)
apply (*auto simp add*: *uncurry-def*)
done

lemmas *read-conflict-refine* = *read-all-refine*[*OF not-deleted-code-refine-conflict*]
lemmas *mop-read-conflict-refine* = *read-all-mop-refine*[*OF not-deleted-code-refine-conflict*]
end

context
fixes *R* **and** *kf* **and** *f'* **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** *P*
assumes *not-deleted-code-refine*: ‹((λ*S*. *kf S*), (λ*S*. *f' S*)) ∈ [(λ*S*. *P S*)]$_a$ *c-assn*^{k} → *x-assn*›
begin

lemmas *read-conflict-refine0* = *read-conflict-refine*[*OF not-deleted-code-refine*[*THEN remove-component-right*],
THEN remove-unused-unit-parameter]
lemmas *mop-read-conflict-refine0* = *mop-read-conflict-refine*[*OF not-deleted-code-refine*[*THEN remove-component-right*]

end

abbreviation *read-b-wl-heur-code* :: ‹-› **where**
‹*read-b-wl-heur-code kf* ≡ *IsaSAT-Init.read-all-st-code* (λ- *M* - - - - - - - - - - - - - -. *kf M*)›
abbreviation *read-b-wl-heur* :: ‹-› **where**
‹*read-b-wl-heur kf* ≡ *IsaSAT-Init.read-all-st* (λ- *M* - - - - - - - - - - - - - -. *kf M*)›

context
fixes *R* **and** *kf* **and** *f'* **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** *P*
assumes *not-deleted-code-refine*: ‹(*uncurry* (λ*S C*. *kf C S*), *uncurry* (λ*S C*. *f' C S*)) ∈ [*uncurry* (λ*S C*. *P C S*)]$_a$ *b-assn*^{k} *$*_a$ (*pure R*)^k → *x-assn*›
begin
private lemma *not-deleted-code-refine-b*:
‹(*uncurry15* (λ- *M* - - - - - - - - - - - - - *C*. *kf C M*), *uncurry15* (λ- *M* - - - - - - - - - - - - - *C'*. *f' C'* *M*)) ∈ [*uncurry15* (λ- *M* - - - - - - - - - - - - - *C*. *P C M*)]$_a$
a-assn^k *$*_a$ *b-assn*^{k} *$*_a$ *c-assn*^{k} *$*_a$ *d-assn*^{k} *$*_a$
e-assn^k *$*_a$ *f-assn*^{k} *$*_a$ *g-assn*^{k} *$*_a$ *h-assn*^{k} *$*_a$ *i-assn*^{k} *$*_a$
j-assn^k *$*_a$ *k-assn*^{k} *$*_a$ *l-assn*^{k} *$*_a$ *m-assn*^{k} *$*_a$
n-assn^k *$*_a$ *o-assn*^{k} *$*_a$ (*pure R*)^k → *x-assn*›
apply (*rule add-pure-parameter2*)
apply (*drule remove-pure-parameter2'*[**where** *f* =*kf* **and** *f'*=*f'*, *OF not-deleted-code-refine*])
apply (*drule remove-component-left*[**where** *X* = *a-assn*])
apply (*drule remove-component-right*[**where** *X* = *c-assn*])
apply (*drule remove-component-right*[**where** *X* = *d-assn*])
apply (*drule remove-component-right*[**where** *X* = *e-assn*])
apply (*drule remove-component-right*[**where** *X* = *f-assn*])
apply (*drule remove-component-right*[**where** *X* = *g-assn*])
apply (*drule remove-component-right*[**where** *X* = *h-assn*])
apply (*drule remove-component-right*[**where** *X* = *i-assn*])
apply (*drule remove-component-right*[**where** *X* = *j-assn*])
apply (*drule remove-component-right*[**where** *X* = *k-assn*])
apply (*drule remove-component-right*[**where** *X* = *l-assn*])
apply (*drule remove-component-right*[**where** *X* = *m-assn*])
apply (*drule remove-component-right*[**where** *X* = *n-assn*])
apply (*drule remove-component-right*[**where** *X* = *o-assn*])
apply (*rule hfref-cong*, *assumption*)
apply (*auto simp add*: *uncurry-def*)
done

lemmas *read-b-refine* = *read-all-refine*[*OF not-deleted-code-refine-b*]
lemmas *mop-read-b-refine* = *read-all-mop-refine*[*OF not-deleted-code-refine-b*]
end

context
fixes *R* **and** *kf* **and** *f'* **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*› **and** *P*
assumes *not-deleted-code-refine*: ‹((λ*S*. *kf S*), (λ*S*. *f' S*)) ∈ [(λ*S*. *P S*)]$_a$ *b-assn*^{k} → *x-assn*›
begin

lemmas *read-b-refine0* = *read-b-refine*[*OF not-deleted-code-refine*[*THEN remove-component-right*], *THEN remove-unused-unit-parameter*]
lemmas *mop-read-b-refine0* = *mop-read-b-refine*[*OF not-deleted-code-refine*[*THEN remove-component-right*]]

end

context
fixes R **and** kf **and** f' **and** $x\text{-}assn :: \langle 'r \Rightarrow 'q \Rightarrow assn \rangle$ **and** P
assumes *not-deleted-code-refine*: $\langle (uncurry\ (\lambda S\ C.\ kf\ C\ S),\ uncurry\ (\lambda S\ C.\ f'\ C\ S)) \in [uncurry\ (\lambda S\ C.\ P\ C\ S)]_a\ n\text{-}assn^k *_a (pure\ R)^k \to x\text{-}assn \rangle$
begin
private lemma *not-deleted-code-refine-n*:
$\langle (uncurry15\ (\lambda\text{- - - - - - - - - - - - - - - }M\text{ - }C.\ kf\ C\ M),\ uncurry15\ (\lambda\text{- - - - - - - - - - - - - - - }M\text{ - }C'.\ f'\ C'\ M)) \in [uncurry15\ (\lambda\text{- - - - - - - - - - - - - - - }M\text{ - }C.\ P\ C\ M)]_a$
$a\text{-}assn^k *_a b\text{-}assn^k *_a c\text{-}assn^k *_a d\text{-}assn^k *_a$
$e\text{-}assn^k *_a f\text{-}assn^k *_a g\text{-}assn^k *_a h\text{-}assn^k *_a i\text{-}assn^k *_a$
$j\text{-}assn^k *_a k\text{-}assn^k *_a l\text{-}assn^k *_a m\text{-}assn^k *_a$
$n\text{-}assn^k *_a o\text{-}assn^k *_a (pure\ R)^k \to x\text{-}assn \rangle$
apply (*rule add-pure-parameter2*)
apply (*drule remove-pure-parameter2'*[**where** $f = kf$ **and** $f' = f'$, *OF not-deleted-code-refine*])
apply (*drule remove-component-left*[**where** $X = a\text{-}assn$])
apply (*drule remove-component-middle*[**where** $X = b\text{-}assn$])
apply (*drule remove-component-middle*[**where** $X = c\text{-}assn$])
apply (*drule remove-component-middle*[**where** $X = d\text{-}assn$])
apply (*drule remove-component-middle*[**where** $X = e\text{-}assn$])
apply (*drule remove-component-middle*[**where** $X = f\text{-}assn$])
apply (*drule remove-component-middle*[**where** $X = g\text{-}assn$])
apply (*drule remove-component-middle*[**where** $X = h\text{-}assn$])
apply (*drule remove-component-middle*[**where** $X = i\text{-}assn$])
apply (*drule remove-component-middle*[**where** $X = j\text{-}assn$])
apply (*drule remove-component-middle*[**where** $X = k\text{-}assn$])
apply (*drule remove-component-middle*[**where** $X = l\text{-}assn$])
apply (*drule remove-component-middle*[**where** $X = m\text{-}assn$])
apply (*drule remove-component-right*[**where** $X = o\text{-}assn$])
apply (*rule hfref-cong*, *assumption*)
apply (*auto simp add*: *uncurry-def*)
done

lemmas *read-n-refine* = *read-all-refine*[*OF not-deleted-code-refine-n*]
lemmas *mop-read-n-refine* = *read-all-mop-refine*[*OF not-deleted-code-refine-n*]
end

context
fixes R **and** kf **and** f' **and** $x\text{-}assn :: \langle 'r \Rightarrow 'q \Rightarrow assn \rangle$ **and** P
assumes *not-deleted-code-refine*: $\langle ((\lambda S.\ kf\ S),\ (\lambda S.\ f'\ S)) \in [(\lambda S.\ P\ S)]_a\ n\text{-}assn^k \to x\text{-}assn \rangle$
begin

lemmas *read-n-refine0* = *read-n-refine*[*OF not-deleted-code-refine*[*THEN remove-component-right*], *THEN remove-unused-unit-parameter*]
lemmas *mop-read-n-refine0* = *mop-read-n-refine*[*OF not-deleted-code-refine*[*THEN remove-component-right*]]

end

context
fixes R **and** kf **and** f' **and** $x\text{-}assn :: \langle 'r \Rightarrow 'q \Rightarrow assn \rangle$ **and** P
assumes *not-deleted-code-refine*: $\langle (uncurry\ (\lambda S\ C.\ kf\ C\ S),\ uncurry\ (\lambda S\ C.\ f'\ C\ S)) \in [uncurry\ (\lambda S$

$C.\ P\ C\ S)]_a$ *m-assn*^{k} $*_a$ (*pure R*)^k $\to$ *x-assn*$\rangle$
begin
private lemma *not-deleted-code-refine-m*:
$\langle$(*uncurry15* (λ- - - - - - - - - - - - *M* - - *C*. *kf C M*), *uncurry15* (λ- - - - - - - - - - - - *M* - - *C'*. *f' C' M*)) $\in$ [*uncurry15* (λ- - - - - - - - - - - - *M* - - *C*. *P C M*)]$_a$
a-assn^k $*_a$ *b-assn*^{k} $*_a$ *c-assn*^{k} $*_a$ *d-assn*^{k} $*_a$
e-assn^k $*_a$ *f-assn*^{k} $*_a$ *g-assn*^{k} $*_a$ *h-assn*^{k} $*_a$ *i-assn*^{k} $*_a$
j-assn^k $*_a$ *k-assn*^{k} $*_a$ *l-assn*^{k} $*_a$ *m-assn*^{k} $*_a$
n-assn^k $*_a$ *o-assn*^{k} $*_a$ (*pure R*)^k $\to$ *x-assn*$\rangle$
apply (*rule add-pure-parameter2*)
apply (*drule remove-pure-parameter2'*[**where** *f* =*kf* **and** *f'*=*f'*, *OF not-deleted-code-refine*])
apply (*drule remove-component-left*[**where** *X* = *a-assn*])
apply (*drule remove-component-middle*[**where** *X* = *b-assn*])
apply (*drule remove-component-middle*[**where** *X* = *c-assn*])
apply (*drule remove-component-middle*[**where** *X* = *d-assn*])
apply (*drule remove-component-middle*[**where** *X* = *e-assn*])
apply (*drule remove-component-middle*[**where** *X* = *f-assn*])
apply (*drule remove-component-middle*[**where** *X* = *g-assn*])
apply (*drule remove-component-middle*[**where** *X* = *h-assn*])
apply (*drule remove-component-middle*[**where** *X* = *i-assn*])
apply (*drule remove-component-middle*[**where** *X* = *j-assn*])
apply (*drule remove-component-middle*[**where** *X* = *k-assn*])
apply (*drule remove-component-middle*[**where** *X* = *l-assn*])
apply (*drule remove-component-right*[**where** *X* = *n-assn*])
apply (*drule remove-component-right*[**where** *X* = *o-assn*])
apply (*rule hfref-cong*, *assumption*)
apply (*auto simp add*: *uncurry-def*)
done

lemmas *read-m-refine* = *read-all-refine*[*OF not-deleted-code-refine-m*]
lemmas *mop-read-m-refine* = *read-all-mop-refine*[*OF not-deleted-code-refine-m*]
end

context
fixes *R* **and** *kf* **and** *f'* **and** *x-assn* :: $\langle$'*r* $\Rightarrow$ '*q* $\Rightarrow$ *assn*$\rangle$ **and** *P*
assumes *not-deleted-code-refine*: $\langle$((*λS*. *kf S*), (*λS*. *f' S*)) $\in$ [(*λS*. *P S*)]$_a$ *m-assn*^{k} $\to$ *x-assn*$\rangle$
begin

lemmas *read-m-refine0* = *read-m-refine*[*OF not-deleted-code-refine*[*THEN remove-component-right*], *THEN remove-unused-unit-parameter*]
lemmas *mop-read-m-refine0* = *mop-read-m-refine*[*OF not-deleted-code-refine*[*THEN remove-component-right*]]

end
end
end
theory *IsaSAT-Initialisation-LLVM*
imports *IsaSAT-VMTF-LLVM Watched-Literals.Watched-Literals-Watch-List-Initialisation*
IsaSAT-Initialisation IsaSAT-Setup-LLVM IsaSAT-Mark-LLVM
IsaSAT-Initialisation-State-LLVM
begin
hide-const (**open**) *NEMonad.RETURN NEMonad.ASSERT*

definition *polarity-st-heur-init* :: $\langle$*twl-st-wl-heur-init* $\Rightarrow$ -$\rangle$ **where**
$\langle$*polarity-st-heur-init S L* = *polarity-pol* (*Tuple15-a S*) *L*$\rangle$

definition *polarity-st-heur-init-code* :: ‹*twl-st-wll-trail-init* ⇒ -› **where**
‹*polarity-st-heur-init-code N C* = *IsaSAT-Init.read-all-st-code* (λ*M* - - - - - - - - - - - - - - -. *polarity-pol-fast-code M C*) *N*›

lemma *polarity-st-heur-init-alt-def*:
‹(λ*N C'*. *IsaSAT-Init.read-all-st* (λ*M* - - - - - - - - - - - - - - -. (*RETURN* ∘∘ *polarity-pol*) *M C'*) *N*) = *RETURN oo polarity-st-heur-init*›
by (*auto simp*: *IsaSAT-Init.read-all-st-def polarity-st-heur-init-def intro*!: *ext*
split: *tuple15.splits*)

lemmas *polarity-st-heur-init-code-refine* [*sepref-fr-rules*] =
IsaSAT-Init.read-trail-refine[*OF polarity-pol-fast-code.refine*,
unfolded polarity-st-heur-init-alt-def polarity-st-heur-init-code-def[*symmetric*]]

definition *get-conflict-wl-is-None-init-code* :: ‹*twl-st-wll-trail-init* ⇒ -› **where**
‹*get-conflict-wl-is-None-init-code* = *IsaSAT-Init.read-all-st-code* (λ- - *M* - - - - - - - - - - - - -. *conflict-is-None-code M*)›

lemma *get-conflict-wl-is-None-heur-init-alt-def*:
‹*RETURN o get-conflict-wl-is-None-heur-init* = *IsaSAT-Init.read-all-st* (λ- - *M* - - - - - - - - - - - -. *conflict-is-None M*)›
by (*auto simp*: *IsaSAT-Init.read-all-st-def polarity-st-heur-init-def get-conflict-wl-is-None-heur-init-def*
conflict-is-None-def
intro!: *ext*
split: *tuple15.splits*)

lemmas *get-conflict-wl-is-None-init-code-refine* [*sepref-fr-rules*] =
IsaSAT-Init.read-conflict-refine0[*OF conflict-is-None-code-refine*,
unfolded get-conflict-wl-is-None-init-code-def[*symmetric*] *get-conflict-wl-is-None-heur-init-alt-def*[*symmetric*]]

definition *full-arena-length-st-init* :: ‹*twl-st-wl-heur-init* ⇒ -› **where**
‹*full-arena-length-st-init S* = *length* (*get-clauses-wl-heur-init S*)›

lemma *full-arena-length-st-init-alt-def*:
‹*RETURN o full-arena-length-st-init* = *IsaSAT-Init.read-all-st* (λ- *M* - - - - - - - - - - - - -. (*RETURN* ∘ *length*) *M*)›
by (*auto simp*: *full-arena-length-st-init-def IsaSAT-Init.read-all-st-def*
intro!: *ext*
split: *tuple15.splits*)

definition *full-arena-length-st-init-code* :: ‹*twl-st-wll-trail-init* ⇒ -› **where**
‹*full-arena-length-st-init-code* = *IsaSAT-Init.read-all-st-code* (λ- *M* - - - - - - - - - - - - -. *arena-full-length-impl M*)›

definition *clss-size-lcount-st-init* :: ‹*twl-st-wl-heur-init* ⇒ -› **where**
‹*clss-size-lcount-st-init S* = *clss-size-lcount* (*get-learned-count-init S*)›

lemma *clss-size-lcount-st-init-alt-def*:
‹*RETURN o clss-size-lcount-st-init* = *IsaSAT-Init.read-all-st* (λ- - - - - - - - - - - - - *M* -. (*RETURN* ∘ *clss-size-lcount*) *M*)›
by (*auto simp*: *clss-size-lcount-st-init-def IsaSAT-Init.read-all-st-def*
intro!: *ext*
split: *tuple15.splits*)

definition *clss-size-lcount-st-init-impl* :: ‹*twl-st-wll-trail-init* ⇒ -› **where**

‹*clss-size-lcount-st-init-impl* = *IsaSAT-Init.read-all-st-code* (λ- - - - - - - - - - - - - *M* -. *clss-size-lcount-fast-code* *M*)›

definition *clss-size-lcountUE-st-init* :: ‹*twl-st-wl-heur-init* ⇒ -› **where**
‹*clss-size-lcountUE-st-init S* = *clss-size-lcountUE* (*get-learned-count-init S*)›

lemma *clss-size-lcountUE-st-init-alt-def*:
‹*RETURN o clss-size-lcountUE-st-init* = *IsaSAT-Init.read-all-st* (λ- - - - - - - - - - - - - *M* -. (*RETURN* ∘ *clss-size-lcountUE*) *M*)›
by (*auto simp*: *clss-size-lcountUE-st-init-def IsaSAT-Init.read-all-st-def*
intro!: *ext*
split: *tuple15.splits*)

definition *clss-size-lcountUE-st-init-impl* :: ‹*twl-st-wll-trail-init* ⇒ -› **where**
‹*clss-size-lcountUE-st-init-impl* = *IsaSAT-Init.read-all-st-code* (λ- - - - - - - - - - - - - *M* -. *clss-size-lcountUE-fast-code* *M*)›

definition *clss-size-lcountUEk-st-init* :: ‹*twl-st-wl-heur-init* ⇒ -› **where**
‹*clss-size-lcountUEk-st-init S* = *clss-size-lcountUEk* (*get-learned-count-init S*)›

lemma *clss-size-lcountUEk-st-init-alt-def*:
‹*RETURN o clss-size-lcountUEk-st-init* = *IsaSAT-Init.read-all-st* (λ- - - - - - - - - - - - - *M* -. (*RETURN* ∘ *clss-size-lcountUEk*) *M*)›
by (*auto simp*: *clss-size-lcountUEk-st-init-def IsaSAT-Init.read-all-st-def*
intro!: *ext*
split: *tuple15.splits*)

definition *clss-size-lcountUEk-st-init-impl* :: ‹*twl-st-wll-trail-init* ⇒ -› **where**
‹*clss-size-lcountUEk-st-init-impl* = *IsaSAT-Init.read-all-st-code* (λ- - - - - - - - - - - - - *M* -. *clss-size-lcountUEk-fast-code* *M*)›

definition *clss-size-lcountUS-st-init* :: ‹*twl-st-wl-heur-init* ⇒ -› **where**
‹*clss-size-lcountUS-st-init S* = *clss-size-lcountUS* (*get-learned-count-init S*)›

lemma *clss-size-lcountUS-st-init-alt-def*:
‹*RETURN o clss-size-lcountUS-st-init* = *IsaSAT-Init.read-all-st* (λ- - - - - - - - - - - - - *M* -. (*RETURN* ∘ *clss-size-lcountUS*) *M*)›
by (*auto simp*: *clss-size-lcountUS-st-init-def IsaSAT-Init.read-all-st-def*
intro!: *ext*
split: *tuple15.splits*)

definition *clss-size-lcountUS-st-init-impl* :: ‹*twl-st-wll-trail-init* ⇒ -› **where**
‹*clss-size-lcountUS-st-init-impl* = *IsaSAT-Init.read-all-st-code* (λ- - - - - - - - - - - - - *M* -. *clss-size-lcountUSt-fast-code* *M*)›

definition *clss-size-lcountU0-st-init* :: ‹*twl-st-wl-heur-init* ⇒ -› **where**
‹*clss-size-lcountU0-st-init S* = *clss-size-lcountU0* (*get-learned-count-init S*)›

lemma *clss-size-lcountU0-st-init-alt-def*:
‹*RETURN o clss-size-lcountU0-st-init* = *IsaSAT-Init.read-all-st* (λ- - - - - - - - - - - - - *M* -. (*RETURN* ∘ *clss-size-lcountU0*) *M*)›
by (*auto simp*: *clss-size-lcountU0-st-init-def IsaSAT-Init.read-all-st-def*
intro!: *ext*
split: *tuple15.splits*)

definition *clss-size-lcountU0-st-init-impl* :: ‹*twl-st-wll-trail-init* ⇒ -› **where**

‹clss-size-lcountU0-st-init-impl = IsaSAT-Init.read-all-st-code (λ- - - - - - - - - - - - - M -. clss-size-lcountU0-fast-code M)›

definition *is-failed-loc* :: ‹*bool* ⇒ *bool*› **where**
‹*is-failed-loc x* = *x*›

sepref-def *is-failed-loc-impl*
is ‹*RETURN o is-failed-loc*›
:: ‹*bool1-assn*^{k} →$_a$ *bool1-assn*›
unfolding *is-failed-loc-def*
by *sepref*

lemma *is-failed-heur-init-alt-def*:
‹*RETURN o is-failed-heur-init* = *IsaSAT-Init.read-all-st* (λ- - - - - - - - - - - - - *M* - -. (*RETURN o is-failed-loc*) *M*)›
by (*auto simp*: *is-failed-loc-def IsaSAT-Init.read-all-st-def*
intro!: *ext*
split: *tuple15.splits*)
definition *is-failed-heur-init-impl* :: ‹*twl-st-wll-trail-init* ⇒ -› **where**
‹*is-failed-heur-init-impl* = *IsaSAT-Init.read-all-st-code* (λ- - - - - - - - - - - - - *M* - -. *is-failed-loc-impl M*)›

lemmas [*sepref-fr-rules*] =
IsaSAT-Init.read-b-refine0[*OF arena-full-length-impl.refine*,
unfolded full-arena-length-st-init-code-def[*symmetric*] *full-arena-length-st-init-alt-def*[*symmetric*]]
IsaSAT-Init.read-n-refine0[*OF get-learned-count-number.not-deleted-code-refine*,
unfolded clss-size-lcount-st-init-impl-def[*symmetric*] *clss-size-lcount-st-init-alt-def*[*symmetric*]]
IsaSAT-Init.read-n-refine0[*OF clss-size-lcountUE-fast-code.refine*,
unfolded clss-size-lcountUE-st-init-impl-def[*symmetric*] *clss-size-lcountUE-st-init-alt-def*[*symmetric*]]
IsaSAT-Init.read-n-refine0[*OF clss-size-lcountUEk-fast-code.refine*,
unfolded clss-size-lcountUEk-st-init-impl-def[*symmetric*] *clss-size-lcountUEk-st-init-alt-def*[*symmetric*]]
IsaSAT-Init.read-n-refine0[*OF clss-size-lcountUSt-fast-code.refine*,
unfolded clss-size-lcountUS-st-init-impl-def[*symmetric*] *clss-size-lcountUS-st-init-alt-def*[*symmetric*]]
IsaSAT-Init.read-n-refine0[*OF clss-size-lcountU0-fast-code.refine*,
unfolded clss-size-lcountU0-st-init-impl-def[*symmetric*] *clss-size-lcountU0-st-init-alt-def*[*symmetric*]]
IsaSAT-Init.read-m-refine0[*OF is-failed-loc-impl.refine*,
unfolded is-failed-heur-init-impl-def[*symmetric*] *is-failed-heur-init-alt-def*[*symmetric*]]

lemmas [*unfolded Tuple15-LLVM.inline-direct-return-node-case*, *llvm-code*] =
polarity-st-heur-init-code-def[*unfolded IsaSAT-Init.read-all-st-code-def*]
full-arena-length-st-init-code-def[*unfolded IsaSAT-Init.read-all-st-code-def*]
clss-size-lcount-st-init-impl-def[*unfolded IsaSAT-Init.read-all-st-code-def*]
clss-size-lcountUE-st-init-impl-def[*unfolded IsaSAT-Init.read-all-st-code-def*]
clss-size-lcountUEk-st-init-impl-def[*unfolded IsaSAT-Init.read-all-st-code-def*]
clss-size-lcountUS-st-init-impl-def[*unfolded IsaSAT-Init.read-all-st-code-def*]
clss-size-lcountU0-st-init-impl-def[*unfolded IsaSAT-Init.read-all-st-code-def*]
is-failed-heur-init-impl-def[*unfolded IsaSAT-Init.read-all-st-code-def*]

sepref-register *atoms-hash-empty*
sepref-def (**in** −) *atoms-hash-empty-code*
is ‹*atoms-hash-int-empty*›
:: ‹*sint32-nat-assn*^{k} →$_a$ *atoms-hash-assn*›
unfolding *atoms-hash-int-empty-def array-fold-custom-replicate*
by *sepref*

sepref-def *distinct-atms-empty-code*
  is ‹*distinct-atms-int-empty*›
  :: ‹*sint64-nat-assn*^{k} $\rightarrow_a$ *distinct-atoms-assn*›
  unfolding *distinct-atms-int-empty-def array-fold-custom-replicate*
    al-fold-custom-empty[**where** *'l=64*]
  by *sepref*

lemmas [*sepref-fr-rules*] = *distinct-atms-empty-code.refine atoms-hash-empty-code.refine*

abbreviation *unat-rel32* :: ‹(*32 word* × *nat*) *set*› **where** ‹*unat-rel32* ≡ *unat-rel*›
abbreviation *unat-rel64* :: ‹(*64 word* × *nat*) *set*› **where** ‹*unat-rel64* ≡ *unat-rel*›
abbreviation *snat-rel32* :: ‹(*32 word* × *nat*) *set*› **where** ‹*snat-rel32* ≡ *snat-rel*›
abbreviation *snat-rel64* :: ‹(*64 word* × *nat*) *set*› **where** ‹*snat-rel64* ≡ *snat-rel*›

sepref-def *hp-init-ACIDS0-code*
  is ‹*uncurry hp-init-ACIDS0*›
  :: ‹(*arl64-assn atom-assn*)^k $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *hp-assn*›
  unfolding *hp-init-ACIDS0-def*
    array-fold-custom-replicate op-list-replicate-def[*symmetric*]
    atom.fold-option hp-assn-def
  apply (*annot-unat-const* ‹*TYPE*(*64*)›)
  by *sepref*

definition *init-ACIDS0'* :: ‹- ⇒ *nat* ⇒ (*nat multiset* × *nat multiset* × (*nat* ⇒ *nat*)) *nres*› **where**
  ‹*init-ACIDS0'* $\mathcal{A}$ *n* = *init-ACIDS0* (*mset* $\mathcal{A}$) *n*›

lemma *hp-acids-empty*:
  ‹(*uncurry hp-init-ACIDS0*, *uncurry init-ACIDS0'*) ∈
   Id ×$_f$ *Id* →$_f$ ⟨(((⟨*nat-rel*⟩*option-rel*, ⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel*)) *O*
   acids-encoded-hmrel⟩*nres-rel*›
proof −
  have *1*: ‹((*A*, (λ-. *None*, λ-. *None*, λ-. *None*, λ-. *None*, λ-. *Some 0*), *None*), (*A*, {#}, λ-. *0*)) ∈
acids-encoded-hmrel› **for** $\mathcal{A}$
    by (*auto simp*: *acids-encoded-hmrel-def bottom-acids0-def pairing-heaps-rel-def map-fun-rel-def*
     ACIDS.hmrel-def encoded-hp-prop-list-conc-def encoded-hp-prop-def empty-outside-def empty-acids0-def
      intro!: *relcompI*)
  have *H*: ‹*mset-nodes ya* ≠ {#}› **for** *ya*
    by (*cases ya*) *auto*
  show *?thesis*
    unfolding *uncurry0-def hp-init-ACIDS0-def init-ACIDS0-def*
      init-ACIDS0'-def uncurry-def case-prod-beta
    apply (*intro frefI nres-relI*)
    apply *refine-rcg*
    apply (*rule relcompI*[*of*])
    defer
    apply (*rule 1*)
    by (*auto simp add*: *acids-encoded-hmrel-def encoded-hp-prop-def hp-init-ACIDS0-def*
      ACIDS.hmrel-def encoded-hp-prop-list-conc-def pairing-heaps-rel-def H map-fun-rel-def
      split: *option.splits dest*!: *multi-member-split*)
qed

lemmas [*sepref-fr-rules*] =

hp-init-ACIDS0-code.refine[*FCOMP hp-acids-empty*, *unfolded hr-comp-assoc*[*symmetric*] *acids-assn-def*[*symmetric*]]

definition *init-ACIDS'* **where**
‹*init-ACIDS' A n = do* {
ac ← *init-ACIDS0' A n*;
RETURN (*ac*, *0*)
}›

lemma *init-ACIDS'-alt*: ‹*init-ACIDS* (*mset N*) *n* = *init-ACIDS' N n*›
by (*auto simp*: *init-ACIDS'-def init-ACIDS0'-def init-ACIDS-def*)

sepref-register *init-ACIDS0' acids-heur-import-variable*

sepref-def *hp-init-ACIDS-code*
is ‹*uncurry init-ACIDS'*›
:: ‹(*arl64-assn atom-assn*)^k *∗$_a$ sint64-nat-assn*^{k} →$_a$ *acids-assn2*›
unfolding *init-ACIDS'-def acids-assn2-def*
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *acids-heur-import-variable-code*
is ‹*uncurry acids-heur-import-variable*›
:: ‹*atom-assn*^{k} *∗$_a$ acids-assn2*^{d} →$_a$ *acids-assn2*›
unfolding *acids-heur-import-variable-def acids-assn2-def*
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *initialise-ACIDS-code*
is ‹*uncurry initialise-ACIDS*›
:: ‹[λ(*N*, *n*). *True*]$_a$ (*arl64-assn atom-assn*)^k *∗$_a$ sint64-nat-assn*^{k} → *acids-assn2*›
unfolding *initialise-ACIDS-def vmtf-cons-def Suc-eq-plus1 atom.fold-option length-uint32-nat-def option.case-eq-if vmtf-init-assn-def init-ACIDS'-alt*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
supply [[*goals-limit* = *1*]]
by *sepref*

lemma *initialise-ACIDS-rev-alt-def*:
‹*initialise-ACIDS-rev N n = do* {
A ← *init-ACIDS* (*mset N*) *n*;
ASSERT(*length N* ≤ *unat32-max*);
(*n*, *A*) ← *WHILE$_T$* λ-. *True*
(λ(*i*, *A*). *i* < *length-uint32-nat N*)
(λ(*i*, *A*). *do* {
ASSERT(*i* < *length-uint32-nat N*);
let L = (*N* ! (*length N* − *1* − *i*));
ASSERT (*snd A* = *i*);
ASSERT(*i* + *1* ≤ *unat32-max*);
A ← *acids-heur-import-variable L A*;
RETURN (*i* + *1*, *A*)
})
(*0*, *A*);
RETURN A
}› (**is** ‹*?A* = *?B*›)
proof −

have [*refine0*]: ‹*init-ACIDS* (*mset* (*rev N*)) *n* ≤ ⇓ *Id* (*init-ACIDS* (*mset N*) *n*)›
‹*init-ACIDS* (*mset* (*N*)) *n* ≤ ⇓ *Id* (*init-ACIDS* (*mset* (*rev N*)) *n*)›
by *auto*
have [*refine0*]: ‹(*A*,*Aa*)∈*Id* ⟹ ((*0*, *A*), *0*, *Aa*) ∈ *Id* ×$_r$ *Id*› **for** *A Aa*
by *auto*
have *K*: ‹⋀*A Aa x x'* *x1 x2 x1a x2a*.
(*x*, *x'*) ∈ *nat-rel* ×$_f$ *Id* ⟹
case x of (*i*, *A*) ⇒ *i* < *length-uint32-nat* (*rev N*) ⟹
case x' *of* (*i*, *A*) ⇒ *i* < *length-uint32-nat N* ⟹
x' = (*x1*, *x2*) ⟹
x = (*x1a*, *x2a*) ⟹
x1 < *length-uint32-nat N* ⟹
x1a < *length-uint32-nat* (*rev N*) ⟹
snd x2 = *x1* ⟹
x1 + *1* ≤ *unat32-max* ⟹
snd x2a = *x1a* ⟹
x1a + *1* ≤ *unat32-max* ⟹
acids-heur-import-variable (*rev N* ! *x1a*) *x2a*
≤ ⇓ *Id*
(*acids-heur-import-variable* (*N* ! (*length N* − *1* − *x1*)) *x2*)›
‹⋀*A Aa x x'* *x1 x2 x1a x2a*.
(*x*, *x'*) ∈ *nat-rel* ×$_f$ *Id* ⟹
case x of (*i*, *A*) ⇒ *i* < *length-uint32-nat N* ⟹
case x' *of* (*i*, *A*) ⇒ *i* < *length-uint32-nat* (*rev N*) ⟹
x' = (*x1*, *x2*) ⟹
x = (*x1a*, *x2a*) ⟹
x1 < *length-uint32-nat N* ⟹
x1a < *length-uint32-nat* (*rev N*) ⟹
snd x2 = *x1* ⟹
x1 + *1* ≤ *unat32-max* ⟹
snd x2a = *x1a* ⟹
x1a + *1* ≤ *unat32-max* ⟹
acids-heur-import-variable (*N* ! (*length N* − *1* − *x1a*)) *x2a*
≤ ⇓ *Id*
(*acids-heur-import-variable* (*rev N* ! *x1*) *x2*)›
by (*auto simp*: *nth-rev*)
have ‹*?A* ≤ ⇓*Id ?B*›
unfolding *initialise-ACIDS-rev-def initialise-ACIDS-def*
apply *refine-vcg*
apply (*auto simp*: *rev-nth*; *fail*)+
apply (*rule K*; *assumption*)
apply (*auto simp*: *rev-nth*; *fail*)+
done
moreover have ‹*?B* ≤ ⇓*Id ?A*›
unfolding *initialise-ACIDS-rev-def initialise-ACIDS-def*
apply *refine-vcg*
apply (*auto simp*: *rev-nth*; *fail*)+
apply (*rule K*; *assumption?*)
apply (*auto simp*: *rev-nth*; *fail*)+
done
ultimately show *?thesis*
by *auto*
qed

sepref-def *initialise-ACIDS-rev-code*
is ‹*uncurry initialise-ACIDS-rev*›

:: ‹$[\lambda(N,\ n).\ True]_a$ $(arl64\text{-}assn\ atom\text{-}assn)^k *_a sint64\text{-}nat\text{-}assn^k \to acids\text{-}assn2$›
unfolding *vmtf-cons-def Suc-eq-plus1 atom.fold-option length-uint32-nat-def option.case-eq-if vmtf-init-assn-def init-ACIDS'-alt initialise-ACIDS-rev-alt-def nth-rev*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-def *initialise-VMTF-code*
is ‹*uncurry initialise-VMTF*›
:: ‹$[\lambda(N,\ n).\ True]_a$ $(arl64\text{-}assn\ atom\text{-}assn)^k *_a sint64\text{-}nat\text{-}assn^k \to vmtf\text{-}init\text{-}assn$›
unfolding *initialise-VMTF-def vmtf-cons-def Suc-eq-plus1 atom.fold-option length-uint32-nat-def option.case-eq-if vmtf-init-assn-def*
apply (*rewrite* **in** ‹*let* - = ⌑ *in* - › *array-fold-custom-replicate op-list-replicate-def*[*symmetric*])
apply (*rewrite at 0* **in** ‹*VMTF-Node* ⌑› *unat-const-fold*[**where** *'a=64*])
apply (*rewrite at* ‹*VMTF-Node* (⌑ + *1*)› *annot-snat-unat-conv*)
apply (*rewrite at 1* **in** ‹*VMTF-Node* ⌑› *unat-const-fold*[**where** *'a=64*])
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
apply (*rewrite* **in** ‹*list-update* - - -› *annot-index-of-atm*)
apply (*rewrite* **in** ‹*if* - *then* - *else list-update* - - -› *annot-index-of-atm*)
apply (*rewrite at* ‹⌑› **in** ‹- ! *atom.the* -› *annot-index-of-atm*)+
apply (*rewrite at* ‹*RETURN* ((-, ⌑, -))› *annot-snat-unat-conv*)
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-register *initialize-Bump-Init*
sepref-def *initialize-Bump-Init-code*
is ‹*uncurry initialize-Bump-Init*›
:: ‹$[\lambda(N,\ n).\ True]_a$ $(arl64\text{-}assn\ atom\text{-}assn)^k *_a sint64\text{-}nat\text{-}assn^k \to heuristic\text{-}bump\text{-}init\text{-}assn$›
unfolding *initialize-Bump-Init-def*
by *sepref*

sepref-register *cons-trail-Propagated-tr*

lemma *propagate-unit-cls-heur-b-alt-def*:
‹*propagate-unit-cls-heur-b L S* =
do {
let (*M*, *S*) = *extract-trail-wl-heur-init S*;
M ← *cons-trail-Propagated-tr L 0 M*;
RETURN (*IsaSAT-Init.update-a M S*)
}›
by (*cases S*)
(*auto simp*:*propagate-unit-cls-heur-b-def propagate-unit-cls-heur-def isasat-init-getters-and-setters-def Let-def intro*!: *ext*)

sepref-def *propagate-unit-cls-code*
is ‹*uncurry* (*propagate-unit-cls-heur-b*)›
:: ‹$unat\text{-}lit\text{-}assn^k *_a isasat\text{-}init\text{-}assn^d \to_a isasat\text{-}init\text{-}assn$›
supply [[*goals-limit=1*]] *DECISION-REASON-def*[*simp*]
unfolding *propagate-unit-cls-heur-b-alt-def PR-CONST-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

declare *propagate-unit-cls-code.refine*[*sepref-fr-rules*]

definition *already-propagated-unit-cls-heur'*
:: ‹*bool* ⇒ *twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*›
where
‹*already-propagated-unit-cls-heur'* = (λ*unbdd S*. *RETURN S*)›

lemma *already-propagated-unit-cls-heur'-alt*:
‹*already-propagated-unit-cls-heur unbd L* = *already-propagated-unit-cls-heur' unbd*›
unfolding *already-propagated-unit-cls-heur'-def already-propagated-unit-cls-heur-def*
by *auto*

definition *already-propagated-unit-cls-heur-b* **where**
‹*already-propagated-unit-cls-heur-b* = *already-propagated-unit-cls-heur' False*›

sepref-def *already-propagated-unit-cls-code*
is ‹*already-propagated-unit-cls-heur-b*›
:: ‹*isasat-init-assn*^{d} →$_a$ *isasat-init-assn*›
supply [[*goals-limit=1*]]
unfolding *already-propagated-unit-cls-heur'-def*
PR-CONST-def already-propagated-unit-cls-heur-b-def
by *sepref*

sepref-def *set-conflict-unit-code*
is ‹*uncurry set-conflict-unit-heur*›
:: ‹[λ(*L*, (*b*, *n*, *xs*)). *atm-of L* < *length xs*]$_a$
unat-lit-assn^k *$*$_a$ *conflict-option-rel-assn*^{d} → *conflict-option-rel-assn*›
supply [[*goals-limit=1*]]
unfolding *set-conflict-unit-heur-def ISIN-def*[*symmetric*] *conflict-option-rel-assn-def*
lookup-clause-rel-assn-def
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

declare *set-conflict-unit-code.refine*[*sepref-fr-rules*]

lemma *conflict-propagated-unit-cls-heur-b-alt-def*:
‹*conflict-propagated-unit-cls-heur-b L S* =
do {
let (*D*, *S*) = *extract-conflict-wl-heur-init S*;
let (*M*, *S*) = *extract-trail-wl-heur-init S*;
Refine-Basic.ASSERT(*atm-of L* < *length* (*snd* (*snd D*)));
D ← *set-conflict-unit-heur L D*;
Refine-Basic.ASSERT(*isa-length-trail-pre M*);
let j = *isa-length-trail M*;
RETURN (*IsaSAT-Init.update-d j* (*IsaSAT-Init.update-c D* (*IsaSAT-Init.update-a M S*)))
}›
by (*cases S*)
(*auto simp*: *isasat-init-getters-and-setters-def conflict-propagated-unit-cls-heur-b-def*
conflict-propagated-unit-cls-heur-def)

sepref-def *conflict-propagated-unit-cls-code*
is ‹*uncurry* (*conflict-propagated-unit-cls-heur-b*)›
:: ‹*unat-lit-assn*^{k} *$*$_a$ *isasat-init-assn*^{d} →$_a$ *isasat-init-assn*›

supply [[*goals-limit=1*]]
unfolding *conflict-propagated-unit-cls-heur-b-alt-def*
PR-CONST-def
by *sepref*

declare *conflict-propagated-unit-cls-code.refine*[*sepref-fr-rules*]

sepref-register *fm-add-new*

lemma *add-init-cls-code-bI*:
assumes
‹*length at'* ≤ *Suc* (*Suc unat32-max*)› **and**
‹*2* ≤ *length at'*› **and**
‹*length a1'j* ≤ *length a1'a*› **and**
‹*length a1'a* ≤ *snat64-max* − *length at'* − *5*›
shows ‹*append-and-length-fast-code-pre* ((*True*, *at'*), *a1'a*)› ‹*5* ≤ *snat64-max* − *length at'*›
using *assms* **unfolding** *append-and-length-fast-code-pre-def*
by (*auto simp*: *unat64-max-def unat32-max-def snat64-max-def*)

lemma *add-init-cls-code-bI2*:
assumes
‹*length at'* ≤ *Suc* (*Suc unat32-max*)›
shows ‹*5* ≤ *snat64-max* − *length at'*›
using *assms* **unfolding** *append-and-length-fast-code-pre-def*
by (*auto simp*: *unat64-max-def unat32-max-def snat64-max-def*)

lemma *add-init-clss-codebI*:
assumes
‹*length at'* ≤ *Suc* (*Suc unat32-max*)› **and**
‹*2* ≤ *length at'*› **and**
‹*length a1'j* ≤ *length a1'a*› **and**
‹*length a1'a* ≤ *unat64-max* − (*length at'* + *5*)›
shows ‹*length a1'j* < *unat64-max*›
using *assms* **by** (*auto simp*: *unat64-max-def unat32-max-def*)

abbreviation *clauses-ll-assn* **where**
‹*clauses-ll-assn* ≡ *aal-assn'* *TYPE*(*64*) *TYPE*(*64*) *unat-lit-assn*›

lemma *op-list-list-llen-alt-def*: ‹*op-list-list-llen xss i* = *length* (*xss* ! *i*)›
unfolding *op-list-list-llen-def*
by *auto*

lemma *op-list-list-idx-alt-def*: ‹*op-list-list-idx xs i j* = *xs* ! *i* ! *j*›
unfolding *op-list-list-idx-def* **..**

sepref-def *append-and-length-fast-code*
is ‹*uncurry2 fm-add-new-fast*›
:: ‹[λ((*b*, *C*), *N*). *append-and-length-fast-code-pre* ((*b*, *C*), *N*)]$_a$
bool1-assn^k *$*_a$ *clause-ll-assn*^{k} *$*_a$ (*arena-fast-assn*)^d →
arena-fast-assn ×$_a$ *sint64-nat-assn*›
supply [[*goals-limit=1*]]
supply [*simp*] = *fm-add-new-bounds1*[*simplified*] *shorten-lbd-le*
supply [*split*] = *if-splits*

unfolding *fm-add-new-fast-def fm-add-new-def append-and-length-fast-code-pre-def*
op-list-list-llen-alt-def[*symmetric*] *op-list-list-idx-alt-def*[*symmetric*]
is-short-clause-def header-size-def
apply (*rewrite at* ‹*APos* ⌑› *unat-const-fold*[**where** *'a=32*])+
apply (*rewrite at* ‹*length - - 2*› *annot-snat-unat-downcast*[**where** *'l=32*])
apply (*rewrite at* ‹*AStatus - ⌑*› *unat-const-fold*[**where** *'a=2*])+
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-register *fm-add-new-fast*

lemma *add-init-cls-heur-b-alt-def*:
‹*add-init-cls-heur-b C S* = *do* {
let C = *C*;
ASSERT(*length C* ≤ *unat32-max* + *2*);
ASSERT(*length C* ≥ *2*);
let (*N*, *S*) = *extract-arena-wl-heur-init S*;
let (*failed*, *S*) = *extract-failed-wl-heur-init S*;
if (*length N* ≤ *snat64-max* − *length C* − *5* ∧ ¬*failed*)
then do {
let (*vdom*, *S*) = *extract-vdom-wl-heur-init S*;
let (*ivdom*, *S*) = *extract-ivdom-wl-heur-init S*;
ASSERT(*length vdom* ≤ *length N* ∧ *vdom* = *ivdom*);
(*N*, *i*) ← *fm-add-new True C N*;
let vdom = *vdom* @ [*i*];
let ivdom = *ivdom* @ [*i*];
RETURN (*IsaSAT-Init.update-b N* (*IsaSAT-Init.update-k vdom* (*IsaSAT-Init.update-l ivdom* (*IsaSAT-Init.update-m failed S*))))
} *else RETURN* (*IsaSAT-Init.update-m True* (*IsaSAT-Init.update-b N S*))}›
by (*cases S*)
(*auto simp*: *isasat-init-getters-and-setters-def conflict-propagated-unit-cls-heur-b-def add-init-cls-heur-b-def add-init-cls-heur-def*
conflict-propagated-unit-cls-heur-def)

sepref-def *add-init-cls-code-b*
is ‹*uncurry add-init-cls-heur-b*›
:: ‹[*λ*(*C*, *S*). *True*]$_a$
(*clause-ll-assn*)^k *$*_a$ *isasat-init-assn*^{d} → *isasat-init-assn*›
supply [[*goals-limit=1*]] *append-ll-def*[*simp*] *add-init-clss-codebI*[*intro*]
add-init-cls-code-bI[*intro*] *add-init-cls-code-bI2*[*intro*]
unfolding *add-init-cls-heur-b-alt-def*
PR-CONST-def
Let-def length-uint64-nat-def add-init-cls-heur-b'-def
op-list-list-llen-alt-def[*symmetric*] *op-list-list-idx-alt-def*[*symmetric*]
unfolding
nth-rll-def[*symmetric*] *delete-index-and-swap-update-def*[*symmetric*]
delete-index-and-swap-ll-def[*symmetric*]
append-ll-def[*symmetric*] *fm-add-new-fast-def*[*symmetric*]
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *already-propagated-unit-cls-conflict-heur-b-alt-def*:
‹*already-propagated-unit-cls-conflict-heur-b L S* = *do* {
ASSERT (*isa-length-trail-pre* (*get-trail-init-wl-heur S*));
let (*M*, *S*) = *extract-trail-wl-heur-init S*;
let j = *isa-length-trail M*;

RETURN (*IsaSAT-Init.update-d j* (*IsaSAT-Init.update-a M S*))
}›
by (*cases S*)
(*auto simp*: *isasat-init-getters-and-setters-def conflict-propagated-unit-cls-heur-b-def add-init-cls-heur-b-def add-init-cls-heur-def*
already-propagated-unit-cls-conflict-heur-def already-propagated-unit-cls-conflict-heur-b-def)

sepref-def *already-propagated-unit-cls-conflict-code*
is ‹*uncurry already-propagated-unit-cls-conflict-heur-b*›
:: ‹*unat-lit-assn*^{k} $*_a$ *isasat-init-assn*^{d} $\rightarrow_a$ *isasat-init-assn*›
supply [[*goals-limit=1*]]
unfolding *already-propagated-unit-cls-conflict-heur-b-alt-def PR-CONST-def*
by *sepref*

sepref-def (**in** −) *set-conflict-empty-code*
is ‹*RETURN o lookup-set-conflict-empty*›
:: ‹*conflict-option-rel-assn*^{d} $\rightarrow_a$ *conflict-option-rel-assn*›
supply [[*goals-limit=1*]]
unfolding *lookup-set-conflict-empty-def conflict-option-rel-assn-def*
by *sepref*

definition *set-conflict-to-empty* **where**
‹*set-conflict-to-empty* = (λ(-, *nxs*). (*False*, *nxs*))›

sepref-def *set-conflict-to-empty-impl*
is ‹*RETURN o set-conflict-to-empty*›
:: ‹*conflict-option-rel-assn*^{d} $\rightarrow_a$ *conflict-option-rel-assn*›
unfolding *set-conflict-to-empty-def conflict-option-rel-assn-def*
by *sepref*

lemma *set-empty-clause-as-conflict-heur-alt-def*:
‹*set-empty-clause-as-conflict-heur S* = (*do* {
let (*M*, *S*) = *extract-trail-wl-heur-init S*;
let (*D*, *S*) = *extract-conflict-wl-heur-init S*;
ASSERT(*isa-length-trail-pre M*);
let j = *isa-length-trail M*;
RETURN (*IsaSAT-Init.update-c* (*set-conflict-to-empty D*) (*IsaSAT-Init.update-d j* (*IsaSAT-Init.update-a M S*)))
})›
by (*cases S*)
(*auto simp*: *isasat-init-getters-and-setters-def conflict-propagated-unit-cls-heur-b-def set-conflict-to-empty-def*
set-empty-clause-as-conflict-heur-def already-propagated-unit-cls-conflict-heur-b-def)

sepref-def *set-empty-clause-as-conflict-code*
is ‹*set-empty-clause-as-conflict-heur*›
:: ‹*isasat-init-assn*^{d} $\rightarrow_a$ *isasat-init-assn*›
supply [[*goals-limit=1*]]
unfolding *set-empty-clause-as-conflict-heur-alt-def lookup-clause-rel-assn-def*
by *sepref*

definition (**in** −) *add-clause-to-others-heur′*
:: ‹*twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*add-clause-to-others-heur′* = (λ *S*.

RETURN S)›

lemma *add-clause-to-others-heur'-alt*: ‹*add-clause-to-others-heur L* = *add-clause-to-others-heur'*›
unfolding *add-clause-to-others-heur'-def add-clause-to-others-heur-def*
by *auto*

sepref-def *add-clause-to-others-code*
is ‹*add-clause-to-others-heur'*›
:: ‹*isasat-init-assn*^{d} →$_a$ *isasat-init-assn*›
supply [[*goals-limit=1*]]
unfolding *add-clause-to-others-heur-def add-clause-to-others-heur'-def*
by *sepref*

declare *add-clause-to-others-code.refine*[*sepref-fr-rules*]

sepref-register *init-dt-step-wl*
get-conflict-wl-is-None-heur-init already-propagated-unit-cls-heur
conflict-propagated-unit-cls-heur add-clause-to-others-heur
add-init-cls-heur set-empty-clause-as-conflict-heur

sepref-register *polarity-st-heur-init propagate-unit-cls-heur*

lemma *is-Nil-length*: ‹*is-Nil xs* ⟷ *length xs* = *0*›
by (*cases xs*) *auto*

definition *pre-simplify-clause-lookup'* **where**
‹*pre-simplify-clause-lookup' i xs* = *pre-simplify-clause-lookup* (*xs* ! *i*)›

lemma *pre-simplify-clause-lookup'I*:
‹*a* < *length bb* ⟹
a1' < *length* (*bb* ! *a*) ⟹
rdomp (*aal-assn'* *TYPE*(*64*) *TYPE*(*64*) *unat-lit-assn*) *bb* ⟹
Suc a1' < *max-snat 64*›
for *aa aaa ad ag*::‹*64 word*› **and** *ac* :: ‹*32 word*› **and** *ae af* :: ‹*1 word*›
by (*auto dest*!: *aal-assn-boundsD'* *bspec*[*of* - - ‹*bb* ! *a*›])

sepref-def *pre-simplify-clause-lookup-impl*
is ‹*uncurry3 pre-simplify-clause-lookup'*›
:: ‹[λ(((*i*,*xs*),-),-). *i* < *length xs*]$_a$
sint64-nat-assn^k *$*$_a$ *clauses-ll-assn*^{k} *$*$_a$ *clause-ll-assn*^{d} *$*$_a$ *marked-struct-assn*^{d} →
bool1-assn ×$_a$ *clause-ll-assn* ×$_a$ *marked-struct-assn*›
supply [*intro*] = *pre-simplify-clause-lookup'I*
unfolding *pre-simplify-clause-lookup-def pre-simplify-clause-lookup'-def*
op-list-list-llen-alt-def[*symmetric*] *op-list-list-idx-alt-def*[*symmetric*]
by (*annot-snat-const* ‹*TYPE*(*64*)›)
sepref

definition *pre-simplify-clause-lookup-st'* **where**
‹*pre-simplify-clause-lookup-st' i xs* = *pre-simplify-clause-lookup-st* (*xs* ! *i*)›

lemma *pre-simplify-clause-lookup-st-alt-def*:
‹*pre-simplify-clause-lookup-st* = (λ*C E* S_0. *do* {
let (*mark*, *S*) = *extract-marked-wl-heur-init* S_0;
(*tauto*, *C*, *mark*) ← *pre-simplify-clause-lookup C E mark*;
RETURN (*tauto*, *C*, (*IsaSAT-Init.update-o mark S*))
})›

by (*auto simp*: *isasat-init-getters-and-setters-def pre-simplify-clause-lookup-st-def*
intro!: *ext split*: *tuple15.splits*)

sepref-register *pre-simplify-clause-lookup′ pre-simplify-clause-lookup-st′*

sepref-def *pre-simplify-clause-lookup-st-impl*
is ‹*uncurry3 pre-simplify-clause-lookup-st′*›
:: ‹[λ(((*i*,*xs*),-),-). *i* < *length xs*]$_a$
sint64-nat-assn^k *$*_a$ *clauses-ll-assn*^{k} *$*_a$ *clause-ll-assn*^{d} *$*_a$ *isasat-init-assn*^{d} →
bool1-assn ×$_a$ *clause-ll-assn* ×$_a$ *isasat-init-assn*›
unfolding *pre-simplify-clause-lookup-st-alt-def*
fold-tuple-optimizations pre-simplify-clause-lookup-st′-def
op-list-list-llen-alt-def[*symmetric*] *op-list-list-idx-alt-def*[*symmetric*]
pre-simplify-clause-lookup′-def[*symmetric*]
by *sepref*

definition *init-dt-step-wl-heur-b′*
:: ‹*nat clause-l list* ⇒ *nat* ⇒ *twl-st-wl-heur-init* × - ⇒ (*twl-st-wl-heur-init* × -) *nres*› **where**
‹*init-dt-step-wl-heur-b′ C i* = *init-dt-step-wl-heur-b* (*C*!*i*)›

definition *add-tautology-to-clauses′* **where**
‹*add-tautology-to-clauses′* = (λ*S*.
RETURN S)›

lemma *add-tautology-to-clauses-alt-def*:
‹*add-tautology-to-clauses C S* = *add-tautology-to-clauses′ S*›
by (*cases S*) (*auto simp*: *add-tautology-to-clauses′-def add-tautology-to-clauses-def*)

sepref-def *add-tautology-to-clauses′-impl*
is *add-tautology-to-clauses′*
:: ‹*isasat-init-assn*^{d} →$_a$ *isasat-init-assn*›
unfolding *add-tautology-to-clauses′-def*
by *sepref*

sepref-def *init-dt-step-wl-code-b*
is ‹*uncurry2* (*init-dt-step-wl-heur-b′*)›
:: ‹[λ((*xs*, *i*), *S*). *i* < *length xs*]$_a$
(*clauses-ll-assn*)^k *$*_a$ *sint64-nat-assn*^{k} *$*_a$ (*isasat-init-assn* ×$_a$ *clause-ll-assn*)^d →
isasat-init-assn ×$_a$ *clause-ll-assn*›
supply [[*goals-limit*=*1*]]
supply *polarity-None-undefined-lit*[*simp*] *polarity-st-init-def*[*simp*]
option.splits[*split*] *get-conflict-wl-is-None-heur-init-alt-def*[*simp*]
tri-bool-eq-def[*simp*]
unfolding *init-dt-step-wl-heur-def PR-CONST-def*
init-dt-step-wl-heur-b-def
list-length-1-def is-Nil-length init-dt-step-wl-heur-b′-def
op-list-list-llen-alt-def[*symmetric*] *op-list-list-idx-alt-def*[*symmetric*]
already-propagated-unit-cls-heur′-alt
add-clause-to-others-heur′-def[*symmetric*]
add-clause-to-others-heur′-alt
already-propagated-unit-cls-heur-b-def[*symmetric*]
propagate-unit-cls-heur-b-def[*symmetric*]
conflict-propagated-unit-cls-heur-b-def[*symmetric*]
pre-simplify-clause-lookup-st′-def[*symmetric*]
add-tautology-to-clauses-alt-def
add-init-cls-heur-b-def[*symmetric*]

unfolding *watched-app-def*[*symmetric*]
unfolding *nth-rll-def*[*symmetric*]
unfolding *is-Nil-length get-conflict-wl-is-None-init*
polarity-st-heur-init-alt-def[*symmetric*]
get-conflict-wl-is-None-heur-init-alt-def[*symmetric*]
SET-TRUE-def[*symmetric*] *SET-FALSE-def*[*symmetric*] *UNSET-def*[*symmetric*]
tri-bool-eq-def[*symmetric*] *polarity-st-heur-init-def*[*symmetric*]
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-register *init-dt-wl-heur-unb*

abbreviation *isasat-atms-ext-rel-assn* **where**
‹*isasat-atms-ext-rel-assn* ≡ *larray64-assn uint64-nat-assn* ×$_a$ *uint32-nat-assn* ×$_a$
arl64-assn atom-assn›

abbreviation *nat-lit-list-hm-assn* **where**
‹*nat-lit-list-hm-assn* ≡ *hr-comp isasat-atms-ext-rel-assn isasat-atms-ext-rel*›

sepref-def *init-next-size-impl*
is ‹*RETURN o init-next-size*›
:: ‹[λ*L*. *L* ≤ *unat32-max div 2*]$_a$ *sint64-nat-assn*^{k} → *sint64-nat-assn*›
unfolding *init-next-size-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *nat-lit-lits-init-assn-assn-in*
is ‹*uncurry add-to-atms-ext*›
:: ‹*atom-assn*^{k} ∗$_a$ *isasat-atms-ext-rel-assn*^{d} →$_a$ *isasat-atms-ext-rel-assn*›
supply [[*goals-limit=1*]]
unfolding *add-to-atms-ext-def length-uint32-nat-def*
apply (*rewrite at* ‹*max* ⌑ -› *value-of-atm-def*[*symmetric*])
apply (*rewrite at* ‹⌑ < -› *value-of-atm-def*[*symmetric*])
apply (*rewrite at* ‹*list-grow* - (*init-next-size* ⌑)› *value-of-atm-def*[*symmetric*])
apply (*rewrite at* ‹*list-grow* - (*init-next-size* ⌑)› *index-of-atm-def*[*symmetric*])
apply (*rewrite at* ‹⌑ < -› *annot-unat-unat-upcast*[**where** *'l=64*])
unfolding *max-def list-grow-alt*
op-list-grow-init'-alt
apply (*annot-all-atm-idxs*)
apply (*rewrite at* ‹*op-list-grow-init* ⌑› *unat-const-fold*[**where** *'a=64*])
apply (*rewrite at* ‹- < ⌑› *annot-snat-unat-conv*)
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma [*sepref-fr-rules*]:
‹(*uncurry nat-lit-lits-init-assn-assn-in*, *uncurry* (*RETURN* ∘∘ *op-set-insert*))
∈ [λ(*a*, *b*). *a* ≤ *unat32-max div 2*]$_a$
atom-assn^k ∗$_a$ *nat-lit-list-hm-assn*^{d} → *nat-lit-list-hm-assn*›
by (*rule nat-lit-lits-init-assn-assn-in.refine*[*FCOMP add-to-atms-ext-op-set-insert*
[*unfolded op-set-insert-def*[*symmetric*]]])
hide-const (**open**) *NEMonad.ASSERT NEMonad.RETURN*

lemma *while-nfoldli*:
do {
(-,σ) $\leftarrow$ $WHILE_T$ (*FOREACH-cond c*) (λx. *do* {*ASSERT* (*FOREACH-cond c x*); *FOREACH-body f x*}) (*l*,σ);
RETURN σ
} $\leq$ *nfoldli l c f* σ
apply (*induct l arbitrary*: σ)
apply (*subst WHILET-unfold*)
apply (*simp add*: *FOREACH-cond-def*)

apply (*subst WHILET-unfold*)
apply (*auto*
simp: *FOREACH-cond-def FOREACH-body-def*
intro: *bind-mono Refine-Basic.bind-mono*(*1*))
done

definition *extract-atms-cls-i'* **where**
‹*extract-atms-cls-i' C i* = *extract-atms-cls-i* (*C*!*i*)›

sepref-def *extract-atms-cls-imp*
is ‹*uncurry2 extract-atms-cls-i'*›
:: ‹[λ((*N*, *i*), -). *i* < *length N*]$_a$
(*clauses-ll-assn*)^k *$*_a$ *sint64-nat-assn*^{k} *$*_a$ *nat-lit-list-hm-assn*^{d} $\rightarrow$ *nat-lit-list-hm-assn*›
supply [*dest*!] = *aal-assn-boundsD'*
unfolding *extract-atms-cls-i-def extract-atms-cls-i'-def*
apply (*subst nfoldli-by-idx*[*abs-def*])
unfolding *nfoldli-upt-by-while*
op-list-list-llen-alt-def[*symmetric*] *op-list-list-idx-alt-def*[*symmetric*]
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

declare *extract-atms-cls-imp.refine*[*sepref-fr-rules*]

sepref-def *extract-atms-clss-imp*
is ‹*uncurry extract-atms-clss-i*›
:: ‹(*clauses-ll-assn*)^k *$*_a$ *nat-lit-list-hm-assn*^{d} $\rightarrow_a$ *nat-lit-list-hm-assn*›
supply [*dest*] = *aal-assn-boundsD'*
unfolding *extract-atms-clss-i-def*
apply (*subst nfoldli-by-idx*)
unfolding *nfoldli-upt-by-while Let-def extract-atms-cls-i'-def*[*symmetric*]
op-list-list-llen-alt-def[*symmetric*] *op-list-list-idx-alt-def*[*symmetric*]
op-list-list-len-def[*symmetric*]
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *extract-atms-clss-hnr*[*sepref-fr-rules*]:
‹(*uncurry extract-atms-clss-imp*, *uncurry* (*RETURN* $\circ\circ$ *extract-atms-clss*))
$\in$ [λ(*a*, *b*). $\forall$ *C*$\in$*set a*. $\forall$ *L*$\in$*set C*. *nat-of-lit L* $\leq$ *unat32-max*]$_a$
(*clauses-ll-assn*)^k *$*_a$ *nat-lit-list-hm-assn*^{d} $\rightarrow$ *nat-lit-list-hm-assn*›
using *extract-atms-clss-imp.refine*[*FCOMP extract-atms-clss-i-extract-atms-clss*]
by *simp*

sepref-def *extract-atms-clss-imp-empty-assn*
is ‹*uncurry0 extract-atms-clss-imp-empty-rel*›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *isasat-atms-ext-rel-assn*›

unfolding *extract-atms-clss-imp-empty-rel-def*
larray-fold-custom-replicate
supply [[*goals-limit=1*]]
apply (*rewrite at* ‹(-, -, ¤)› *al-fold-custom-empty*[**where** *'l=64*])
apply (*rewrite* **in** ‹(¤, -, -)› *annotate-assn*[**where** *A=‹larray64-assn uint64-nat-assn›*])
apply (*rewrite* **in** ‹(¤, -, -)› *snat-const-fold*[**where** *'a=64*])
apply (*rewrite* **in** ‹(-, ¤, -)› *unat-const-fold*[**where** *'a=32*])
apply (*annot-unat-const ‹TYPE(64)›*)
by *sepref*

lemma *extract-atms-clss-imp-empty-assn*[*sepref-fr-rules*]:
‹(*uncurry0 extract-atms-clss-imp-empty-assn*, *uncurry0* (*RETURN op-extract-list-empty*))
∈ *unit-assn*^{k} →$_a$ *nat-lit-list-hm-assn*›
using *extract-atms-clss-imp-empty-assn.refine*[*unfolded uncurry0-def*, *FCOMP extract-atms-clss-imp-empty-rel*
]
unfolding *uncurry0-def*
by *simp*

lemma *extract-atms-clss-imp-empty-rel-alt-def*:
‹*extract-atms-clss-imp-empty-rel* = (*RETURN* (*op-larray-custom-replicate 1024 0*, *0*, []))›
by (*auto simp*: *extract-atms-clss-imp-empty-rel-def*)

Full Initialisation

lemma *rewatch-heur-st-init-alt-def*:
‹*rewatch-heur-st-init* = (λ*S*$_0$. *do* {
let (*vdom*, *S*) = *extract-vdom-wl-heur-init S*$_0$;
ASSERT (*vdom* = *get-vdom-heur-init S*$_0$);
let (*arena*, *S*) = *extract-arena-wl-heur-init S*;
ASSERT (*arena* = *get-clauses-wl-heur-init S*$_0$);
let (*W*, *S*) = *extract-watchlist-wl-heur-init S*;
ASSERT(*length* (*vdom*) ≤ *length arena*);
W ← *rewatch-heur vdom arena W*;
RETURN (*IsaSAT-Init.update-e W* (*IsaSAT-Init.update-b arena* (*IsaSAT-Init.update-k vdom S*)))
})›
by (*auto simp*: *rewatch-heur-st-init-def isasat-init-getters-and-setters-def split*: *tuple15.splits*
intro!: *ext*)
sepref-def *rewatch-heur-st-fast-code*
is ‹(*rewatch-heur-st-init*)›
:: ‹[*rewatch-heur-st-fast-pre*]$_a$
isasat-init-assn^d → *isasat-init-assn*›
supply [[*goals-limit=1*]]
unfolding *rewatch-heur-st-init-alt-def PR-CONST-def rewatch-heur-st-fast-pre-def*
rewatch-heur-st-fast-def
by *sepref*

declare
rewatch-heur-st-fast-code.refine[*sepref-fr-rules*]

sepref-register *rewatch-heur-st init-dt-step-wl-heur*

sepref-def *init-dt-wl-heur-code-b*
is ‹*uncurry* (*init-dt-wl-heur-b*)›
:: ‹(*clauses-ll-assn*)^k *$_a$ *isasat-init-assn*^{d} →$_a$
isasat-init-assn›

supply [[*goals-limit=1*]]
unfolding *init-dt-wl-heur-def PR-CONST-def init-dt-step-wl-heur-b-def*[*symmetric*] *if-True init-dt-wl-heur-b-def*
apply (*subst nfoldli-by-idx*[*abs-def*])
unfolding *nfoldli-upt-by-while op-list-list-len-def*[*symmetric*] *Let-def init-dt-step-wl-heur-b'-def*[*symmetric*]
apply (*rewrite at* ‹(-, -, ¤)› *al-fold-custom-empty*[**where** *'l=64*])
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

definition *extract-lits-sorted'* **where**
‹*extract-lits-sorted' xs n vars* = *extract-lits-sorted* (*xs*, *n*, *vars*)›

lemma *extract-lits-sorted-extract-lits-sorted'*:
‹*extract-lits-sorted* = (λ(*xs*, *n*, *vars*). *do* {*res* ← *extract-lits-sorted' xs n vars*; *mop-free xs*; *RETURN res*})›
by (*auto simp*: *extract-lits-sorted'-def mop-free-def intro*!: *ext*)

sepref-def (**in** −) *extract-lits-sorted'-impl*
is ‹*uncurry2 extract-lits-sorted'›
:: ‹[λ((*xs*, *n*), *vars*). (∀ *x*∈#*mset vars*. *x* < *length xs*)]$_a$
(*larray64-assn uint64-nat-assn*)^k *$*_a$ *uint32-nat-assn*^{k} *$*_a$
(*arl64-assn atom-assn*)^d → *arl64-assn atom-assn* ×$_a$ *uint32-nat-assn*›
unfolding *extract-lits-sorted'-def extract-lits-sorted-def nres-monad1 prod.case*
by *sepref*

lemmas [*sepref-fr-rules*] = *extract-lits-sorted'-impl.refine*

sepref-def (**in** −) *extract-lits-sorted-code*
is ‹*extract-lits-sorted*›
:: ‹[λ(*xs*, *n*, *vars*). (∀ *x*∈#*mset vars*. *x* < *length xs*)]$_a$
isasat-atms-ext-rel-assn^d → *arl64-assn atom-assn* ×$_a$ *uint32-nat-assn*›
apply (*subst extract-lits-sorted-extract-lits-sorted'*)
unfolding *extract-lits-sorted'-def extract-lits-sorted-def nres-monad1 prod.case*
supply [[*goals-limit* = *1*]]
supply *mset-eq-setD*[*dest*] *mset-eq-length*[*dest*]
by *sepref*

declare *extract-lits-sorted-code.refine*[*sepref-fr-rules*]

abbreviation *lits-with-max-assn* **where**
‹*lits-with-max-assn* ≡ *hr-comp* (*arl64-assn atom-assn* ×$_a$ *uint32-nat-assn*) *lits-with-max-rel*›

lemma *extract-lits-sorted-hnr*[*sepref-fr-rules*]:
‹(*extract-lits-sorted-code*, *RETURN* ∘ *mset-set*) ∈ *nat-lit-list-hm-assn*^{d} →$_a$ *lits-with-max-assn*›
(**is** ‹*?c* ∈ [*?pre*]$_a$ *?im* → *?f*›)
proof −
have *H*: ‹*hrr-comp isasat-atms-ext-rel*
(λ-. *al-assn atom-assn* ×$_a$ *unat-assn*) (λ-. *lits-with-max-rel*) =

```
      (λ-. lits-with-max-assn)›
    by (auto simp: hrr-comp-def intro!: ext)

  have H: ‹?c
    ∈ [comp-PRE isasat-atms-ext-rel (λ-. True)
        (λ- (xs, n, vars). ∀x∈#mset vars. x < length xs) (λ-. True)]_a
      hrp-comp (isasat-atms-ext-rel-assn^d) isasat-atms-ext-rel → lits-with-max-assn›
    (is ‹- ∈ [?pre']_a ?im' → ?f'›)
    using hfref-compI-PRE-aux[OF extract-lits-sorted-code.refine
      extract-lits-sorted-mset-set]
      unfolding H
    by auto

  have pre: ‹?pre' x› if ‹?pre x› for x
    using that by (auto simp: comp-PRE-def isasat-atms-ext-rel-def init-valid-rep-def)
  have im: ‹?im' = ?im›
    unfolding prod-hrp-comp hrp-comp-dest hrp-comp-keep by simp
  show ?thesis
    apply (rule hfref-weaken-pre[OF ])
     defer
    using H unfolding im PR-CONST-def apply assumption
    using pre ..
qed


definition INITIAL-OUTL-SIZE :: ‹nat› where
[simp]: ‹INITIAL-OUTL-SIZE = 160›

sepref-def INITIAL-OUTL-SIZE-impl
  is ‹uncurry0 (RETURN INITIAL-OUTL-SIZE)›
  :: ‹unit-assn^k →_a sint64-nat-assn›
  unfolding INITIAL-OUTL-SIZE-def
  apply (annot-snat-const ‹TYPE(64)›)
  by sepref

definition atom-of-value :: ‹nat ⇒ nat› where [simp]: ‹atom-of-value x = x›

lemma atom-of-value-simp-hnr:
  ‹(∃x. (↑(x = unat xi ∧ P x) ∧* ↑(x = unat xi)) s) =
    (∃x. (↑(x = unat xi ∧ P x)) s)›
  ‹(∃x. (↑(x = unat xi ∧ P x)) s) = (↑(P (unat xi))) s›
  unfolding import-param-3[symmetric]
  by (auto simp: pred-lift-extract-simps)


lemma atom-of-value-hnr[sepref-fr-rules]:
  ‹(Mreturn o (λx. x), RETURN o atom-of-value) ∈ [λn. n < 2 ^31]_a (uint32-nat-assn)^d → atom-assn›
  apply sepref-to-hoare
  apply vcg'
  apply (auto simp: unat-rel-def atom-rel-def unat.rel-def br-def ENTAILS-def
    atom-of-value-simp-hnr pure-true-conv Defer-Slot.remove-slot)
  apply (rule Defer-Slot.remove-slot)
  done

sepref-register atom-of-value
```

lemma [*sepref-gen-algo-rules*]: ‹*GEN-ALGO* (*Pos 0*) (*is-init unat-lit-assn*)›
  by (*auto simp*: *unat-lit-rel-def is-init-def unat-rel-def unat.rel-def*
    br-def nat-lit-rel-def GEN-ALGO-def)

schematic-goal *mk-free-lbd-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE marked-struct-assn ?fr*›
  unfolding *marked-struct-assn-def*
  by *synthesize-free*

sepref-def *reduce-interval-init-impl*
  is ‹*uncurry0* (*RETURN reduce-interval-init*)›
  :: ‹*unit-assn*^{k} →$_a$ *word64-assn*›
  unfolding *reduce-interval-init-def*
  by *sepref*

sepref-def *inprocessing-interval-init-impl*
  is ‹*uncurry0* (*RETURN inprocessing-interval-init*)›
  :: ‹*unit-assn*^{k} →$_a$ *word-assn*›
  unfolding *inprocessing-interval-init-def*
  by *sepref*

sepref-def *rephasing-end-of-initial-phase-impl*
  is ‹*uncurry0* (*RETURN rephasing-end-of-initial-phase*)›
  :: ‹*unit-assn*^{k} →$_a$ *word-assn*›
  unfolding *rephasing-end-of-initial-phase-def*
  by *sepref*

sepref-def *rephasing-initial-phase-impl*
  is ‹*uncurry0* (*RETURN rephasing-initial-phase*)›
  :: ‹*unit-assn*^{k} →$_a$ *word-assn*›
  unfolding *rephasing-initial-phase-def*
  by *sepref*

sepref-def *subsuming-length-initial-phase-impl*
  is ‹*uncurry0* (*RETURN subsuming-length-initial-phase*)›
  :: ‹*unit-assn*^{k} →$_a$ *word-assn*›
  unfolding *subsuming-length-initial-phase-def*
  by *sepref*

definition *empty-heuristics-stats* :: ‹- ⇒ - ⇒ *restart-heuristics*› **where**
  ‹*empty-heuristics-stats opts φ* = (
  let fema = *ema-init* (*opts-fema opts*) *in*
  let sema = *ema-init* (*opts-sema opts*) *in let ccount* = *restart-info-init in*
  let n = (*length φ*) *in*
    (*fema*, *sema*, *ccount*, *0*, (*φ*, *0*, *replicate n False*, *0*, *replicate n False*, *rephasing-end-of-initial-phase*,
0, *rephasing-initial-phase*),
   reluctant-init, *False*, *replicate n False*, (*inprocessing-interval-init*, *reduce-interval-init*, *subsuming-length-initial-phase*),
fema, *sema*))›

sepref-def *empty-heuristics-stats-impl*
  is ‹*uncurry* (*RETURN oo empty-heuristics-stats*)›
  :: ‹*opts-assn*^{k} *$_a$ *phase-saver-assn*^{d} →$_a$ *heuristic-int-assn*›
  supply [[*goals-limit=1*]]
  unfolding *heuristic-int-assn-def empty-heuristics-stats-def phase-heur-assn-def*
  apply (*rewrite at* ‹*replicate - False*› *annotate-assn*[**where** *A=phase-saver′-assn*])
  apply (*rewrite* **in** ‹*replicate - False*› *array-fold-custom-replicate*)
  apply (*rewrite at* ‹*replicate - False*› *annotate-assn*[**where** *A=phase-saver′-assn*])

apply (*rewrite* **in** ‹*replicate - False*› *array-fold-custom-replicate*)
apply (*rewrite at* ‹*replicate - False*› *annotate-assn*[**where** *A=phase-saver-assn*])
apply (*rewrite* **in** ‹*replicate - False*› *larray-fold-custom-replicate*)
by *sepref*

lemma *finalise-init-code-alt-def*:
‹*finalise-init-code opts =*
(λ*S*. *case S of Tuple15 M' N' D' Q' W' vm φ clvls cach*
lbd vdom ivdom failed lcount mark ⇒ *do* {
let init-stats = empty-stats-clss (*of-nat*(*length ivdom*));
let heur = empty-heuristics-stats opts φ;
mop-free mark; *mop-free failed*;
let occs = replicate (*length W'*) [];
vm ← *finalize-bump-init vm*;
RETURN (*IsaSAT M' N' D' Q' W' vm*
clvls cach lbd (*take 1*(*replicate 160* (*Pos 0*))) *init-stats*
(*Restart-Heuristics heur*) (*AIvdom-init vdom* [] *ivdom*) *lcount opts* [] *occs*)
})›
unfolding *finalise-init-code-def mop-free-def empty-heuristics-stats-def*
by (*auto simp*: *Let-def split*: *prod.splits tuple15.splits intro*!: *ext*)

sepref-def *finalize-vmtf-init-code*
is ‹*finalize-vmtf-init*›
:: ‹*vmtf-init-assn*^{d} →$_a$ *vmtf-assn*›
unfolding *finalize-vmtf-init-def vmtf-init-assn-def vmtf-assn-def atom.fold-option*
by *sepref*

sepref-def *finalize-bump-init-code*
is ‹*finalize-bump-init*›
:: ‹*heuristic-bump-init-assn*^{d} →$_a$ *heuristic-bump-assn*›
unfolding *finalize-bump-init-def*
by *sepref*

sepref-def *finalise-init-code'*
is ‹*uncurry finalise-init-code*›
:: ‹[λ(-, *S*). *length* (*get-clauses-wl-heur-init S*) ≤ *snat64-max*]$_a$
opts-assn^d **$_a$ isasat-init-assn*^{d} → *isasat-bounded-assn*›
supply [[*goals-limit=1*]] *of-nat-snat*[*sepref-import-param*]
unfolding *finalise-init-code-alt-def isasat-init-assn-def*
INITIAL-OUTL-SIZE-def[*symmetric*]
phase-heur-assn-def op-list-list-len-def[*symmetric*]
apply (*rewrite at* ‹*Pos* ⌑› *unat-const-fold*[**where** *'a=32*])
apply (*rewrite at* ‹*Pos* ⌑› *atom-of-value-def*[*symmetric*])
apply (*rewrite at* ‹*take* ⌑› *snat-const-fold*[**where** *'a=64*])
apply (*rewrite at* ‹*AIvdom-init - ⌑ -*› *al-fold-custom-empty*[**where** *'l=64*])
apply (*rewrite at* ‹*IsaSAT - - - - - - - - - - - - - - - ⌑ -*› *al-fold-custom-empty*[**where** *'l=64*])
apply (*rewrite* **in** ‹*take - ⌑*› *al-fold-custom-replicate*)
apply (*rewrite* **in** ‹*replicate - []*› *aal-fold-custom-empty*(*1*)[**where** *'l=64* **and** *'ll=64*])
by *sepref*

sepref-register *initialise-VMTF*

sepref-def *init-trail-D-fast-code*
is ‹*uncurry2 init-trail-D-fast*›
:: ‹(*arl64-assn atom-assn*)^k **$_a$ sint64-nat-assn*^{k} **$_a$ sint64-nat-assn*^{k} →$_a$ *trail-pol-fast-assn*›
unfolding *init-trail-D-def PR-CONST-def init-trail-D-fast-def trail-pol-fast-assn-def*

apply (*rewrite* **in** ‹*let - = ⌑ in ->* *annotate-assn*[**where** *A=‹arl64-assn unat-lit-assn›*])
apply (*rewrite* **in** ‹*let - = ⌑ in ->* *al-fold-custom-empty*[**where** *'l=64*])
apply (*rewrite* **in** ‹*let - = -; - = ⌑ in ->* *al-fold-custom-empty*[**where** *'l=64*])
apply (*rewrite* **in** ‹*let - = -; - = ⌑ in ->* *annotate-assn*[**where** *A=‹arl64-assn uint32-nat-assn›*])

apply (*rewrite* **in** ‹*let - = -;- = ⌑ in ->* *annotate-assn*[**where** *A=‹larray64-assn (tri-bool-assn)›*])
apply (*rewrite* **in** ‹*let - = -;- = -;- = ⌑ in ->* *annotate-assn*[**where** *A=‹larray64-assn uint32-nat-assn›*])
apply (*rewrite* **in** ‹*let - = - in ->* *larray-fold-custom-replicate*)
apply (*rewrite* **in** ‹*let - = - in ->* *larray-fold-custom-replicate*)
apply (*rewrite* **in** ‹*let - = - in ->* *larray-fold-custom-replicate*)
apply (*rewrite at* ‹(-, ⌑, -)› *unat-const-fold*[**where** *'a=32*])
apply (*rewrite at* ‹(*op-larray-custom-replicate* - ⌑)› *unat-const-fold*[**where** *'a=32*])
apply (*rewrite at* ‹(-, -, -, -, ⌑)› *snat-const-fold*[**where** *'a=64*])
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
supply [[*goals-limit = 1*]]
by *sepref*

declare *init-trail-D-fast-code.refine*[*sepref-fr-rules*]
definition *empty-mark-struct* :: ‹*nat* ⇒ *nat* × *bool option list*› **where**
‹*empty-mark-struct* (*n*::*nat*) = (*0*::*nat*, *replicate n NoMark*)›

sepref-def *empty-mark-struct-impl*
is ‹*RETURN o empty-mark-struct*›
:: ‹*sint64-nat-assn*^{k} →$_a$ *marked-struct-assn*›
unfolding *empty-mark-struct-def marked-struct-assn-def*
apply (*rewrite at* ‹(⌑, *replicate - NoMark*)› *unat-const-fold*[**where** *'a=32*])
unfolding *array-fold-custom-replicate*
by *sepref*

definition *combine-conflict* **where**
‹*combine-conflict x = x*›
sepref-def *combine-conflict-impl*
is ‹*RETURN o combine-conflict*›
:: ‹(*bool1-assn* ×$_a$ *unat32-assn* ×$_a$ *IICF-Array.array-assn option-bool-impl-assn*)^d →$_a$ *conflict-option-rel-assn*›
unfolding *combine-conflict-def conflict-option-rel-assn-def lookup-clause-rel-assn-def*
by *sepref*

definition *combine-ccach* **where**
‹*combine-ccach x = x*›
sepref-def *combine-ccach-impl*
is ‹*RETURN o combine-ccach*›
:: ‹(*array-assn minimize-status-assn* ×$_a$ *arl64-assn atom-assn*)^d →$_a$ *cach-refinement-l-assn*›
unfolding *combine-ccach-def cach-refinement-l-assn-def*
by *sepref*

definition *combine-lcount* **where**
‹*combine-lcount x = x*›
sepref-def *combine-lcount-impl*
is ‹*RETURN o combine-lcount*›
:: ‹(*uint64-nat-assn* ×$_a$ *uint64-nat-assn* ×$_a$ *uint64-nat-assn* ×$_a$ *uint64-nat-assn* ×$_a$ *uint64-nat-assn*)^k
→$_a$ *lcount-assn*›
unfolding *combine-lcount-def lcount-assn-def*
by *sepref*

sepref-register *empty-mark-struct combine-conflict combine-ccach combine-lcount*

lemma *init-state-wl-D'-alt-def*:
  ‹*init-state-wl-D'* = (λ(\mathcal{A}_{in}, n). *do* {
    ASSERT(*Suc* (*2* ∗ (*n*)) ≤ *unat32-max*);
    let n = *Suc* (*n*);
    let m = *2* ∗ *n*;
    M ← *init-trail-D* \mathcal{A}_{in} *n m*;
    let N = [];
    let D = *combine-conflict* (*True*, *0*, *replicate n NOTIN*);
    let mark = (*0*, *replicate n None*);
    let WS = *replicate m* [];
    vm ← *initialize-Bump-Init* \mathcal{A}_{in} *n*;
    let φ = *replicate n False*;
    let cach = *combine-ccach* (*replicate n SEEN-UNKNOWN*, []);
    let lbd = *empty-lbd*;
    let vdom = [];
    let ivdom = [];
    let lcount = *combine-lcount* (*0*, *0*, *0*, *0*, *0*);
    RETURN (*Tuple15 M N D 0 WS vm φ 0 cach lbd vdom ivdom False lcount mark*)
    })›
  unfolding *combine-conflict-def combine-ccach-def init-state-wl-D'-def combine-lcount-def* **by** *auto*

sepref-definition *init-state-wl-D'-code*
  is ‹*init-state-wl-D'*›
  :: ‹(*arl64-assn atom-assn* ×_a *uint32-nat-assn*)^k →_a *isasat-init-assn*›
  supply[[*goals-limit=1*]]
  unfolding *init-state-wl-D'-alt-def PR-CONST-def init-trail-D-fast-def*[*symmetric*] *Suc-eq-plus1-left*
    NoMark-def[*symmetric*] *empty-mark-struct-def*[*symmetric*] *of-nat-snat*[*sepref-import-param*]
  apply (*rewrite at* ‹*let* - = *1* + ¤ *in* -› *annot-unat-snat-upcast*[**where** *'l=64*])
  apply (*rewrite at* ‹*let* - = *combine-ccach* (-, ¤) *in* -› *al-fold-custom-empty*[**where** *'l=64*])
  apply (*rewrite at* ‹*let* - = *combine-ccach* (¤,-) *in* -› *annotate-assn*[**where** *A*= ‹*array-assn mini-mize-status-assn*›])
  apply (*rewrite at* ‹*let* - = *combine-ccach* (-, ¤) *in* -› *annotate-assn*[**where** *A*= ‹*arl64-assn atom-assn*›])
  apply (*rewrite* **in** ‹*replicate* - []› *aal-fold-custom-empty*(*1*)[**where** *'l=64* **and** *'ll=64*])
  apply (*rewrite at* ‹*let* -= -; -= ¤ *in* -› *annotate-assn*[**where** *A*=‹*watchlist-fast-assn*›])
  apply (*rewrite at* ‹*let* -= ¤; -=-;-=-;- = -;-=-; - = - *in RETURN* -› *annotate-assn*[**where** *A*=‹*phase-saver-assn*›])
  apply (*rewrite* **in** ‹*let* -= ¤; -=-;-=-;- = -; -= -; -=- *in RETURN* -› *larray-fold-custom-replicate*)
  apply (*rewrite* **in** ‹*let* -= *combine-conflict*(*True*, -, ¤) *in* -› *array-fold-custom-replicate*)
  unfolding *array-fold-custom-replicate*
  apply (*rewrite at* ‹*let* - = ¤ *in let* - = *combine-conflict* (*True*, -, -) *in* -› *al-fold-custom-empty*[**where** *'l=64*])
  apply (*rewrite* **in** ‹*let* -= *combine-conflict* (*True*, ¤, -) *in* -› *unat-const-fold*[**where** *'a=32*])
  apply (*rewrite at* ‹*let* - = ¤ *in* -› *annotate-assn*[**where** *A*=‹*arena-fast-assn*›])
  apply (*rewrite at* ‹*let* -= ¤; - = - ; - =- *in RETURN* -› *annotate-assn*[**where** *A* = ‹*vdom-fast-assn*›])
  apply (*rewrite at* ‹*let* -= ¤; - = - *in RETURN* -› *annotate-assn*[**where** *A* = ‹*vdom-fast-assn*›])
  apply (*rewrite* **in** ‹*let* -= ¤; -=-; - = - *in RETURN* -› *al-fold-custom-empty*[**where** *'l=64*])
  apply (*rewrite* **in** ‹*let* -= ¤; - = - *in RETURN* -› *al-fold-custom-empty*[**where** *'l=64*])
  apply (*rewrite at* ‹*Tuple15* - - - - - - - ¤› *unat-const-fold*[**where** *'a=32*])
  apply (*rewrite at* ‹*let* - = *combine-lcount* (¤, -, -) *in RETURN* -› *unat-const-fold*[**where** *'a=64*])
  apply (*rewrite at* ‹*let* - = *combine-lcount* (-, ¤, -) *in RETURN* -› *unat-const-fold*[**where** *'a=64*])
  apply (*rewrite at* ‹*let* - = *combine-lcount* (-, -, ¤, -) *in RETURN* -› *unat-const-fold*[**where** *'a=64*])
  apply (*rewrite at* ‹*let* - = *combine-lcount* (-, -, -, ¤, -) *in RETURN* -› *unat-const-fold*[**where** *'a=64*])
  apply (*rewrite at* ‹*let* - = *combine-lcount* (-, -, -, -, ¤) *in RETURN* -› *unat-const-fold*[**where** *'a=64*])
  apply (*annot-snat-const* ‹*TYPE*(*64*)›)
  apply (*rewrite at* ‹*RETURN* ¤› *annotate-assn*[**where** *A*=‹*isasat-init-assn*›])
  by *sepref*

declare *init-state-wl-D′-code.refine*[*sepref-fr-rules*]

lemma *to-init-state-code-hnr*:
‹(*Mreturn o to-init-state-code*, *RETURN o id*) ∈ *isasat-init-assn*^{d} →$_a$ *isasat-init-assn*›
unfolding *to-init-state-code-def*
by *sepref-to-hoare vcg′*

abbreviation (**in** −)*lits-with-max-assn-clss* **where**
‹*lits-with-max-assn-clss* ≡ *hr-comp lits-with-max-assn* (⟨*nat-rel*⟩*mset-rel*)›

lemmas [*unfolded Tuple15-LLVM.inline-direct-return-node-case*, *llvm-code*] =
get-conflict-wl-is-None-init-code-def[*unfolded IsaSAT-Init.read-all-st-code-def*]

lemmas [*llvm-code*] =
init-state-wl-D′-code-def[*unfolded comp-def*]

lemmas [*unfolded Tuple15-LLVM.inline-direct-return-node-case*, *llvm-code*] =
get-conflict-wl-is-None-init-code-def[*unfolded IsaSAT-Init.read-all-st-code-def*]

schematic-goal *mk-free-isasat-init-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE isasat-init-assn ?fr*›
unfolding *isasat-init-assn-def*
by *synthesize-free+*

experiment
begin
export-llvm *init-state-wl-D′-code*
rewatch-heur-st-fast-code
init-dt-wl-heur-code-b
init-state-wl-D′-code
end

end
theory *IsaSAT-Conflict-Analysis-Defs*
imports *IsaSAT-Setup*
IsaSAT-Bump-Heuristics
IsaSAT-VMTF IsaSAT-LBD
begin

definition *lit-and-ann-of-propagated-st* :: ‹*nat twl-st-wl* ⇒ *nat literal* × *nat*› **where**
‹*lit-and-ann-of-propagated-st S* = *lit-and-ann-of-propagated* (*hd* (*get-trail-wl S*))›

definition *lit-and-ann-of-propagated-st-heur*
:: ‹*isasat* ⇒ (*nat literal* × *nat*) *nres*›
where
‹*lit-and-ann-of-propagated-st-heur* = (λ*S*. *do* {
let (*M*, -, -, *reasons*, -, -) = *get-trail-wl-heur S*;
ASSERT(*M* ≠ [] ∧ *atm-of* (*last M*) < *length reasons*);
RETURN (*last M*, *reasons* ! (*atm-of* (*last M*)))})›

definition *tl-state-wl-heur-pre* :: ‹*isasat* ⇒ *bool*› **where**
‹*tl-state-wl-heur-pre* =
(λ*S*.
let M = *get-trail-wl-heur S in let x* = *get-vmtf-heur S in fst M* ≠ [] ∧

tl-trailt-tr-pre M)›

definition *tl-state-wl-heur* :: ‹*isasat* ⇒ (*bool* × *isasat*) *nres*› **where**
‹*tl-state-wl-heur* = (λ*S*. *do* {
ASSERT(*tl-state-wl-heur-pre S*);
let M = *get-trail-wl-heur S*; *let vm* = *get-vmtf-heur S*;
let S = *set-trail-wl-heur* (*tl-trailt-tr M*) *S*;
ASSERT (*isa-bump-unset-pre* (*atm-of* (*lit-of-last-trail-pol M*)) *vm*);
vm ← *isa-bump-unset* (*atm-of* (*lit-of-last-trail-pol M*)) *vm*;
let S = *set-vmtf-wl-heur vm S*;
RETURN (*False*, *S*)
})›

definition *update-confl-tl-wl-heur*
:: ‹*nat literal* ⇒ *nat* ⇒ *isasat* ⇒ (*bool* × *isasat*) *nres*›
where
‹*update-confl-tl-wl-heur* = (λ*L C S*. *do* {
let M = *get-trail-wl-heur S*;
let N = *get-clauses-wl-heur S*;
let lbd = *get-lbd S*;
let outl = *get-outlearned-heur S*;
let clvls = *get-count-max-lvls-heur S*;
let vm = *get-vmtf-heur S*;
let (*b*, (*n*, *xs*)) = *get-conflict-wl-heur S*;
(*N*, *lbd*) ← *calculate-LBD-heur-st M N lbd C*;
ASSERT (*clvls* ≥ *1*);
let L' = *atm-of L*;
ASSERT(*arena-is-valid-clause-idx N C*);
((*b*, (*n*, *xs*)), *clvls*, *outl*) ←
if arena-length N C = *2 then isasat-lookup-merge-eq2 L M N C* (*b*, (*n*, *xs*)) *clvls outl*
else isa-resolve-merge-conflict-gt2 M N C (*b*, (*n*, *xs*)) *clvls outl*;
ASSERT(*curry lookup-conflict-remove1-pre L* (*n*, *xs*) ∧ *clvls* ≥ *1*);
let (*n*, *xs*) = *lookup-conflict-remove1 L* (*n*, *xs*);
ASSERT(*arena-act-pre N C*);
vm ← *isa-vmtf-bump-to-rescore-also-reasons-cl M N C* (−*L*) *vm*;
ASSERT(*isa-bump-unset-pre L' vm*);
ASSERT(*tl-trailt-tr-pre M*);
vm ← *isa-bump-unset L' vm*;
let S = *set-trail-wl-heur* (*tl-trailt-tr M*) *S*;
let S = *set-conflict-wl-heur* (*b*, (*n*, *xs*)) *S*;
let S = *set-vmtf-wl-heur vm S*;
let S = *set-count-max-wl-heur* (*clvls* − *1*) *S*;
let S = *set-outl-wl-heur outl S*;
let S = *set-clauses-wl-heur N S*;
let S = *set-lbd-wl-heur lbd S*;
RETURN (*False*, *S*)
})›

definition *is-decided-hd-trail-wl-heur* :: ‹*isasat* ⇒ *bool*› **where**
‹*is-decided-hd-trail-wl-heur* = (λ*S*. *is-None* (*snd* (*last-trail-pol* (*get-trail-wl-heur S*))))›

definition *skip-and-resolve-loop-wl-D-heur-inv* **where**
‹*skip-and-resolve-loop-wl-D-heur-inv S*$_0$' =
(λ(*brk*, *S'*). ∃ *S S*$_0$. (*S'*, *S*) ∈ *twl-st-heur-conflict-ana* ∧ (*S*$_0$', *S*$_0$) ∈ *twl-st-heur-conflict-ana* ∧
skip-and-resolve-loop-wl-inv S$_0$ *brk S* ∧
length (*get-clauses-wl-heur S'*) = *length* (*get-clauses-wl-heur S*$_0$') ∧

get-learned-count S' = *get-learned-count* S_0')›

definition *update-confl-tl-wl-heur-pre*
:: ‹(*nat* × *nat literal*) × *isasat* ⇒ *bool*›
where
‹*update-confl-tl-wl-heur-pre* =
(λ((*i*, *L*), *S*).
let M = *get-trail-wl-heur S*; *x* = *get-vmtf-heur S in*
i > 0 ∧
(*fst M*) ≠ [] ∧
isa-bump-unset-pre (*atm-of L*) *x* ∧
L = (*last* (*fst M*))
)›

definition *lit-and-ann-of-propagated-st-heur-pre* **where**
‹*lit-and-ann-of-propagated-st-heur-pre* = (λ((*M*, -, -, *reasons*, -), -). *atm-of* (*last M*) < *length reasons* ∧ *M* ≠ [])›

definition *atm-is-in-conflict-st-heur-pre*
:: ‹*nat literal* × *isasat* ⇒ *bool*›
where
‹*atm-is-in-conflict-st-heur-pre* = (λ(*L*, *S*). *atm-of L* < *length* (*snd* (*snd* (*get-conflict-wl-heur S*))))›

definition *maximum-level-removed-eq-count-dec-heur* **where**
‹*maximum-level-removed-eq-count-dec-heur L S* =
RETURN (*get-count-max-lvls-heur S* > 1)›

definition *is-decided-hd-trail-wl-heur-pre* **where**
‹*is-decided-hd-trail-wl-heur-pre* =
(λ*S*. *fst* (*get-trail-wl-heur S*) ≠ [] ∧ *last-trail-pol-pre* (*get-trail-wl-heur S*))›

definition *skip-and-resolve-loop-wl-D-heur*
:: ‹*isasat* ⇒ *isasat nres*›
where
‹*skip-and-resolve-loop-wl-D-heur* S_0 =
do {
- ← *RETURN* (*IsaSAT-Profile.start-analyze*);
(-, *S*) ←
$WHILE_T^{skip\text{-}and\text{-}resolve\text{-}loop\text{-}wl\text{-}D\text{-}heur\text{-}inv\ S_0}$
(λ(*brk*, *S*). ¬*brk* ∧ ¬*is-decided-hd-trail-wl-heur S*)
(λ(*brk*, *S*).
do {
ASSERT(¬*brk* ∧ ¬*is-decided-hd-trail-wl-heur S*);
(*L*, *C*) ← *lit-and-ann-of-propagated-st-heur S*;
b ← *atm-is-in-conflict-st-heur* (−*L*) *S*;
if b then
tl-state-wl-heur S
else do {
b ← *maximum-level-removed-eq-count-dec-heur L S*;
if b
then do {
update-confl-tl-wl-heur L C S
}
else
RETURN (*True*, *S*)

```
          }
        }
      )
      (False, S_0);
    - ← RETURN (IsaSAT-Profile.stop-analyze);
    RETURN S
  }
 ›
lemma twl-st-heur-conflict-ana-trail-empty: ‹(T, x) ∈ twl-st-heur-conflict-ana ⟹
 fst (get-trail-wl-heur T) = [] ⟷ get-trail-wl x = []›
 by
  (clarsimp simp: twl-st-heur-def state-wl-l-def twl-st-l-def twl-st-heur-conflict-ana-def
   trail-pol-alt-def last-trail-pol-pre-def last-rev hd-map literals-are-in-\mathcal{L}_in-trail-def simp flip: rev-map
   dest: multi-member-split)
lemma twl-st-heur-conflict-ana-last-trail-pol-pre:
 ‹(T, x) ∈ twl-st-heur-conflict-ana ⟹ fst (get-trail-wl-heur T) ≠ [] ⟹ last-trail-pol-pre (get-trail-wl-heur
T)›
 apply (cases ‹get-trail-wl x›)
 by (clarsimp-all simp: twl-st-heur-def state-wl-l-def twl-st-l-def twl-st-heur-conflict-ana-def
   trail-pol-alt-def last-trail-pol-pre-def last-rev hd-map literals-are-in-\mathcal{L}_in-trail-def simp flip: rev-map
   dest: multi-member-split)
   (clarsimp-all dest!: multi-member-split simp: ann-lits-split-reasons-def)


lemma skip-and-resolve-loop-wl-DI:
 assumes
   ‹skip-and-resolve-loop-wl-D-heur-inv S (b, T)›
 shows ‹is-decided-hd-trail-wl-heur-pre T›
 using assms apply −
 unfolding skip-and-resolve-loop-wl-inv-def skip-and-resolve-loop-inv-l-def skip-and-resolve-loop-inv-def
    skip-and-resolve-loop-wl-D-heur-inv-def is-decided-hd-trail-wl-heur-pre-def
 apply (subst (asm) case-prod-beta)+
 unfolding prod.case
 apply normalize-goal+
 by (simp add: twl-st-heur-conflict-ana-trail-empty twl-st-heur-conflict-ana-last-trail-pol-pre)

lemma isasat-fast-after-skip-and-resolve-loop-wl-D-heur-inv:
 ‹isasat-fast x ⟹ skip-and-resolve-loop-wl-D-heur-inv x (False, a2') ⟹ isasat-fast a2'›
 unfolding skip-and-resolve-loop-wl-D-heur-inv-def isasat-fast-def
 by (auto dest: get-learned-count-learned-clss-countD2)

end
theory IsaSAT-Conflict-Analysis
 imports IsaSAT-Conflict-Analysis-Defs
begin


lemma literals-are-in-\mathcal{L}_in-mm-all-atms-self[simp]:
 ‹literals-are-in-\mathcal{L}_in-mm (all-atms ca NUE) {#mset (fst x). x ∈# ran-m ca#}›
 by (auto simp: literals-are-in-\mathcal{L}_in-mm-def in-\mathcal{L}_all-atm-of-\mathcal{A}_in
   all-atms-def all-lits-def in-all-lits-of-mm-ain-atms-of-iff)

lemma literals-are-in-\mathcal{L}_in-mm-ran-m[simp]:
 ‹literals-are-in-\mathcal{L}_in-mm (all-atms-st (x1a, x1b, y))
     {#mset (fst x). x ∈# ran-m x1b#}›
 by (cases y; auto simp: literals-are-in-\mathcal{L}_in-mm-def all-lits-st-def all-lits-def all-lits-of-mm-union
```

simp flip: *all-lits-st-alt-def*)

Skip and resolve **definition** *maximum-level-removed-eq-count-dec* **where**
‹*maximum-level-removed-eq-count-dec L S* ⟷
get-maximum-level-remove (*get-trail-wl S*) (*the* (*get-conflict-wl S*)) *L* =
count-decided (*get-trail-wl S*)›

definition *maximum-level-removed-eq-count-dec-pre* **where**
‹*maximum-level-removed-eq-count-dec-pre* =
(λ(*L*, *S*). *L* = −*lit-of* (*hd* (*get-trail-wl S*)) ∧ *L* ∈# *the* (*get-conflict-wl S*) ∧
get-conflict-wl S ≠ *None* ∧ *get-trail-wl S* ≠ [] ∧ *count-decided* (*get-trail-wl S*) ≥ *1*)›

lemma *maximum-level-removed-eq-count-dec-heur-maximum-level-removed-eq-count-dec*:
‹(*uncurry maximum-level-removed-eq-count-dec-heur*,
uncurry mop-maximum-level-removed-wl) ∈
[λ-. *True*]$_f$
Id ×$_r$ *twl-st-heur-conflict-ana* → ⟨*bool-rel*⟩*nres-rel*›
unfolding *maximum-level-removed-eq-count-dec-heur-def mop-maximum-level-removed-wl-def*
uncurry-def
apply (*intro frefI nres-relI*)
subgoal for *x y*
apply *refine-rcg*
apply (*cases x*)
apply (*auto simp*: *count-decided-st-def counts-maximum-level-def twl-st-heur-conflict-ana-def*
maximum-level-removed-eq-count-dec-heur-def maximum-level-removed-eq-count-dec-def
maximum-level-removed-eq-count-dec-pre-def mop-maximum-level-removed-wl-pre-def
mop-maximum-level-removed-l-pre-def mop-maximum-level-removed-pre-def state-wl-l-def
twl-st-l-def get-maximum-level-card-max-lvl-ge1 card-max-lvl-remove-hd-trail-iff)
done
done

lemma *lit-and-ann-of-propagated-st-heur-lit-and-ann-of-propagated-st*:
‹(*lit-and-ann-of-propagated-st-heur*, *mop-hd-trail-wl*) ∈
[λ*S*. *True*]$_f$ *twl-st-heur-conflict-ana* → ⟨*Id* ×$_f$ *Id*⟩*nres-rel*›
apply (*intro frefI nres-relI*)
unfolding *lit-and-ann-of-propagated-st-heur-def mop-hd-trail-wl-def*
apply *refine-rcg*
apply (*case-tac* ‹*get-trail-wl-heur x*›)
apply (*auto simp*: *twl-st-heur-conflict-ana-def mop-hd-trail-wl-def mop-hd-trail-wl-pre-def*
mop-hd-trail-l-pre-def twl-st-l-def state-wl-l-def mop-hd-trail-pre-def last-rev hd-map
lit-and-ann-of-propagated-st-def trail-pol-alt-def ann-lits-split-reasons-def
intro!: *ASSERT-leI ASSERT-refine-right simp flip*: *rev-map elim*: *is-propedE*)
apply (*auto elim*!: *is-propedE*)
done

definition *tl-state-wl-pre* **where**
‹*tl-state-wl-pre S* ⟷ *get-trail-wl S* ≠ [] ∧
literals-are-in-$\mathcal{L}_{in}$*-trail* (*all-atms-st S*) (*get-trail-wl S*) ∧
(*lit-of* (*hd* (*get-trail-wl S*))) ∉# *the* (*get-conflict-wl S*) ∧
−(*lit-of* (*hd* (*get-trail-wl S*))) ∉# *the* (*get-conflict-wl S*) ∧
¬*tautology* (*the* (*get-conflict-wl S*)) ∧
distinct-mset (*the* (*get-conflict-wl S*)) ∧
¬*is-decided* (*hd* (*get-trail-wl S*)) ∧
count-decided (*get-trail-wl S*) > *0*›

lemma *tl-state-out-learned*:
  ‹*lit-of* (*hd a*) ∉# *the at'* ⟹
    − *lit-of* (*hd a*) ∉# *the at'* ⟹
    ¬ *is-decided* (*hd a*) ⟹
    out-learned (*tl a*) *at' an* ⟷ *out-learned a at' an*›
  by (*cases a*) (*auto simp*: *out-learned-def get-level-cons-if atm-of-eq-atm-of*
    intro!: *filter-mset-cong*)

lemma *mop-tl-state-wl-pre-tl-state-wl-heur-pre*:
  ‹(*x*, *y*) ∈ *twl-st-heur-conflict-ana* ⟹ *mop-tl-state-wl-pre y* ⟹ *tl-state-wl-heur-pre x*›
  ‹(*x*, *y*) ∈ *twl-st-heur-conflict-ana* ⟹ *mop-tl-state-wl-pre y* ⟹
    isa-bump-unset-pre (*atm-of* (*lit-of-last-trail-pol* (*get-trail-wl-heur x*)))
  (*get-vmtf-heur x*)
›
  using *tl-trailt-tr-pre*[*of* ‹*get-trail-wl y*› ‹*get-trail-wl-heur x*› ‹*all-atms-st y*›]
  subgoal
    unfolding *mop-tl-state-wl-pre-def tl-state-wl-heur-pre-def mop-tl-state-l-pre-def*
      mop-tl-state-pre-def tl-state-wl-heur-pre-def
    by (*cases* ‹*get-trail-wl-heur x*›; *cases y*)
      (*clarsimp-all simp*: *twl-st-heur-conflict-ana-def state-wl-l-def twl-st-l-def trail-pol-alt-def*
      rev-map[*symmetric*] *last-rev hd-map simp flip*: *all-lits-st-alt-def*
    intro!: *isa-bump-unset-pre*[**where** *M* = ‹*get-trail-wl y*›])
  subgoal
    unfolding *mop-tl-state-wl-pre-def tl-state-wl-heur-pre-def mop-tl-state-l-pre-def*
      mop-tl-state-pre-def tl-state-wl-heur-pre-def
    apply *normalize-goal+*
    apply (*cases* ‹*get-trail-wl y*›)
    apply (*solves simp*)
    apply (*rule isa-bump-unset-pre*[*of* - ‹*all-atms-st y*› ‹*get-trail-wl y*›])
    apply (*simp add*: *twl-st-heur-conflict-ana-def*)
    apply (*simp add*: *twl-st-heur-conflict-ana-def lit-of-last-trail-pol-def*)
    apply (*clarsimp-all simp*: *twl-st-heur-conflict-ana-def state-wl-l-def twl-st-l-def trail-pol-alt-def*
      rev-map[*symmetric*] *last-rev hd-map lit-of-last-trail-pol-def*
      simp flip: *all-lits-st-alt-def IsaSAT-Setup.all-lits-st-alt-def*
    intro!: *isa-bump-unset-pre*[**where** *M* = ‹*get-trail-wl y*›])
    using *IsaSAT-Setup.all-lits-st-alt-def* **by** *blast*
  done

lemma *mop-tl-state-wl-pre-simps*:
  ‹*mop-tl-state-wl-pre* ([], *ax*, *ay*, *az*, *bga*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *bh*, *bi*) ⟷ *False*›
  ‹*mop-tl-state-wl-pre* (*xa*, *ax*, *ay*, *az*, *bga*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *bh*, *bi*) ⟹
    lit-of (*hd xa*) ∈# *all-lits-st* (*xa'*, *ax*, *ay''*, *az*, *bga*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *bh''*, *bi''*)›
  ‹*mop-tl-state-wl-pre* (*xa*, *ax*, *ay*, *az*, *bga*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *bh*, *bi*) ⟹ *lit-of* (*hd xa*) ∉# *the ay*›
  ‹*mop-tl-state-wl-pre* (*xa*, *ax*, *ay*, *az*, *bga*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *bh*, *bi*) ⟹ −*lit-of* (*hd xa*) ∉# *the ay*›
  ‹*mop-tl-state-wl-pre* (*xa*, *ax*, *Some ay'*, *az*, *bga*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *bh*, *bi*) ⟹ *lit-of* (*hd xa*) ∉# *ay'*›
  ‹*mop-tl-state-wl-pre* (*xa*, *ax*, *Some ay'*, *az*, *bga*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *bh*, *bi*) ⟹ −*lit-of* (*hd xa*) ∉# *ay'*›
  ‹*mop-tl-state-wl-pre* (*xa*, *ax*, *ay*, *az*, *bga*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *bh*, *bi*) ⟹ *is-proped* (*hd xa*)›
  ‹*mop-tl-state-wl-pre* (*xa*, *ax*, *ay*, *az*, *bga*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *bh*, *bi*) ⟹ *count-decided xa* > *0*›
  unfolding *mop-tl-state-wl-pre-def tl-state-wl-heur-pre-def mop-tl-state-l-pre-def*
    mop-tl-state-pre-def tl-state-wl-heur-pre-def

apply (*auto simp*: *twl-st-heur-conflict-ana-def state-wl-l-def twl-st-l-def trail-pol-alt-def*
rev-map[*symmetric*] *last-rev hd-map mset-take-mset-drop-mset'* $\mathcal{L}_{all}$*-all-atms-all-lits*
simp flip: *image-mset-union all-lits-def all-lits-st-alt-def*; *fail*)[]
apply (*normalize-goal*+; *simp add*: *all-lits-st-def*; *fail*)+
done

lemma *all-atms-st-st-simps2*[*simp*]:
‹*all-atms-st* (*tl xaa*, *bu*, *bv*, *bw*, *bx*, *by*, *NEk*, *UEk*, *bz*, *ca*, *cb*, *cc*, *cd*) =
all-atms-st (*xaa*, *bu*, *bv*, *bw*, *bx*, *by*, *NEk*, *UEk*, *bz*, *ca*, *cb*, *cc*, *cd*)›
‹*all-atms-st* (*xaa*, *bu*, *Some* (*bv'* ∪# *tt* − *uu*), *bw*, *bx*, *by*, *NEk*, *UEk*, *bz*, *ca*, *cb*, *cc*, *cd*) =
all-atms-st (*xaa*, *bu*, *Some bv'*, *bw*, *bx*, *by*, *NEk*, *UEk*, *bz*, *ca*, *cb*, *cc*, *cd*)›
by (*auto simp*: *all-atms-st-def*)

declare *isasat-input-bounded-def*[*simp del*]
lemma *tl-state-wl-heur-tl-state-wl*:
‹(*tl-state-wl-heur*, *mop-tl-state-wl*) ∈
[λ-. *True*]$_f$ *twl-st-heur-conflict-ana' r lcount* →
⟨*bool-rel* ×$_f$ *twl-st-heur-conflict-ana' r lcount*⟩*nres-rel*›
supply [[*goals-limit=1*]]
unfolding *tl-state-wl-heur-def mop-tl-state-wl-def*
apply (*intro frefI nres-relI*)
apply *refine-vcg*
subgoal for *x y*
using *mop-tl-state-wl-pre-tl-state-wl-heur-pre*[*of x y*] **by** *simp*
subgoal for *x y*
using *mop-tl-state-wl-pre-tl-state-wl-heur-pre*[*of x y*] **by** *simp*
subgoal for *x y*
apply (*cases y*)
apply (*auto simp*: *twl-st-heur-conflict-ana-def tl-state-wl-heur-def tl-state-wl-def vmtf-unset-vmtf-tl*
mop-tl-state-wl-pre-simps lit-of-last-trail-pol-lit-of-last-trail[*THEN fref-to-Down-unRET-Id*]
card-max-lvl-tl tl-state-out-learned
dest: *no-dup-tlD simp flip*: *all-lits-st-alt-def*
intro!: *tl-trail-tr*[*THEN fref-to-Down-unRET*] *isa-bump-unset-vmtf-tl*)
apply (*subst lit-of-last-trail-pol-lit-of-last-trail*[*THEN fref-to-Down-unRET-Id*])
apply (*auto simp*: *mop-tl-state-wl-pre-simps lit-of-hd-trail-def mop-tl-state-wl-pre-simps counts-maximum-level-def*
intro!: *isa-bump-unset-vmtf-tl*[*THEN order-trans*] *IsaSAT-Trail.tl-trail-tr*[*THEN fref-to-Down-unRET*]
intro: *no-dup-tlD*)
apply (*metis* (*no-types*, *lifting*) *IsaSAT-Setup.all-lits-st-alt-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff mop-tl-state-wl-pre-simps*(*2*
apply (*metis* (*no-types*, *lifting*) *IsaSAT-Setup.all-lits-st-alt-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff mop-tl-state-wl-pre-simps*(*2*
apply (*rule no-dup-tlD*, *assumption*)
apply (*subst card-max-lvl-tl*)
apply (*auto simp*: *mop-tl-state-wl-pre-simps lookup-clause-rel-not-tautolgy lookup-clause-rel-distinct-mset*
option-lookup-clause-rel-def)
apply (*subst tl-state-out-learned*)
apply (*auto simp*: *mop-tl-state-wl-pre-simps lookup-clause-rel-not-tautolgy lookup-clause-rel-distinct-mset*
option-lookup-clause-rel-def)
apply (*subst tl-state-out-learned*)
apply (*auto simp*: *mop-tl-state-wl-pre-simps lookup-clause-rel-not-tautolgy lookup-clause-rel-distinct-mset*
option-lookup-clause-rel-def)
done
done

lemma *arena-act-pre-mark-used*:
‹*arena-act-pre arena C* ⟹
arena-act-pre (*mark-used arena C*) *C*›
unfolding *arena-act-pre-def arena-is-valid-clause-idx-def*

apply *clarify*
apply (*rule-tac x=N* **in** *exI*)
apply (*rule-tac x=vdom* **in** *exI*)
by (*auto simp*: *arena-act-pre-def*
simp: *valid-arena-mark-used*)

definition (**in** −) *get-max-lvl-st* :: ‹*nat twl-st-wl* ⇒ *nat literal* ⇒ *nat*› **where**
‹*get-max-lvl-st S L* = *get-maximum-level-remove* (*get-trail-wl S*) (*the* (*get-conflict-wl S*)) *L*›

lemma *card-max-lvl-remove1-mset-hd*:
‹−*lit-of* (*hd M*) ∈# *y* ⟹ *is-proped* (*hd M*) ⟹
card-max-lvl M (*remove1-mset* (−*lit-of* (*hd M*)) *y*) = *card-max-lvl M y* − *1*›
by (*cases M*) (*auto dest*!: *multi-member-split simp*: *card-max-lvl-add-mset*)

lemma *update-confl-tl-wl-heur-state-helper*:
‹(*L*, *C*) = *lit-and-ann-of-propagated* (*hd* (*get-trail-wl S*)) ⟹ *get-trail-wl S* ≠ [] ⟹
is-proped (*hd* (*get-trail-wl S*)) ⟹ *L* = *lit-of* (*hd* (*get-trail-wl S*))›
by (*cases S*; *cases* ‹*hd* (*get-trail-wl S*)›) *auto*

lemma (**in** −) *not-ge-Suc0*: ‹¬*Suc 0* ≤ *n* ⟷ *n* = *0*›
by *auto*

definition *update-confl-tl-wl-pre'* :: ‹((*nat literal* × *nat*) × *nat twl-st-wl*) ⇒ *bool*› **where**
‹*update-confl-tl-wl-pre'* = (λ((*L*, *C*), *S*).
C ∈# *dom-m* (*get-clauses-wl S*) ∧
get-conflict-wl S ≠ *None* ∧ *get-trail-wl S* ≠ [] ∧
− *L* ∈# *the* (*get-conflict-wl S*) ∧
L ∉# *the* (*get-conflict-wl S*) ∧
(*L*, *C*) = *lit-and-ann-of-propagated* (*hd* (*get-trail-wl S*)) ∧
L ∈# *all-lits-st S* ∧
is-proped (*hd* (*get-trail-wl S*)) ∧
C > *0* ∧
card-max-lvl (*get-trail-wl S*) (*the* (*get-conflict-wl S*)) ≥ *1* ∧
distinct-mset (*the* (*get-conflict-wl S*)) ∧
− *L* ∉ *set* (*get-clauses-wl S* ∝ *C*) ∧
(*length* (*get-clauses-wl S* ∝ *C*) ≠ *2* ⟶
L ∉ *set* (*tl* (*get-clauses-wl S* ∝ *C*)) ∧
get-clauses-wl S ∝ *C* ! *0* = *L* ∧
mset (*tl* (*get-clauses-wl S* ∝ *C*)) = *remove1-mset L* (*mset* (*get-clauses-wl S* ∝ *C*)) ∧
(∀ *L*∈*set* (*tl*(*get-clauses-wl S* ∝ *C*)). − *L* ∉# *the* (*get-conflict-wl S*)) ∧
card-max-lvl (*get-trail-wl S*) (*mset* (*tl* (*get-clauses-wl S* ∝ *C*)) ∪# *the* (*get-conflict-wl S*)) =
card-max-lvl (*get-trail-wl S*) (*remove1-mset L* (*mset* (*get-clauses-wl S* ∝ *C*)) ∪# *the* (*get-conflict-wl S*))) ∧
L ∈ *set* (*watched-l* (*get-clauses-wl S* ∝ *C*)) ∧
distinct (*get-clauses-wl S* ∝ *C*) ∧
¬*tautology* (*the* (*get-conflict-wl S*)) ∧
¬*tautology* (*mset* (*get-clauses-wl S* ∝ *C*)) ∧
¬*tautology* (*remove1-mset L* (*remove1-mset* (− *L*)
((*the* (*get-conflict-wl S*) ∪# *mset* (*get-clauses-wl S* ∝ *C*))))) ∧
count-decided (*get-trail-wl S*) > *0* ∧
literals-are-in-$\mathcal{L}_{in}$ (*all-atms-st S*) (*the* (*get-conflict-wl S*)) ∧
literals-are-$\mathcal{L}_{in}$ (*all-atms-st S*) *S* ∧
literals-are-in-$\mathcal{L}_{in}$*-trail* (*all-atms-st S*) (*get-trail-wl S*) ∧
(∀ *K*. *K* ∈# *remove1-mset L* (*mset* (*get-clauses-wl S* ∝ *C*)) ⟶ − *K* ∉# *the* (*get-conflict-wl S*)) ∧
size (*remove1-mset L* (*mset* (*get-clauses-wl S* ∝ *C*)) ∪# *the* (*get-conflict-wl S*)) > *0* ∧

Suc 0 ≤ *card-max-lvl* (*get-trail-wl S*) (*remove1-mset L* (*mset* (*get-clauses-wl S* ∝ *C*)) ∪# *the* (*get-conflict-wl S*)) ∧
size (*remove1-mset L* (*mset* (*get-clauses-wl S* ∝ *C*)) ∪# *the* (*get-conflict-wl S*)) =
size (*the* (*get-conflict-wl S*) ∪# *mset* (*get-clauses-wl S* ∝ *C*) − {#*L*, − *L*#}) + *Suc 0* ∧
lit-of (*hd* (*get-trail-wl S*)) = *L* ∧
card-max-lvl (*get-trail-wl S*) ((*mset* (*get-clauses-wl S* ∝ *C*) − *unmark* (*hd* (*get-trail-wl S*))) ∪# *the* (*get-conflict-wl S*)) =
card-max-lvl (*tl* (*get-trail-wl S*)) (*the* (*get-conflict-wl S*) ∪# *mset* (*get-clauses-wl S* ∝ *C*) − {#*L*, − *L*#}) + *Suc 0* ∧
out-learned (*tl* (*get-trail-wl S*)) (*Some* (*the* (*get-conflict-wl S*) ∪# *mset* (*get-clauses-wl S* ∝ *C*) − {#*L*, − *L*#})) =
out-learned (*get-trail-wl S*) (*Some* ((*mset* (*get-clauses-wl S* ∝ *C*) − *unmark* (*hd* (*get-trail-wl S*))) ∪# *the* (*get-conflict-wl S*)))
)›

lemma *remove1-mset-union-distrib1*:
‹*L* ∉# *B* ⟹ *remove1-mset L* (*A* ∪# *B*) = *remove1-mset L A* ∪# *B*› **and**
remove1-mset-union-distrib2:
‹*L* ∉# *A* ⟹ *remove1-mset L* (*A* ∪# *B*) = *A* ∪# *remove1-mset L B*›
by (*auto simp*: *remove1-mset-union-distrib*)

lemma *update-confl-tl-wl-pre-update-confl-tl-wl-pre'*:
assumes ‹*update-confl-tl-wl-pre L C S*›
shows ‹*update-confl-tl-wl-pre'* ((*L*, *C*), *S*)›
proof −
obtain *x xa* **where**
Sx: ‹(*S*, *x*) ∈ *state-wl-l None*› **and**
‹*correct-watching S*› **and**
x-xa: ‹(*x*, *xa*) ∈ *twl-st-l None*› **and**
st-invs: ‹*twl-struct-invs xa*› **and**
list-invs: ‹*twl-list-invs x*› **and**
‹*twl-stgy-invs xa*› **and**
C: ‹*C* ∈# *dom-m* (*get-clauses-wl S*)› **and**
nempty: ‹*get-trail-wl S* ≠ []› **and**
‹*literals-to-update-wl S* = {#}› **and**
hd: ‹*hd* (*get-trail-wl S*) = *Propagated L C*› **and**
C-0: ‹*0* < *C*› **and**
confl: ‹*get-conflict-wl S* ≠ *None*› **and**
‹*0* < *count-decided* (*get-trail-wl S*)› **and**
‹*get-conflict-wl S* ≠ *Some* {#}› **and**
‹*L* ∈ *set* (*get-clauses-wl S* ∝ *C*)› **and**
uL-D: ‹− *L* ∈# *the* (*get-conflict-wl S*)› **and**
xa: ‹*hd* (*get-trail xa*) = *Propagated L* (*mset* (*get-clauses-wl S* ∝ *C*))› **and**
L: ‹*L* ∈# *all-lits-st S*› **and**
blits: ‹*blits-in-*$\mathcal{L}_{in}$ *S*›
using *assms*
unfolding *update-confl-tl-wl-pre-def*
update-confl-tl-l-pre-def update-confl-tl-pre-def
prod.case **apply** −
by *normalize-goal+*
(*simp flip*: *all-lits-def all-lits-alt-def2*
add: *mset-take-mset-drop-mset'* $\mathcal{L}_{all}$*-all-atms-all-lits*)

have
dist: ‹$cdcl_W$*-restart-mset.distinct-cdcl*$_W$*-state* (*state*$_W$*-of xa*)› **and**

M-lev: ‹$cdcl_W$-*restart-mset.cdcl*$_W$-*M-level-inv* (*state*$_W$-*of xa*)› **and**
confl': ‹$cdcl_W$-*restart-mset.cdcl*$_W$-*conflicting* (*state*$_W$-*of xa*)› **and**
st-inv: ‹*twl-st-inv xa*›
using *st-invs* **unfolding** *twl-struct-invs-def* $cdcl_W$-*restart-mset.cdcl*$_W$-*all-struct-inv-def*
pcdcl-all-struct-invs-def *state*$_W$-*of-def*[*symmetric*]
by *fast+*

have *eq*: ‹*lits-of-l* (*tl* (*get-trail-wl S*)) = *lits-of-l* (*tl* (*get-trail xa*))›
using *Sx x-xa* **unfolding** *list.set-map*[*symmetric*] *lits-of-def*
by (*cases S*; *cases x*; *cases xa*;
auto simp: *state-wl-l-def twl-st-l-def map-tl list-of-l-convert-map-lit-of simp del*: *list.set-map*)

have *card-max*: ‹*card-max-lvl* (*get-trail-wl S*) (*the* (*get-conflict-wl S*)) ≥ *1*›
using *hd uL-D nempty* **by** (*cases* ‹*get-trail-wl S*›; *auto dest*!: *multi-member-split simp*: *card-max-lvl-def*)

have *dist-C*: ‹*distinct-mset* (*the* (*get-conflict-wl S*))›
using *dist Sx x-xa confl C* **unfolding** $cdcl_W$-*restart-mset.distinct-cdcl*$_W$-*state-def*
by (*auto simp*: *twl-st*)
have *dist*: ‹*distinct* (*get-clauses-wl S* ∝ *C*)›
using *dist Sx x-xa confl C* **unfolding** $cdcl_W$-*restart-mset.distinct-cdcl*$_W$-*state-alt-def*
by (*auto simp*: *image-image ran-m-def dest*!: *multi-member-split*)

have *n-d*: ‹*no-dup* (*get-trail-wl S*)›
using *Sx x-xa M-lev* **unfolding** $cdcl_W$-*restart-mset.cdcl*$_W$-*M-level-inv-def*
by (*auto simp*: *twl-st*)
have *CNot-D*: ‹*get-trail-wl S* ⊨*as CNot* (*the* (*get-conflict-wl S*))›
using *confl' confl Sx x-xa* **unfolding** $cdcl_W$-*restart-mset.cdcl*$_W$-*conflicting-def*
by (*auto simp*: *twl-st*)
then have ‹*tl* (*get-trail-wl S*) ⊨*as CNot* (*the* (*get-conflict-wl S*) − {#−*L*#})›
using *dist-C uL-D n-d hd nempty* **by** (*cases* ‹*get-trail-wl S*›)
(*auto dest*!: *multi-member-split simp*: *true-annots-true-cls-def-iff-negation-in-model*)
moreover have *CNot-C'*: ‹*tl* (*get-trail-wl S*) ⊨*as CNot* (*mset* (*get-clauses-wl S* ∝ *C*) − {#*L*#})›
and
L-C: ‹*L* ∈# *mset* (*get-clauses-wl S* ∝ *C*)›
using *confl' nempty x-xa xa hd Sx nempty eq*
unfolding $cdcl_W$-*restart-mset.cdcl*$_W$-*conflicting-def*
by (*cases* ‹*get-trail xa*›; *fastforce simp*: *twl-st twl-st-l true-annots-true-cls-def-iff-negation-in-model*
dest: *spec*[*of* - ‹[]›])+

ultimately have *tl*: ‹*tl* (*get-trail-wl S*) ⊨*as CNot* ((*the* (*get-conflict-wl S*) − {#−*L*#}) ∪# (*mset* (*get-clauses-wl S* ∝ *C*) − {#*L*#}))›
by (*auto simp*: *true-annots-true-cls-def-iff-negation-in-model*)
then have ‹(*the* (*get-conflict-wl S*) − {#−*L*#}) ∪# (*mset* (*get-clauses-wl S* ∝ *C*) − {#*L*#}) =
(*the* (*get-conflict-wl S*) ∪# *mset* (*get-clauses-wl S* ∝ *C*) −
{#*L*, − *L*#})›
using *multi-member-split*[*OF L-C*] *uL-D dist dist-C n-d hd nempty*
apply (*cases* ‹*get-trail-wl S*›; *auto dest*!: *multi-member-split*
simp: *true-annots-true-cls-def-iff-negation-in-model*)
apply (*subst sup-union-left1*)
apply (*auto dest*!: *multi-member-split dest*: *in-lits-of-l-defined-litD*)
done
with *tl* **have** ‹*tl* (*get-trail-wl S*) ⊨*as CNot* (*the* (*get-conflict-wl S*) ∪# *mset* (*get-clauses-wl S* ∝ *C*)
−
{#*L*, − *L*#})› **by** *simp*
with *distinct-consistent-interp*[*OF no-dup-tlD*[*OF n-d*]] **have** *1*: ‹¬*tautology*
(*the* (*get-conflict-wl S*) ∪# *mset* (*get-clauses-wl S* ∝ *C*) −

```
    {#L, - L#})›
   unfolding true-annots-true-cls
   by (rule consistent-CNot-not-tautology)
  have False if ‹- L ∈# mset (get-clauses-wl S ∝ C)›
    using multi-member-split[OF L-C] hd nempty n-d CNot-C' multi-member-split[OF that]
    by (cases ‹get-trail-wl S›; auto dest!: multi-member-split
        simp: add-mset-eq-add-mset true-annots-true-cls-def-iff-negation-in-model
        dest!: in-lits-of-l-defined-litD)
  then have 2: ‹-L ∉ set (get-clauses-wl S ∝ C)›
    by auto

  have ‹length (get-clauses-wl S ∝ C) ≥ 2›
   using st-inv C  Sx x-xa by (cases xa; cases x; cases S; cases ‹irred (get-clauses-wl S) C›;
     auto simp: twl-st-inv.simps state-wl-l-def twl-st-l-def conj-disj-distribR Collect-disj-eq image-Un
       ran-m-def Collect-conv-if dest!: multi-member-split)
  then have [simp]: ‹length (get-clauses-wl S ∝ C) ≠ 2 ⟷ length (get-clauses-wl S ∝ C) > 2›
   by (cases ‹get-clauses-wl S ∝ C›; cases ‹tl (get-clauses-wl S ∝ C)›;
     auto simp: twl-list-invs-def all-conj-distrib dest: in-set-takeD)


  have CNot-C:  ‹¬tautology (mset (get-clauses-wl S ∝ C))›
   using CNot-C' Sx hd nempty C-0 dist multi-member-split[OF L-C] dist
      consistent-CNot-not-tautology[OF distinct-consistent-interp[OF no-dup-tlD[OF n-d]]
        CNot-C'[unfolded true-annots-true-cls]] 2
   unfolding true-annots-true-cls-def-iff-negation-in-model
   apply (auto simp: tautology-add-mset dest: arg-cong[of ‹mset -› - set-mset])
   by (metis member-add-mset set-mset-mset)


  have stupid: ‹K ∈# removeAll-mset L D ⟷ K ≠ L ∧ K ∈# D› for K L D
   by auto
  have K: ‹K ∈# remove1-mset L (mset (get-clauses-wl S ∝ C)) ⟹ - K ∉# the (get-conflict-wl S)›
and
    uL-C: ‹-L ∉# (mset (get-clauses-wl S ∝ C))› for K
   apply (subst (asm) distinct-mset-remove1-All)
   subgoal using dist by auto
   apply (subst (asm)stupid)
   apply (rule conjE, assumption)
  apply (drule multi-member-split)
   using 1 uL-D CNot-C dist 2[unfolded in-multiset-in-set[symmetric]] dist-C
     consistent-CNot-not-tautology[OF distinct-consistent-interp[OF n-d]
        CNot-D[unfolded true-annots-true-cls]]  ‹L ∈# mset (get-clauses-wl S ∝ C)›
    by (auto dest!: multi-member-split[of ‹-L›] multi-member-split in-set-takeD
      simp: tautology-add-mset add-mset-eq-add-mset uminus-lit-swap diff-union-swap2
      simp del: set-mset-mset in-multiset-in-set
        distinct-mset-mset-distinct simp flip: distinct-mset-mset-distinct)
 have size: ‹size (remove1-mset L (mset (get-clauses-wl S ∝ C)) ∪# the (get-conflict-wl S)) > 0›
   using uL-D uL-C by (auto dest!: multi-member-split)

  have max-lvl: ‹Suc 0 ≤ card-max-lvl (get-trail-wl S) (remove1-mset L (mset (get-clauses-wl S ∝ C))
∪# the (get-conflict-wl S))›
   using uL-D hd nempty uL-C by (cases ‹get-trail-wl S›; auto simp: card-max-lvl-def dest!: multi-member-split)


  have s: ‹size (remove1-mset L (mset (get-clauses-wl S ∝ C)) ∪# the (get-conflict-wl S)) =
      size (the (get-conflict-wl S) ∪# mset (get-clauses-wl S ∝ C) - {#L, - L#}) + 1›
```

using *uL-D hd nempty uL-C dist-C* **apply** (*cases ‹get-trail-wl S›*; *auto simp*: *dest*!: *multi-member-split*)
by (*metis* (*no-types*, *opaque-lifting*) *‹remove1-mset (− L) (the (get-conflict-wl S)) ∪# remove1-mset L (mset (get-clauses-wl S ∝ C)) = the (get-conflict-wl S) ∪# mset (get-clauses-wl S ∝ C) − {#L, − L#}›
add-mset-commute add-mset-diff-bothsides add-mset-remove-trivial set-mset-mset subset-mset.sup-commute sup-union-left1*)

have *SC-0*: *‹length (get-clauses-wl S ∝ C) > 2 ⟹ L ∉ set (tl (get-clauses-wl S ∝ C)) ∧ get-clauses-wl S ∝ C ! 0 = L ∧*
mset (tl (get-clauses-wl S ∝ C)) = remove1-mset L (mset (get-clauses-wl S ∝ C)) ∧
(∀ L∈set (tl(get-clauses-wl S ∝ C)). − L ∉# the (get-conflict-wl S)) ∧
card-max-lvl (get-trail-wl S) (mset (tl (get-clauses-wl S ∝ C)) ∪# the (get-conflict-wl S)) =
card-max-lvl (get-trail-wl S) (remove1-mset L (mset (get-clauses-wl S ∝ C)) ∪# the (get-conflict-wl S))›
‹L ∈ set (watched-l (get-clauses-wl S ∝ C))›
‹L ∈# mset (get-clauses-wl S ∝ C)›
using *list-invs Sx hd nempty C-0 dist L-C K*
by (*cases ‹get-trail-wl S›*; *cases ‹get-clauses-wl S ∝ C›*;
auto simp: *twl-list-invs-def all-conj-distrib dest*: *in-set-takeD*; *fail*)+

have *max*: *‹card-max-lvl (get-trail-wl S) ((mset (get-clauses-wl S ∝ C) − unmark (hd (get-trail-wl S))) ∪# the (get-conflict-wl S)) =*
card-max-lvl (tl (get-trail-wl S)) (the (get-conflict-wl S) ∪# mset (get-clauses-wl S ∝ C) − {#L, − L#}) + Suc 0›
using *multi-member-split*[*OF uL-D*] *L-C hd nempty n-d dist dist-C 1 ‹0 < count-decided (get-trail-wl S)› uL-C n-d*
consistent-CNot-not-tautology[*OF distinct-consistent-interp*[*OF n-d*]
CNot-D[*unfolded true-annots-true-cls*]] *CNot-C* **apply** (*cases ‹get-trail-wl S›*;
auto simp: *card-max-lvl-Cons card-max-lvl-add-mset subset-mset.sup-commute*[*of - ‹remove1-mset - -›]*
subset-mset.sup-commute[*of - ‹mset -›] tautology-add-mset*)
apply (*subst card-max-lvl-Cons*)
apply (*auto simp*: *card-max-lvl-Cons card-max-lvl-add-mset subset-mset.sup-commute*[*of - ‹remove1-mset - -›]*
subset-mset.sup-commute[*of - ‹mset -›] tautology-add-mset remove1-mset-union-distrib1*)
using *K uL-D* **apply** *presburger*
done

have *xx*: *‹out-learned (tl (get-trail-wl S)) (Some (the (get-conflict-wl S) ∪# mset (get-clauses-wl S ∝ C) − {#L, − L#})) outl ⟷*
out-learned (get-trail-wl S) (Some (the (get-conflict-wl S) ∪# mset (get-clauses-wl S ∝ C) − {#L, − L#})) outl› **for** *outl*
apply (*subst tl-state-out-learned*)
apply (*cases ‹get-trail-wl S›*; *use L-C hd nempty dist dist-C* **in** *auto*)
apply (*meson distinct-mem-diff-mset distinct-mset-mset-distinct distinct-mset-union-mset union-single-eq-member*)
apply (*cases ‹get-trail-wl S›*; *use L-C hd nempty dist dist-C* **in** *auto*)
apply (*metis add-mset-commute distinct-mem-diff-mset distinct-mset-mset-distinct distinct-mset-union-mset union-single-eq-member*)
apply (*cases ‹get-trail-wl S›*; *use L-C hd nempty dist dist-C* **in** *auto*)
..

have [*simp*]: *‹get-level (get-trail-wl S) L = count-decided (get-trail-wl S)›*
by (*cases ‹get-trail-wl S›*; *use L-C hd nempty dist dist-C* **in** *auto*)
have *yy*: *‹out-learned (get-trail-wl S) (Some ((the (get-conflict-wl S) ∪# mset (get-clauses-wl S ∝ C)) − {#L, − L#})) outl ⟷*

```
    out-learned (get-trail-wl S) (Some ((mset (get-clauses-wl S ∝ C) − unmark (hd (get-trail-wl S)))
∪# the (get-conflict-wl S))) outl› for outl
  by (use L-C hd nempty dist dist-C in ‹auto simp add: out-learned-def ac-simps›)

 have xx: ‹out-learned (tl (get-trail-wl S)) (Some (the (get-conflict-wl S) ∪# mset (get-clauses-wl S ∝
C) − {#L, − L#})) =
    out-learned (get-trail-wl S) (Some ((mset (get-clauses-wl S ∝ C) − unmark (hd (get-trail-wl S)))
∪# the (get-conflict-wl S)))›
   using xx yy by (auto intro!: ext)
 show ?thesis
   using Sx x-xa C C-0 confl nempty uL-D L blits 1 2 card-max dist-C dist SC-0 max
     consistent-CNot-not-tautology[OF distinct-consistent-interp[OF n-d]
       CNot-D[unfolded true-annots-true-cls]] CNot-C K xx
     ‹0 < count-decided (get-trail-wl S)› size max-lvl s
    literals-are-ℒ_in-literals-are-in-ℒ_in-conflict[OF Sx st-invs x-xa - , of ‹all-atms-st S›]
    literals-are-ℒ_in-literals-are-ℒ_in-trail[OF Sx st-invs x-xa - , of ‹all-atms-st S›]
   unfolding update-confl-tl-wl-pre'-def
   by (clarsimp simp flip: all-lits-def simp add: hd mset-take-mset-drop-mset' ℒ_all-all-atms-all-lits)

qed

lemma (in −)out-learned-add-mset-highest-level:
  ‹L = lit-of (hd M) ⟹ out-learned M (Some (add-mset (− L) A)) outl ⟷
   out-learned M (Some A) outl›
 by (cases M)
   (auto simp: out-learned-def get-level-cons-if atm-of-eq-atm-of count-decided-ge-get-level
     uminus-lit-swap cong: if-cong
     intro!: filter-mset-cong2)

lemma (in −)out-learned-tl-Some-notin:
  ‹is-proped (hd M) ⟹ lit-of (hd M) ∉# C ⟹ −lit-of (hd M) ∉# C ⟹
   out-learned M (Some C) outl ⟷ out-learned (tl M) (Some C) outl›
 by (cases M) (auto simp: out-learned-def get-level-cons-if atm-of-eq-atm-of
     intro!: filter-mset-cong2)


lemma update-confl-tl-wl-heur-update-confl-tl-wl:
 ‹(uncurry2 (update-confl-tl-wl-heur), uncurry2 mop-update-confl-tl-wl) ∈
 [λ-. True]_f
  Id ×_f nat-rel ×_f twl-st-heur-conflict-ana' r lcount →
   ⟨bool-rel ×_f twl-st-heur-conflict-ana' r lcount⟩nres-rel›
proof −
 have mop-update-confl-tl-wl-alt-def: ‹mop-update-confl-tl-wl = (λL C (M, N, D, NE, UE, WS, Q).
do {
     ASSERT(update-confl-tl-wl-pre L C (M, N, D, NE, UE, WS, Q));
     N ← calculate-LBD-st M N C;
     D ← RETURN (resolve-cls-wl' (M, N, D, NE, UE, WS, Q) C L);
     N ← RETURN N;
     - ← RETURN ();
     N ← RETURN N;
     - ← RETURN ();
     RETURN (False, (tl M, N, Some D, NE, UE, WS, Q))
   })›
   by (auto simp: mop-update-confl-tl-wl-def update-confl-tl-wl-def calculate-LBD-st-def
     intro!: ext)
 define rr where
```

```
  ‹rr L M N C b n xs clvls outl = do {
        ((b, (n, xs)), clvls, outl) ←
          if arena-length N C = 2 then isasat-lookup-merge-eq2 L M N C (b, (n, xs)) clvls outl
          else isa-resolve-merge-conflict-gt2 M N C (b, (n, xs)) clvls outl;
        ASSERT(curry lookup-conflict-remove1-pre L (n, xs) ∧ clvls ≥ 1);
        let (nxs) = lookup-conflict-remove1 L (n, xs);
        RETURN ((b, (nxs)), clvls, outl) }›
    for  L M N C b n xs clvls lbd outl
  have update-confl-tl-wl-heur-alt-def:
    ‹update-confl-tl-wl-heur = (λL C S. do {
      let M = get-trail-wl-heur S;
      let N = get-clauses-wl-heur S;
      let lbd = get-lbd S;
      let outl = get-outlearned-heur S;
      let clvls = get-count-max-lvls-heur S;
      let vm = get-vmtf-heur S;
      let (b, (n, xs)) = get-conflict-wl-heur S;
      (N, lbd) ← calculate-LBD-heur-st M N lbd C;
      ASSERT (clvls ≥ 1);
      let L' = atm-of L;
      ASSERT(arena-is-valid-clause-idx N C);
      ((b, (n, xs)), clvls, outl) ← rr L M N C b n xs clvls outl;
      ASSERT(arena-act-pre N C);
      vm ← isa-vmtf-bump-to-rescore-also-reasons-cl M N C (−L) vm;
      ASSERT(isa-bump-unset-pre L' vm);
      ASSERT(tl-trailt-tr-pre M);
      vm ← isa-bump-unset L' vm;
      let S = set-trail-wl-heur (tl-trailt-tr M) S;
      let S = set-conflict-wl-heur (b, (n, xs)) S;
      let S = set-vmtf-wl-heur vm S;
      let S = set-count-max-wl-heur (clvls − 1) S;
      let S = set-outl-wl-heur outl S;
      let S = set-clauses-wl-heur N S;
      let S = set-lbd-wl-heur lbd S;
      RETURN (False, S)
    })›
  by (auto simp: update-confl-tl-wl-heur-def Let-def rr-def intro!: ext bind-cong[OF refl])

note [[goals-limit=1]]
  have rr: ‹(((x1g, x2g), S),
     (x1, x2), x1a, x1b, x1c, x1d, x1e, NS, US, x1f, x2f)
    ∈ nat-lit-lit-rel ×f nat-rel ×f twl-st-heur-conflict-ana' r lcount ⟹
    CLS = ((x1, x2), x1a, x1b, x1c, x1d, x1e, NS, US, x1f, x2f) ⟹
    CLS' = ((x1g, x2g), S, s, t) ⟹
    update-confl-tl-wl-pre x1 x2 (x1a, x1b, x1c, x1d, x1e, NS, US, x1f, x2f) ⟹
    1 ≤ x1q ⟹
    arena-is-valid-clause-idx x1i x2g ⟹
    (x1k, (x1l, x2k)) = get-conflict-wl-heur S ⟹
    rr x1g (get-trail-wl-heur S) (get-clauses-wl-heur S) x2g x1k x1l x2k (get-count-max-lvls-heur S)
     (get-outlearned-heur S)
    ≤ ⇓ {((C, clvls, outl), D). (C, Some D) ∈ option-lookup-clause-rel (all-atms-st (x1a, x1b, x1c, x1d,
x1e, NS, US, x1f, x2f)) ∧
         clvls = card-max-lvl x1a (remove1-mset x1 (mset (x1b ∝ x2)) ∪# the x1c) ∧
        out-learned x1a (Some (remove1-mset x1 (mset (x1b ∝ x2)) ∪# the x1c)) (outl) ∧
        size (remove1-mset x1 (mset (x1b ∝ x2)) ∪# the x1c) =
          size ((mset (x1b ∝ x2)) ∪# the x1c − {#x1, −x1#}) + Suc 0 ∧
```

D = *resolve-cls-wl′* (*x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *NS*, *US*, *x1f*, *x2f*) *x2 x1*}
(*RETURN* (*resolve-cls-wl′* (*x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *NS*, *US*, *x1f*, *x2f*) *x2 x1*))›
for *m n p q ra s t x1 x2 x1a x1b x1c x1d x1e x1f x2f x1g x2g x1h x1i x1k*
x1l x2k x1m x1n x1p x1q x1r x1t CLS CLS′ NS US x1s S
unfolding *rr-def*
apply (*refine-vcg lhs-step-If*)
apply (*rule isasat-lookup-merge-eq2-lookup-merge-eq2*[**where**
vdom = ‹*set* (*get-vdom S*)› **and** *M* = *x1a* **and** *N* = *x1b* **and** *C* = *x1c* **and**
$\mathcal{A}$ = ‹*all-atms-st* (*x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *NS*, *US*, *x1f*, *x2f*) ›, *THEN order-trans*])
subgoal by (*auto simp*: *twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′ simp*: *update-confl-tl-wl-pre′-def*)
subgoal by (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-mm-def all-lits-st-def all-lits-of-mm-union*
all-lits-def
simp flip: *all-lits-st-alt-def*)
subgoal by (*auto simp*: *twl-st-heur-conflict-ana-def*)
subgoal by (*auto simp*: *twl-st-heur-conflict-ana-def*)
subgoal by (*auto simp*: *twl-st-heur-conflict-ana-def*)
subgoal unfolding *Down-id-eq*
apply (*rule lookup-merge-eq2-spec*[**where** *M* = *x1a* **and** *C* = ‹*the x1c*› **and**
$\mathcal{A}$ = ‹*all-atms-st* (*x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *NS*, *US*, *x1f*, *x2f*)›, *THEN order-trans*])
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def literals-are-in-*$\mathcal{L}_{in}$*-mm-def all-lits-st-def all-lits-of-mm-union*
all-lits-def simp flip: *all-lits-st-alt-def*
intro!: *literals-are-in-*$\mathcal{L}_{in}$*-mm-literals-are-in-*$\mathcal{L}_{in}$)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def twl-st-heur-conflict-ana-def counts-maximum-level-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def arena-lifting twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def arena-lifting twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def merge-conflict-m-eq2-def conc-fun-SPEC lookup-conflict-remove1-pre-def*
atms-of-def
option-lookup-clause-rel-def lookup-clause-rel-def resolve-cls-wl′-def lookup-conflict-remove1-def
remove1-mset-union-distrib1 remove1-mset-union-distrib2 subset-mset.sup-commute[*of* - ‹*remove1-mset* - -›]
subset-mset.sup-commute[*of* - ‹*mset* (- ∝ -)›]
simp flip: *all-lits-st-alt-def*[*symmetric*]

intro!: *mset-as-position-remove3*
intro!: *ASSERT-leI*)
done
subgoal
apply (*subst* (*asm*) *arena-lifting*(*4*)[**where** *vdom* = ‹*set* (*get-vdom-aivdom* (*get-aivdom S*))› **and** *N* = *x1b*, *symmetric*])
subgoal by (*auto simp*: *twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def*)
apply (*rule isa-resolve-merge-conflict-gt2*[**where**
$\mathcal{A}$ = ‹*all-atms-st* (*x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *NS*, *US*, *x1f*, *x2f*)› **and** *vdom* = ‹*set* (*get-vdom-aivdom* (*get-aivdom S*))›,
THEN fref-to-Down-curry5, *of x1a x1b x2g x1c* ‹*get-count-max-lvls-heur S*› ‹*get-outlearned-heur S*›,
THEN order-trans])
subgoal unfolding *merge-conflict-m-pre-def*
by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def twl-st-heur-conflict-ana-def counts-maximum-level-def*)
subgoal by (*auto simp*: *twl-st-heur-conflict-ana-def*)
subgoal
by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def conc-fun-SPEC lookup-conflict-remove1-pre-def atms-of-def*
option-lookup-clause-rel-def lookup-clause-rel-def resolve-cls-wl′-def
merge-conflict-m-def lookup-conflict-remove1-def subset-mset.sup-commute[*of* - ‹*mset* (- ∝ -)›]
remove1-mset-union-distrib1 remove1-mset-union-distrib2
simp flip: *all-lits-st-alt-def*[*symmetric*]
intro!: *mset-as-position-remove3*
intro!: *ASSERT-leI*)
done
done
have *rr*: ‹(((*x1g*, *x2g*), *S*),
(*x1*, *x2*), *x1a*, *N*, *x1c*, *x1d*, *x1e*, *x1f*, *ha*, *ia*, *ja*)
∈ *nat-lit-lit-rel* ×$_f$ *nat-rel* ×$_f$ *twl-st-heur-conflict-ana′ r lcount* ⟹
CLS = ((*x1*, *x2*), *x1a*, *N*, *x1c*, *x1d*, *x1e*, *x1f*, *ha*, *ia*, *ja*) ⟹
CLS′ = ((*x1g*, *x2g*), *S*) ⟹
get-conflict-wl-heur S = (*x1k*, (*x1l*, *x2k*)) ⟹
update-confl-tl-wl-pre x1 x2
(*x1a*, *N*, *x1c*, *x1d*, *x1e*, *x1f*, *ha*, *ia*, *ja*) ⟹
((*x1t*, *x2t* :: *bool list*), *x1b*)
∈ {((*arena′*, *lbd*), *N′*).
valid-arena arena′ N′
(*set* (*get-vdom*
(*snd* ((*x1g*, *x2g*), *S*)))) ∧
N = *N′* ∧ *length x1i* = *length arena′*} ⟹
1 ≤ *x1p* ⟹
arena-is-valid-clause-idx x1t x2g ⟹
rr x1g (*get-trail-wl-heur S*) *x1t x2g x1k x1l x2k* (*get-count-max-lvls-heur S*)
(*get-outlearned-heur S*)
≤ ⇓ {((*C*, *clvls*, *outl*), *D*). (*C*, *Some D*) ∈ *option-lookup-clause-rel* (*all-atms-st* (*x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *x1f*, *ha*, *ia*, *ja*)) ∧
clvls = *card-max-lvl x1a* (*remove1-mset x1* (*mset* (*x1b* ∝ *x2*)) ∪# *the x1c*) ∧
out-learned x1a (*Some* (*remove1-mset x1* (*mset* (*x1b* ∝ *x2*)) ∪# *the x1c*)) (*outl*) ∧
size (*remove1-mset x1* (*mset* (*x1b* ∝ *x2*)) ∪# *the x1c*) =
size ((*mset* (*x1b* ∝ *x2*)) ∪# *the x1c* − {#*x1*, −*x1*#}) + *Suc 0* ∧
D = *resolve-cls-wl′* (*x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *x1f*, *ha*, *ia*, *ja*) *x2 x1*}
(*RETURN* (*resolve-cls-wl′* (*x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *x1f*, *ha*, *ia*, *ja*) *x2 x1*))›

for *l m n p q ra s ha ia ja x1 x2 x1a x1b x1c x1d x1e x1f x1g x2g x1h x1i*
x1k x1l x2k x1m x1n x1o x1p x1q x1r x1s N x1t x2t CLS CLS′ S
unfolding *rr-def*
apply (*refine-vcg lhs-step-If*)
apply (*rule isasat-lookup-merge-eq2-lookup-merge-eq2*[**where**
vdom = ‹*set* (*get-vdom S*)› **and** *M* = *x1a* **and** *N* = *x1b* **and** *C* = *x1c* **and**
$\mathcal{A}$ = ‹*all-atms-st* (*x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *x1f*, *ha*, *ia*, *ja*)›, *THEN order-trans*])
subgoal by (*auto simp*: *twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′ simp*: *update-confl-tl-wl-pre′-def*)
subgoal by *auto*
subgoal by (*auto simp*: *twl-st-heur-conflict-ana-def*)
subgoal by (*auto simp*: *twl-st-heur-conflict-ana-def*)
subgoal by (*auto simp*: *twl-st-heur-conflict-ana-def*)
subgoal unfolding *Down-id-eq*
apply (*rule lookup-merge-eq2-spec*[**where** *M* = *x1a* **and** *C* = ‹*the x1c*› **and**
$\mathcal{A}$ = ‹*all-atms-st* (*x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *x1f*, *ha*, *ia*, *ja*)›, *THEN order-trans*])
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def intro*!: *literals-are-in-*$\mathcal{L}_{in}$*-mm-literals-are-in-*$\mathcal{L}_{in}$)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def twl-st-heur-conflict-ana-def counts-maximum-level-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def arena-lifting twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def arena-lifting twl-st-heur-conflict-ana-def*
dest: *in-set-takeD*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def merge-conflict-m-eq2-def conc-fun-SPEC lookup-conflict-remove1-pre-def*
atms-of-def
option-lookup-clause-rel-def lookup-clause-rel-def resolve-cls-wl′-def lookup-conflict-remove1-def
remove1-mset-union-distrib1 remove1-mset-union-distrib2 subset-mset.sup-commute[*of* - ‹*remove1-mset* - -›]
subset-mset.sup-commute[*of* - ‹*mset* (- ∝ -)›]
simp flip: *all-lits-st-alt-def*
intro!: *mset-as-position-remove3*
intro!: *ASSERT-leI*)
done
subgoal
apply (*subst* (*asm*) *arena-lifting*(*4*)[**where** *vdom* = ‹*set* (*get-vdom-aivdom* (*get-aivdom S*))› **and**

N = *x1b*, *symmetric*])
subgoal by (*auto simp*: *twl-st-heur-conflict-ana-def*)
subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def*)
apply (*rule isa-resolve-merge-conflict-gt2*[**where**
$\mathcal{A}$ = ‹*all-atms-st* (*x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *x1f*, *ha*, *ia*, *ja*)› **and** *vdom* = ‹*set* (*get-vdom-aivdom* (*get-aivdom S*))›,
THEN fref-to-Down-curry5, *of x1a x1b x2g x1c* ‹*get-count-max-lvls-heur S*› ‹*get-outlearned-heur S*›,
THEN order-trans])
subgoal unfolding *merge-conflict-m-pre-def*
by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def twl-st-heur-conflict-ana-def counts-maximum-level-def*)
subgoal by (*auto simp*: *twl-st-heur-conflict-ana-def*)
subgoal
by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def conc-fun-SPEC lookup-conflict-remove1-pre-def atms-of-def*
option-lookup-clause-rel-def lookup-clause-rel-def resolve-cls-wl′-def
merge-conflict-m-def lookup-conflict-remove1-def subset-mset.sup-commute[*of* - ‹*mset* (- ∝ -)›]
remove1-mset-union-distrib1 remove1-mset-union-distrib2
intro!: *mset-as-position-remove3*
simp flip: *all-lits-st-alt-def*
intro!: *ASSERT-leI*)
done
done
have *all-in-dom*: ‹∀ *C*∈*set* (*N* ∝ *x2*).
C ∈# $\mathcal{L}_{all}$ (*all-atms-st* (*x1a*, *N*, *x1c*, *x1d*, *x1e*, *x1f*, *ha*, *ia*, *ja*, *ka*, *la*))› **if**
‹*valid-arena x1t N vdom*›
‹*x2* ∈# *dom-m N*›
for *x2 x1t C x1a N x1c x2d x1e x1f ha ia ja ka la x1d vdom*
apply (*intro ballI*)
subgoal for *C*
using *that*(*2*)
by (*auto intro*!: *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$ *in-clause-in-all-lits-of-m*
simp: *literals-are-in-*$\mathcal{L}_{in}$*-def all-atms-st-def all-atms-def all-lits-def*
all-lits-of-mm-add-mset $\mathcal{L}_{all}$*-union ran-m-def* $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm*
dest!: *multi-member-split*[*of* - ‹*dom-m* -›])
done
have *all-in-do*: ‹*C*∈*set* [*x2*..<*x2* + *arena-length x1t x2*] ⟹
arena-lit x1t C ∈# $\mathcal{L}_{all}$ (*all-atms-st* (*x1a*, *N*, *x1c*, *x1d*, *x1e*, *x1f*, *ha*, *ia*, *ja*, *ka*, *la*))› **if**
‹*valid-arena x1t N vdom*›
‹*x2* ∈# *dom-m N*›
for *x2 x1t C x1a N x1c x2d x1e x1f ha ia ja ka la x1d vdom*
apply (*subst* (*asm*) *arena-lifting*(*4*)[*OF that*, *symmetric*])
using *that*(*2*) *arena-lifting*(*5*)[*OF that*, *of* ‹*C* − *x2*›, *symmetric*]
by (*auto intro*!: *literals-are-in-*$\mathcal{L}_{in}$*-in-*$\mathcal{L}_{all}$ *in-clause-in-all-lits-of-m*
simp: *literals-are-in-*$\mathcal{L}_{in}$*-def all-atms-st-def all-atms-def all-lits-def*
all-lits-of-mm-add-mset $\mathcal{L}_{all}$*-union ran-m-def* $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm*
dest!: *multi-member-split*[*of* - ‹*dom-m* -›])

have *isa-vmtf-mark-to-rescore-clause*: ‹
(((*x1g*, *x2g*), *S*), (*x1*, *x2*), *x1a*, *x1b*, *x1c*, *x1d*,
x1e, *x1f*, *ha*, *ia*, *ja*, *ka*, *la*)
∈ *nat-lit-lit-rel* ×$_f$ *nat-rel* ×$_f$ *twl-st-heur-conflict-ana′ r lcount* ⟹
CLS = ((*x1*, *x2*), *x1a*, *x1b*, *x1c*, *x1d*, *x1e*, *x1f*, *ha*, *ia*, *ja*, *ka*, *la*) ⟹
CLS′ = ((*x1g*, *x2g*), *S*) ⟹

```
    update-confl-tl-wl-pre x1 x2 (x1a, x1b, x1c, x1d, x1e, x1f, ha, ia, ja, ka, la) ⟹
    ((x1t, x2t), N)
    ∈ {((arena', lbd), N').
    valid-arena arena' N'
     (set (get-vdom (snd ((x1g, x2g), S)))) ∧
    x1b = N' ∧ length x1i = length arena'} ⟹
    1 ≤ x1p ⟹
    arena-is-valid-clause-idx x1t x2g ⟹
    (((x1v, x1w, x2v), x1x, x2x), D)
    ∈ {a. case a of
       (a, b) ⇒
      (case a of
       (C, a) ⇒
         case a of
         (clvls, outl) ⇒
        λD. (C, Some D) ∈ option-lookup-clause-rel (all-atms-st (x1a, N, x1c, x1d, x1e, x1f, ha, ia, ja,
ka, la)) ∧
            clvls = card-max-lvl x1a (remove1-mset x1 (mset (N ∝ x2)) ∪# the x1c) ∧
            out-learned x1a (Some (remove1-mset x1 (mset (N ∝ x2)) ∪# the x1c)) outl ∧
             size (remove1-mset x1 (mset (N ∝ x2)) ∪# the x1c) = size (mset (N ∝ x2) ∪# the x1c −
{#x1, − x1#}) + Suc 0 ∧
            D = resolve-cls-wl' (x1a, N, x1c, x1d, x1e, x1f, ha, ia, ja, ka, la) x2 x1)
       b} ⟹
    arena-act-pre x1t x2g ⟹
    isa-vmtf-bump-to-rescore-also-reasons-cl (get-trail-wl-heur S) x1t x2g (−x1g) (get-vmtf-heur S)
          ≤ ⇓ {(vm, N'). N = N' ∧ vm ∈ bump-heur (all-atms-st (x1a, N, x1c, x1d, x1e, x1f, ha, ia,
ja, ka, la)) x1a}
    (RETURN N)›
   for l m n p q ra s ha ia ja ka la x1 x2 x1a x1b x1c x1d x1e x1f x1g x2g x1h x1i x1k x1l x2k
     x1m x1n x1o x1p x1q x1r x1s N x1t x2t D x1v x1w x2v x1x x2x CLS CLS' S
  unfolding twl-st-heur-conflict-ana-def apply (clarsimp simp only: prod-rel-iff)
  subgoal
    apply (rule isa-vmtf-bump-to-rescore-also-reasons-cl-vmtf-mark-to-rescore-also-reasons-cl[
       where A = ‹all-atms-st (x1a, N, x1c, x1d, x1e, x1f, ha, ia, ja, ka, la)›,
       THEN fref-to-Down-curry4,
         of - - - - - x1a x1t x2 ‹−x1› ‹get-vmtf-heur S›,
       THEN order-trans])
    subgoal by (simp add: twl-st-heur-conflict-ana-def)
    subgoal by (auto simp add: twl-st-heur-conflict-ana-def)
    subgoal
      apply (rule ref-two-step'[THEN order-trans, OF vmtf-mark-to-rescore-also-reasons-cl-spec,
        of - - x1a -
          N ‹set (get-vdom-aivdom (get-aivdom S))›])
      subgoal by (auto simp: )
      subgoal by (auto simp: update-confl-tl-wl-pre-def update-confl-tl-l-pre-def
        twl-st-l-def state-wl-l-def)
      subgoal by (auto simp: update-confl-tl-wl-pre-def update-confl-tl-l-pre-def
        twl-st-l-def state-wl-l-def)
      subgoal by auto
      subgoal
        by (rule all-in-dom)
          (auto simp: update-confl-tl-wl-pre-def update-confl-tl-l-pre-def ran-m-def
          twl-st-l-def state-wl-l-def)
      subgoal
        by (intro conjI impI ballI allI,
            (auto simp: update-confl-tl-wl-pre-def update-confl-tl-l-pre-def ran-m-def
```

```
        twl-st-l-def state-wl-l-def twl-list-invs-def)[])
      (rule all-in-do,
       auto simp: update-confl-tl-wl-pre-def update-confl-tl-l-pre-def ran-m-def
       twl-st-l-def state-wl-l-def twl-list-invs-def)
   subgoal by (auto simp: RETURN-def conc-fun-RES)
  done
 done
done

have isa-bump-unset: ‹isa-bump-unset L x ≤ ⇓{(a, -). a ∈ bump-heur 𝒜 (tl M)} (RETURN ())›
if
  vmtf: ‹x∈ bump-heur 𝒜 M› and
  L-N: ‹L ∈ atms-of (ℒ_all 𝒜)› and [simp]: ‹M ≠ []› and
  nz: ‹count-decided M > 0› and
  L: ‹L = atm-of (lit-of (hd M))› for L x M 𝒜
  unfolding L
  by (rule isa-bump-unset-vmtf-tl[OF that(1–4)[unfolded L], THEN order-trans])
    (auto simp: conc-fun-RES RETURN-def)

have mset-tl-nth-0: ‹length Na > 0 ⟹ mset (tl (Na)) = mset Na − {#(Na ! 0)#}› for Na
  by (cases Na) auto
show ?thesis
  supply [[goals-limit = 1]]
  supply RETURN-as-SPEC-refine[refine2 del]
     update-confl-tl-wl-pre-update-confl-tl-wl-pre'[dest!]
  apply (intro frefI nres-relI)
  subgoal for CLS' CLS
    apply (cases CLS'; cases CLS; hypsubst+)
    unfolding uncurry-def update-confl-tl-wl-heur-alt-def comp-def Let-def
      update-confl-tl-wl-def mop-update-confl-tl-wl-alt-def prod.case
    apply (refine-rcg calculate-LBD-heur-st-calculate-LBD-st[where
        vdom = ‹set (get-vdom (snd CLS'))› and
        𝒜 = ‹all-atms-st (snd CLS)›]
      isa-bump-unset[where 𝒜3 = ‹all-atms-st (snd CLS)› and M3 = ‹get-trail-wl (snd CLS)›];
      remove-dummy-vars)
    subgoal
      by (auto simp: twl-st-heur-conflict-ana-def update-confl-tl-wl-pre'-def
         RES-RETURN-RES RETURN-def counts-maximum-level-def)
    subgoal by (auto dest!: update-confl-tl-wl-pre-update-confl-tl-wl-pre'
       simp: update-confl-tl-wl-pre'-def arena-is-valid-clause-idx-def twl-st-heur-conflict-ana-def)
    subgoal by (auto dest!: update-confl-tl-wl-pre-update-confl-tl-wl-pre'
       simp: update-confl-tl-wl-pre'-def arena-is-valid-clause-idx-def twl-st-heur-conflict-ana-def)
    subgoal
      using literals-are-in-ℒ_in-nth[of ‹snd (fst CLS)› ‹snd CLS›
       ‹all-atms-st (snd CLS)›, simplified]
      by (auto
       simp: update-confl-tl-wl-pre'-def arena-is-valid-clause-idx-def twl-st-heur-conflict-ana-def)
    subgoal by auto
    subgoal
      by (auto simp: twl-st-heur-conflict-ana-def update-confl-tl-wl-pre'-def
         RES-RETURN-RES RETURN-def counts-maximum-level-def)
   subgoal by (auto intro!: exI[of - ‹get-clauses-wl (snd CLS)›] exI[of - ‹set (get-vdom (snd CLS'))›]
       simp: update-confl-tl-wl-pre'-def arena-is-valid-clause-idx-def twl-st-heur-conflict-ana-def)
    apply (rule rr; assumption)
    subgoal by (simp add: arena-act-pre-def)
    apply (rule isa-vmtf-mark-to-rescore-clause; assumption)
```

subgoal by (*auto dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def arena-is-valid-clause-idx-def twl-st-heur-conflict-ana-def*
simp flip: *all-lits-st-alt-def*
intro!: *isa-bump-unset-pre*
dest: *literals-are-in-*$\mathcal{L}_{in}$*-trail-in-lits-of-l*[*of - M* ‹*lit-of* (*hd M*)› **for** *M*])
subgoal for *m n p q ra s t ha ia ja x1 x2 x1a x1b x1c x1d x1e x1f x1g x2g x1h x1i x1k x1l x2k x1m x1n x1o x1p x1q x1r x1t*
by (*rule tl-trailt-tr-pre*[*of x1 -* ‹*all-atms-st* (*x1*, *x1i*, *x1a*, *x1b*, *x1c*, *x1d*, *n*, *p*, *q*, *ra*, *s*, *t*, *ha*)›])
(*clarsimp-all dest*!: *update-confl-tl-wl-pre-update-confl-tl-wl-pre′*
simp: *update-confl-tl-wl-pre′-def arena-is-valid-clause-idx-def twl-st-heur-conflict-ana-def*
simp flip: *all-lits-st-alt-def*
intro!: *tl-trailt-tr-pre*)
subgoal by *auto*
subgoal apply (*clarsimp simp*: *twl-st-heur-conflict-ana-def update-confl-tl-wl-pre′-def*
valid-arena-mark-used subset-mset.sup-commute[*of -* ‹*remove1-mset - -*›]
tl-trail-tr[*THEN fref-to-Down-unRET*] *resolve-cls-wl′-def isa-bump-unset-vmtf-tl no-dup-tlD*
counts-maximum-level-def
simp flip: *all-lits-st-alt-def*
dest: *literals-are-in-*$\mathcal{L}_{in}$*-trail-in-lits-of-l*[*of - M* ‹*lit-of* (*hd M*)› **for** *M*])
apply (*meson all-in-dom atm-of-lit-in-atms-of*)
by (*meson atm-of-in-atms-of literals-are-in-*$\mathcal{L}_{in}$*-in-mset-*$\mathcal{L}_{all}$)+
subgoal by (*clarsimp simp*: *twl-st-heur-conflict-ana-def update-confl-tl-wl-pre′-def*
valid-arena-mark-used subset-mset.sup-commute[*of -* ‹*remove1-mset - -*›]
tl-trail-tr[*THEN fref-to-Down-unRET*] *resolve-cls-wl′-def isa-bump-unset-vmtf-tl no-dup-tlD*
counts-maximum-level-def
simp flip: *all-lits-st-alt-def*
dest: *literals-are-in-*$\mathcal{L}_{in}$*-trail-in-lits-of-l*[*of - M* ‹*lit-of* (*hd M*)› **for** *M*])
subgoal by (*clarsimp simp*: *twl-st-heur-conflict-ana-def update-confl-tl-wl-pre′-def*
valid-arena-mark-used subset-mset.sup-commute[*of -* ‹*remove1-mset - -*›]
tl-trail-tr[*THEN fref-to-Down-unRET*] *resolve-cls-wl′-def isa-bump-unset-vmtf-tl no-dup-tlD*
counts-maximum-level-def
simp flip: *all-lits-st-alt-def*
dest: *literals-are-in-*$\mathcal{L}_{in}$*-trail-in-lits-of-l*[*of - M* ‹*lit-of* (*hd M*)› **for** *M*])
subgoal by (*clarsimp simp*: *twl-st-heur-conflict-ana-def update-confl-tl-wl-pre′-def*
valid-arena-mark-used subset-mset.sup-commute[*of -* ‹*remove1-mset - -*›]
tl-trail-tr[*THEN fref-to-Down-unRET*] *resolve-cls-wl′-def isa-bump-unset-vmtf-tl no-dup-tlD*
counts-maximum-level-def
simp flip: *all-lits-st-alt-def*
dest: *literals-are-in-*$\mathcal{L}_{in}$*-trail-in-lits-of-l*[*of - M* ‹*lit-of* (*hd M*)› **for** *M*])
subgoal by (*clarsimp simp*: *twl-st-heur-conflict-ana-def update-confl-tl-wl-pre′-def*
valid-arena-mark-used subset-mset.sup-commute[*of -* ‹*remove1-mset - -*›]
tl-trail-tr[*THEN fref-to-Down-unRET*] *resolve-cls-wl′-def isa-bump-unset-vmtf-tl no-dup-tlD*
counts-maximum-level-def
simp flip: *all-lits-st-alt-def*
dest: *literals-are-in-*$\mathcal{L}_{in}$*-trail-in-lits-of-l*[*of - M* ‹*lit-of* (*hd M*)› **for** *M*])
done
done
qed

lemma *phase-saving-le*: ‹*phase-saving* $\mathcal{A}$ $\varphi \Longrightarrow A \in\# \mathcal{A} \Longrightarrow A <$ *length* φ›
‹*phase-saving* $\mathcal{A}$ $\varphi \Longrightarrow B \in\# \mathcal{L}_{all}\ \mathcal{A} \Longrightarrow$ *atm-of* $B <$ *length* φ›
by (*auto simp*: *phase-saving-def atms-of-*$\mathcal{L}_{all}$*-*$\mathcal{A}_{in}$)

lemma *trail-pol-nempty*: ‹¬(([], *aa*, *ab*, *ac*, *ad*, *b*), *L* # *ys*) ∈ *trail-pol* $\mathcal{A}$›
by (*auto simp*: *trail-pol-def ann-lits-split-reasons-def*)

```
lemma is-decided-hd-trail-wl-heur-hd-get-trail:
  ‹(RETURN o is-decided-hd-trail-wl-heur, RETURN o (λM. is-decided (hd (get-trail-wl M))))
   ∈ [λM. get-trail-wl M ≠ []]_f twl-st-heur-conflict-ana' r lcount → ⟨bool-rel⟩ nres-rel›
   by (intro frefI nres-relI)
     (auto simp: is-decided-hd-trail-wl-heur-def twl-st-heur-conflict-ana-def neq-Nil-conv
        trail-pol-def ann-lits-split-reasons-def is-decided-no-proped-iff last-trail-pol-def
      split: option.splits)


lemma skip-and-resolve-loop-wl-D-heur-alt-def:
  ‹skip-and-resolve-loop-wl-D-heur S_0 =
    do {
      (-, S) ←
        WHILE_T^skip-and-resolve-loop-wl-D-heur-inv S_0
        (λ(brk, S). ¬brk ∧ ¬is-decided-hd-trail-wl-heur S)
        (λ(brk, S).
          do {
            ASSERT(¬brk ∧ ¬is-decided-hd-trail-wl-heur S);
            (L, C) ← lit-and-ann-of-propagated-st-heur S;
            b ← atm-is-in-conflict-st-heur (−L) S;
            if b then
      tl-state-wl-heur S
            else do {
              b ← maximum-level-removed-eq-count-dec-heur L S;
              if b
              then do {
                update-confl-tl-wl-heur L C S
              }
              else
                RETURN (True, S)
            }
          }
        )
        (False, S_0);
      RETURN S
    }
  ›
  unfolding IsaSAT-Profile.start-def IsaSAT-Profile.stop-def nres-monad1
    skip-and-resolve-loop-wl-D-heur-def ..


lemma atm-is-in-conflict-st-heur-is-in-conflict-st:
  ‹(uncurry (atm-is-in-conflict-st-heur), uncurry (mop-lit-notin-conflict-wl)) ∈
   [λ(L, S). True]_f
   Id ×_r twl-st-heur-conflict-ana → ⟨Id⟩ nres-rel›
proof −
  have 1: ‹aaa ∈# L_all A ⟹ atm-of aaa  ∈ atms-of (L_all A)› for aaa A
    by (auto simp: atms-of-def)
  show ?thesis
 unfolding atm-is-in-conflict-st-heur-def twl-st-heur-def option-lookup-clause-rel-def uncurry-def comp-def
    mop-lit-notin-conflict-wl-def twl-st-heur-conflict-ana-def
  apply (intro frefI nres-relI)
  apply refine-rcg
  apply clarsimp
  subgoal
     apply (rule atm-in-conflict-lookup-pre[of - ‹all-atms-st -›])
```

```
    unfolding \mathcal{L}_{all}-all-atms-all-lits[symmetric] all-lits-st-alt-def[symmetric]
    apply auto
    done
  subgoal for x y x1 x2 x1a x2a x1b x2b
   apply (subst atm-in-conflict-lookup-atm-in-conflict[THEN fref-to-Down-unRET-uncurry-Id, of ‹all-atms-st
x2› ‹atm-of x1› ‹the (get-conflict-wl (snd y))›])
    apply (simp add: \mathcal{L}_{all}-all-atms-all-lits atms-of-def all-lits-st-alt-def[symmetric])[]
    apply (auto simp add: \mathcal{L}_{all}-all-atms-all-lits atms-of-def option-lookup-clause-rel-def)[]
    apply (simp add: atm-in-conflict-def atm-of-in-atms-of)
    done
  done
qed

lemma skip-and-resolve-loop-wl-alt-def:
  ‹skip-and-resolve-loop-wl S_0 =
    do {
      ASSERT(get-conflict-wl S_0 ≠ None);
      (-, S) ←
        WHILE_T^λ(brk, S). skip-and-resolve-loop-wl-inv S_0 brk S
        (λ(brk, S). ¬brk ∧ ¬is-decided (hd (get-trail-wl S)))
        (λ(-, S).
          do {
            (L, C) ← mop-hd-trail-wl S;
            b ← mop-lit-notin-conflict-wl (−L) S;
            if b then
              mop-tl-state-wl S
            else do {
              b ← mop-maximum-level-removed-wl L S;
              if b
              then do {
                mop-update-confl-tl-wl L C S
              }
              else
                do {RETURN (True, S)}
            }
          }
        )
        (False, S_0);
      RETURN S
    }›
  unfolding skip-and-resolve-loop-wl-def calculate-LBD-st-def IsaSAT-Profile.start-def
    IsaSAT-Profile.stop-def
  by (auto cong: if-cong)

lemma skip-and-resolve-loop-wl-D-heur-skip-and-resolve-loop-wl-D:
  ‹(skip-and-resolve-loop-wl-D-heur, skip-and-resolve-loop-wl)
    ∈ twl-st-heur-conflict-ana' r lcount →_f ⟨twl-st-heur-conflict-ana' r lcount⟩nres-rel›
proof −
  have H[refine0]: ‹(x, y) ∈ twl-st-heur-conflict-ana' r lcount ⟹
          ((False, x), False, y)
          ∈ bool-rel ×_f
            twl-st-heur-conflict-ana' r lcount› for x y
    by auto
  have H1: ‹(xa, x')
    ∈ bool-rel ×_f
      twl-st-heur-conflict-ana' r lcount ⟹
```

```
    case x' of (x, xa) ⇒ skip-and-resolve-loop-wl-inv y x xa ⟹
    xa = (x1, x2) ⟹
      x' = (x1a, x2a) ⟹ (x2, x2a) ∈ twl-st-heur-conflict-ana' r lcount› for xa x' x1 x2 x1a x2a y
   by auto
  show ?thesis
    supply [[goals-limit=1]]
    unfolding skip-and-resolve-loop-wl-D-heur-alt-def skip-and-resolve-loop-wl-alt-def
    apply (intro frefI nres-relI)
    apply (refine-vcg
        update-confl-tl-wl-heur-update-confl-tl-wl[THEN fref-to-Down-curry2, unfolded comp-def]
        maximum-level-removed-eq-count-dec-heur-maximum-level-removed-eq-count-dec
        [THEN fref-to-Down-curry] lit-and-ann-of-propagated-st-heur-lit-and-ann-of-propagated-st[THEN
fref-to-Down]
        tl-state-wl-heur-tl-state-wl[THEN fref-to-Down]
        atm-is-in-conflict-st-heur-is-in-conflict-st[THEN fref-to-Down-curry])
   subgoal for S T brkS brkT
     unfolding skip-and-resolve-loop-wl-D-heur-inv-def
     apply (subst case-prod-beta)
     apply (rule exI[of - ‹snd brkT›])
     apply (rule exI[of - ‹T›])
     apply (subst (asm) (1) surjective-pairing[of brkS])
     apply (subst (asm) surjective-pairing[of brkT])
     unfolding prod-rel-iff
     by auto
   subgoal for x y xa x' x1 x2 x1a x2a
     apply (subst is-decided-hd-trail-wl-heur-hd-get-trail[of r, THEN fref-to-Down-unRET-Id, of x2a])
     subgoal
     unfolding skip-and-resolve-loop-wl-inv-def skip-and-resolve-loop-inv-l-def skip-and-resolve-loop-inv-def
       apply (subst (asm) case-prod-beta)+
       unfolding prod.case
       apply normalize-goal+
       by (auto simp: )
    apply (rule H1; assumption)
    subgoal by auto
    done
   subgoal by auto
   subgoal by auto
   subgoal by auto
   subgoal by auto
   subgoal by auto
   subgoal by auto
   subgoal by auto
   subgoal by auto
   subgoal by auto
   subgoal by auto
   subgoal by auto
   subgoal by auto
   done
qed

lemmas get-learned-count-learned-clss-countD =
  get-learned-count-learned-clss-countD2

end
theory IsaSAT-Conflict-Analysis-LLVM
imports IsaSAT-Conflict-Analysis-Defs IsaSAT-VMTF-LLVM IsaSAT-Setup-LLVM IsaSAT-LBD-LLVM
begin
```

sepref-def *maximum-level-removed-eq-count-dec-fast-code*
is ‹*uncurry* (*maximum-level-removed-eq-count-dec-heur*)›
:: ‹*unat-lit-assn*^{k} $*_a$ *isasat-bounded-assn*^{k} $\rightarrow_a$ *bool1-assn*›
unfolding *maximum-level-removed-eq-count-dec-heur-def*
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

definition *is-decided-trail* **where** ‹*is-decided-trail* = (λ(*M*, *xs*, *lvls*, *reasons*, *k*).
let r = *reasons* ! (*atm-of* (*last M*)) *in*
RETURN (*r* = *DECISION-REASON*))›

sepref-def *is-decided-trail-impl*
is ‹*is-decided-trail*›
:: ‹[(λ*S*. *fst S* ≠ [] ∧ *last-trail-pol-pre S*)]$_a$ *trail-pol-fast-assn*^{k} → *bool1-assn*›
unfolding *is-decided-trail-def trail-pol-fast-assn-def last-trail-pol-pre-def*
by *sepref*

definition *is-decided-hd-trail-wl-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*is-decided-hd-trail-wl-fast-code* = *read-trail-wl-heur-code is-decided-trail-impl*›
global-interpretation *is-decided-hd*: *read-trail-param-adder0* **where**
f = ‹*is-decided-trail-impl*› **and**
f' = ‹*is-decided-trail*› **and**
x-assn = *bool1-assn* **and**
P = ‹(λ*S*. *fst S* ≠ [] ∧ *last-trail-pol-pre S*)›
rewrites ‹*read-trail-wl-heur is-decided-trail* = *RETURN o is-decided-hd-trail-wl-heur*› **and**
‹*read-trail-wl-heur-code is-decided-trail-impl* = *is-decided-hd-trail-wl-fast-code*› **and**
‹*case-isasat-int* (λ*M* - - - - - - - - - - - - - - - - -. *fst M* ≠ [] ∧ *last-trail-pol-pre M*) = *is-decided-hd-trail-wl-heur-pre*›
apply *unfold-locales*
apply (*rule is-decided-trail-impl.refine*)
subgoal
by (*auto simp*: *read-all-st-def is-decided-hd-trail-wl-heur-def is-decided-trail-def last-trail-pol-def Let-def*
intro!: *ext*
split: *isasat-int-splits*)
subgoal
by (*auto simp*: *is-decided-hd-trail-wl-fast-code-def*)
subgoal by (*auto simp*: *is-decided-hd-trail-wl-heur-pre-def intro*!: *ext split*: *isasat-int-splits*)
done

definition *lit-and-ann-of-propagated-trail-heur*
:: ‹- ⇒ (*nat literal* × *nat*) *nres*›
where
‹*lit-and-ann-of-propagated-trail-heur* = (λ(*M*, -, -, *reasons*, -) . *do* {
ASSERT(*M* ≠ [] ∧ *atm-of* (*last M*) < *length reasons*);
RETURN (*last M*, *reasons* ! (*atm-of* (*last M*)))})›

sepref-def *lit-and-ann-of-propagated-trail-heur-impl*
is *lit-and-ann-of-propagated-trail-heur*
:: ‹*trail-pol-fast-assn*^{k} $\rightarrow_a$ (*unat-lit-assn* $\times_a$ *sint64-nat-assn*)›
unfolding *lit-and-ann-of-propagated-trail-heur-def trail-pol-fast-assn-def*
by *sepref*

definition *lit-and-ann-of-propagated-st-heur-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*lit-and-ann-of-propagated-st-heur-fast-code* = *read-trail-wl-heur-code lit-and-ann-of-propagated-trail-heur-impl*›

global-interpretation *lit-and-of-proped-lit*: *read-trail-param-adder0* **where**
f = ‹*lit-and-ann-of-propagated-trail-heur-impl*› **and**
f' = ‹*lit-and-ann-of-propagated-trail-heur*› **and**
x-assn = ‹*unat-lit-assn* ×$_a$ *sint64-nat-assn*› **and**
P = ‹(λ*S*. *True*)›
rewrites ‹*read-trail-wl-heur lit-and-ann-of-propagated-trail-heur* = *lit-and-ann-of-propagated-st-heur*›
and
‹*read-trail-wl-heur-code lit-and-ann-of-propagated-trail-heur-impl* = *lit-and-ann-of-propagated-st-heur-fast-code*›
apply *unfold-locales*
apply (*rule lit-and-ann-of-propagated-trail-heur-impl.refine*)
subgoal
by (*auto simp*: *read-all-st-def lit-and-ann-of-propagated-st-heur-def lit-and-ann-of-propagated-trail-heur-def last-trail-pol-def Let-def*
intro!: *ext*
split: *isasat-int-splits*)
subgoal
by (*auto simp*: *lit-and-ann-of-propagated-st-heur-fast-code-def*)
done

definition *atm-is-in-conflict-confl-heur* :: ‹- ⇒ *nat literal* ⇒*bool nres*› **where**
‹*atm-is-in-conflict-confl-heur* = (λ(-, *D*) *L*. *do* {
ASSERT (*atm-in-conflict-lookup-pre* (*atm-of L*) *D*); *RETURN* (¬*atm-in-conflict-lookup* (*atm-of L*) *D*) })›

sepref-def *atm-is-in-conflict-confl-heur-impl*
is ‹*uncurry atm-is-in-conflict-confl-heur*›
:: ‹*conflict-option-rel-assn*^{k} ×$_a$ *unat-lit-assn*^{k} →$_a$ *bool1-assn*›
unfolding *atm-is-in-conflict-confl-heur-def conflict-option-rel-assn-def*
by *sepref*

definition *atm-is-in-conflict-st-heur-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*atm-is-in-conflict-st-heur-fast-code* = (λ*N C*. *read-conflict-wl-heur-code* (λ*M*. *atm-is-in-conflict-confl-heur-impl M C*) *N*)›

definition *atm-is-in-conflict-st-heur'* :: ‹*isasat* ⇒ *nat literal* ⇒ *bool nres*› **where**
‹*atm-is-in-conflict-st-heur' S L* = (λ(-, *D*). *do* {
ASSERT (*atm-in-conflict-lookup-pre* (*atm-of L*) *D*); *RETURN* (¬*atm-in-conflict-lookup* (*atm-of L*) *D*) }) (*get-conflict-wl-heur S*)›

global-interpretation *atm-in-conflict*: *read-conflict-param-adder* **where**
f = ‹ λ*S L*. *atm-is-in-conflict-confl-heur-impl L S*› **and**
f' = ‹λ*S L*. *atm-is-in-conflict-confl-heur L S*› **and**
x-assn = ‹*bool1-assn*› **and**
P = ‹(λ- -. *True*)› **and**
R = ‹*unat-lit-rel*›
rewrites ‹(λ*N C*. *read-conflict-wl-heur* (λ*M*. *atm-is-in-conflict-confl-heur M C*) *N*) = *atm-is-in-conflict-st-heur'*›
and
‹(λ*N C*. *read-conflict-wl-heur-code* (λ*M*. *atm-is-in-conflict-confl-heur-impl M C*) *N*) = *atm-is-in-conflict-st-heur-fast-coo*
apply *unfold-locales*
apply (*subst lambda-comp-true*,
rule atm-is-in-conflict-confl-heur-impl.refine)
subgoal
by (*auto simp*: *read-all-st-def atm-is-in-conflict-st-heur'-def atm-is-in-conflict-confl-heur-def Let-def*
intro!: *ext*

split: *isasat-int-splits*)
subgoal
by (*auto simp*: *atm-is-in-conflict-st-heur-fast-code-def*)
done

lemmas [*unfolded lambda-comp-true*, *sepref-fr-rules*] = *is-decided-hd.refine lit-and-of-proped-lit.refine atm-in-conflict.refine*
lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
is-decided-hd-trail-wl-fast-code-def[*unfolded read-all-st-code-def*]
lit-and-ann-of-propagated-st-heur-fast-code-def[*unfolded read-all-st-code-def*]
atm-is-in-conflict-st-heur-fast-code-def[*unfolded read-all-st-code-def*]

sepref-def *atm-is-in-conflict-st-heur-fast2-code*
is ‹*uncurry* (*atm-is-in-conflict-st-heur*)›
:: ‹[λ-. *True*]$_a$ *unat-lit-assn*^{k} *$*_a$ *isasat-bounded-assn*^{k} → *bool1-assn*›
supply [[*goals-limit=1*]]
unfolding *atm-is-in-conflict-st-heur-def atm-is-in-conflict-st-heur'-def*[*symmetric*]
by *sepref*

lemma *tl-state-wl-heurI*: ‹*tl-state-wl-heur-pre S* ⟹ *fst* (*get-trail-wl-heur S*) ≠ []›
‹*tl-state-wl-heur-pre S* ⟹ *tl-trailt-tr-pre* (*get-trail-wl-heur S*)›
by (*auto simp*: *tl-state-wl-heur-pre-def tl-trailt-tr-pre-def Let-def isa-bump-unset-pre-def vmtf-unset-pre-def lit-of-last-trail-pol-def*)

lemma *tl-state-wl-heur-alt-def*:
‹*tl-state-wl-heur* = (λS_0. *do* {
ASSERT(*tl-state-wl-heur-pre* S_0);
let (*M*, *S*) = *extract-trail-wl-heur* S_0; *let* (*vm*, *S*) = *extract-vmtf-wl-heur S*;
ASSERT (*M* = *get-trail-wl-heur* S_0);
ASSERT (*vm* = *get-vmtf-heur* S_0);
let L = *lit-of-last-trail-pol M*;
let S = *update-trail-wl-heur* (*tl-trailt-tr M*) *S*;
ASSERT (*isa-bump-unset-pre* (*atm-of L*) *vm*);
vm ← *isa-bump-unset* (*atm-of L*) *vm*;
let S = *update-vmtf-wl-heur vm S*;
RETURN (*False*, *S*)
})›
by (*auto simp*: *tl-state-wl-heur-def state-extractors Let-def intro*!: *ext split*: *isasat-int-splits*)

sepref-register *isa-bump-unset*
sepref-def *tl-state-wl-heur-fast-code*
is ‹*tl-state-wl-heur*›
:: ‹[λ-. *True*]$_a$ *isasat-bounded-assn*^{d} → *bool1-assn* ×$_a$ *isasat-bounded-assn*›
supply [[*goals-limit=1*]] *if-splits*[*split*] *tl-state-wl-heurI*[*dest*]
unfolding *vmtf-unset-def bind-ref-tag-def short-circuit-conv tl-state-wl-heur-alt-def*
by *sepref*

definition *extract-values-of-lookup-conflict* :: ‹*conflict-option-rel* ⇒ *bool*› **where**
‹*extract-values-of-lookup-conflict* = (λ(*b*, (-, *xs*)). *b*)›

sepref-def *extract-values-of-lookup-conflict-impl*
is ‹*RETURN o extract-values-of-lookup-conflict*›
:: ‹*conflict-option-rel-assn*^{k} →$_a$ *bool1-assn*›
unfolding *extract-values-of-lookup-conflict-def conflict-option-rel-assn-def*

lookup-clause-rel-assn-def
by *sepref*

sepref-register *extract-values-of-lookup-conflict*
declare *extract-values-of-lookup-conflict-impl.refine*[*sepref-fr-rules*]

sepref-register *isasat-lookup-merge-eq2 update-confl-tl-wl-heur*

lemma *update-confl-tl-wl-heur-alt-def*:
‹*update-confl-tl-wl-heur* = (λ*L C* S_0. *do* {
let (*M*, *S*) = *extract-trail-wl-heur* S_0;
let (*N*, *S*) = *extract-arena-wl-heur S*;
let (*lbd*, *S*) = *extract-lbd-wl-heur S*;
let (*outl*, *S*) = *extract-outl-wl-heur S*;
let (*clvls*, *S*) = *extract-clvls-wl-heur S*;
let (*vm*, *S*) = *extract-vmtf-wl-heur S*;
let (*bnxs*, *S*) = *extract-conflict-wl-heur S*;
(*N*, *lbd*) ← *calculate-LBD-heur-st M N lbd C*;
ASSERT (*clvls* ≥ *1*);
let L' = *atm-of L*;
ASSERT(*arena-is-valid-clause-idx N C*);
(*bnxs*, *clvls*, *outl*) ←
if arena-length N C = *2 then isasat-lookup-merge-eq2 L M N C bnxs clvls outl*
else isa-resolve-merge-conflict-gt2 M N C bnxs clvls outl;
let b = *extract-values-of-lookup-conflict bnxs*;
let nxs = *the-lookup-conflict bnxs*;
ASSERT(*curry lookup-conflict-remove1-pre L* (*nxs*) ∧ *clvls* ≥ *1*);
let (*nxs*) = *lookup-conflict-remove1 L* (*nxs*);
ASSERT(*arena-act-pre N C*);
vm ← *isa-vmtf-bump-to-rescore-also-reasons-cl M N C* (−*L*) *vm*;
ASSERT(*isa-bump-unset-pre L' vm*);
ASSERT(*tl-trailt-tr-pre M*);
vm ← *isa-bump-unset L' vm*;
let S = *update-trail-wl-heur* (*tl-trailt-tr M*) *S*;
let S = *update-conflict-wl-heur* (*None-lookup-conflict b nxs*) *S*;
let S = *update-vmtf-wl-heur vm S*;
let S = *update-clvls-wl-heur* (*clvls* − *1*) *S*;
let S = *update-outl-wl-heur outl S*;
let S = *update-arena-wl-heur N S*;
let S = *update-lbd-wl-heur lbd S*;
RETURN (*False*, *S*)
})›
unfolding *update-confl-tl-wl-heur-def*
by (*auto intro*!: *ext bind-cong simp*: *None-lookup-conflict-def the-lookup-conflict-def*
extract-values-of-lookup-conflict-def Let-def state-extractors split: *isasat-int-splits*)

sepref-def *update-confl-tl-wl-fast-code*
is ‹*uncurry2 update-confl-tl-wl-heur*›
:: ‹[λ((*i*, *L*), *S*). *isasat-fast S*]$_a$
unat-lit-assn^k *$*_a$ *sint64-nat-assn*^{k} *$*_a$*isasat-bounded-assn*^{d} → *bool1-assn* ×$_a$ *isasat-bounded-assn*›
supply [[*goals-limit=1*]] *isasat-fast-length-leD*[*intro*]
unfolding *update-confl-tl-wl-heur-alt-def*
PR-CONST-def
apply (*rewrite at* ‹*If* (- = ¤)› *snat-const-fold*[**where** *'a=64*])
apply (*annot-unat-const* ‹*TYPE* (*32*)›)
by *sepref*

sepref-register *is-in-conflict-st atm-is-in-conflict-st-heur*
sepref-def *skip-and-resolve-loop-wl-D-fast*
  is ‹*skip-and-resolve-loop-wl-D-heur*›
  :: ‹[λ*S*. *isasat-fast S*]$_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
  supply [[*goals-limit=1*]]
    skip-and-resolve-loop-wl-DI[*intro*]
    isasat-fast-after-skip-and-resolve-loop-wl-D-heur-inv[*intro*]
  unfolding *skip-and-resolve-loop-wl-D-heur-def*
  apply (*rewrite at* ‹¬- ∧ ¬ -› *short-circuit-conv*)
  by *sepref*

experiment
begin
  export-llvm
    get-count-max-lvls-heur-impl
    maximum-level-removed-eq-count-dec-fast-code
    is-decided-hd-trail-wl-fast-code
    lit-and-ann-of-propagated-st-heur-fast-code
    is-in-option-lookup-conflict-code
    atm-is-in-conflict-st-heur-fast-code
    lit-of-last-trail-fast-code
    tl-state-wl-heur-fast-code
    None-lookup-conflict-impl
    extract-values-of-lookup-conflict-impl
    update-confl-tl-wl-fast-code
    skip-and-resolve-loop-wl-D-fast

end

end
theory *IsaSAT-Propagate-Conflict-Defs*
  imports *IsaSAT-Setup IsaSAT-Inner-Propagation-Defs*
begin

end
theory *IsaSAT-Propagate-Conflict*
  imports *IsaSAT-Setup IsaSAT-Inner-Propagation IsaSAT-Propagate-Conflict-Defs*
begin

Chapter 18

Propagation Loop And Conflict

18.1 Unit Propagation, Inner Loop

definition (**in** −) *length-ll-fs* :: ‹*nat twl-st-wl* ⇒ *nat literal* ⇒ *nat*› **where**
‹*length-ll-fs* = (λ(-, -, -, -, -, -, -, -, -, -, -, -, *W*) *L*. *length* (*W L*))›

definition (**in** −) *length-ll-fs-heur* :: ‹*isasat* ⇒ *nat literal* ⇒ *nat*› **where**
‹*length-ll-fs-heur S L* = *length* (*watched-by-int S L*)›

lemma *unit-propagation-inner-loop-wl-loop-D-heur-fast*:
‹*length* (*get-clauses-wl-heur b*) ≤ *unat64-max* ⟹
unit-propagation-inner-loop-wl-loop-D-heur-inv b a (*a1'*, *a1'a*, *a2'a*) ⟹
length (*get-clauses-wl-heur a2'a*) ≤ *unat64-max*›
unfolding *unit-propagation-inner-loop-wl-loop-D-heur-inv-def*
by *auto*

lemma *unit-propagation-inner-loop-wl-loop-D-heur-alt-def*:
‹*unit-propagation-inner-loop-wl-loop-D-heur L* S_0 = *do* {
ASSERT (*length* (*watched-by-int* S_0 *L*) ≤ *length* (*get-clauses-wl-heur* S_0));
n ← *mop-length-watched-by-int* S_0 *L*;
let b = (*0*, *0*, S_0);
$WHILE_T$*unit-propagation-inner-loop-wl-loop-D-heur-inv* S_0 *L*
(λ(*j*, *w*, *S*). *w* < *n* ∧ *get-conflict-wl-is-None-heur S*)
(λ(*j*, *w*, *S*). *do* {
unit-propagation-inner-loop-body-wl-heur L j w S
})
b
}›
unfolding *unit-propagation-inner-loop-wl-loop-D-heur-def Let-def*
IsaSAT-Profile.start-def IsaSAT-Profile.stop-def **..**

18.2 Unit propagation, Outer Loop

lemma *select-and-remove-from-literals-to-update-wl-heur-alt-def*:
‹*select-and-remove-from-literals-to-update-wl-heur* =
(λ*S*. *do* {
let j = *literals-to-update-wl-heur S*;
ASSERT(*j* < *length* (*fst* (*get-trail-wl-heur S*)));
ASSERT(*j* + *1* ≤ *unat32-max*);
L ← *isa-trail-nth* (*get-trail-wl-heur S*) *j*;

```
    RETURN (set-literals-to-update-wl-heur (j+1) S, −L)
   })
  ›
 unfolding select-and-remove-from-literals-to-update-wl-heur-def
 apply (intro ext)
 apply (rename-tac S; case-tac S)
 by (auto intro!: ext simp: rev-trail-nth-def Let-def)

lemma unit-propagation-outer-loop-wl-D-heur-fast:
 ‹length (get-clauses-wl-heur x) ≤ unat64-max ⟹
     unit-propagation-outer-loop-wl-D-heur-inv x s' ⟹
     length (get-clauses-wl-heur a1') =
     length (get-clauses-wl-heur s') ⟹
     length (get-clauses-wl-heur s') ≤ unat64-max›
 by (auto simp: unit-propagation-outer-loop-wl-D-heur-inv-def)

theorem unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D:
 ‹(unit-propagation-outer-loop-wl-D-heur, unit-propagation-outer-loop-wl) ∈
   {(S, T). (S, T) ∈ twl-st-heur ∧ get-learned-count S = lcount} →f
     ⟨{(S, T). (S, T) ∈ twl-st-heur ∧ get-learned-count S = lcount}⟩ nres-rel›
 using twl-st-heur''D-twl-st-heurD[OF
   unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D']
 .

end
theory IsaSAT-Propagate-Conflict-LLVM
 imports IsaSAT-Propagate-Conflict-Defs IsaSAT-Inner-Propagation-LLVM
begin



lemma unit-propagation-inner-loop-wl-loop-D-heur-fast:
 ‹length (get-clauses-wl-heur b) ≤ snat64-max ⟹
   unit-propagation-inner-loop-wl-loop-D-heur-inv b a (a1', a1'a, a2'a) ⟹
    length (get-clauses-wl-heur a2'a) ≤ snat64-max›
 unfolding unit-propagation-inner-loop-wl-loop-D-heur-inv-def
 by auto

sepref-def unit-propagation-inner-loop-wl-loop-D-fast
 is ‹uncurry unit-propagation-inner-loop-wl-loop-D-heur›
 :: ‹[λ(L, S). length (get-clauses-wl-heur S) ≤ snat64-max]a
   unat-lit-assn^k *a isasat-bounded-assn^d → sint64-nat-assn ×a sint64-nat-assn ×a isasat-bounded-assn›
 unfolding unit-propagation-inner-loop-wl-loop-D-heur-def PR-CONST-def
 unfolding watched-by-nth-watched-app watched-app-def[symmetric]
   length-ll-fs-heur-def[symmetric]
 unfolding delete-index-and-swap-update-def[symmetric] append-update-def[symmetric]
   is-None-def[symmetric] get-conflict-wl-is-None-heur-alt-def[symmetric]
 unfolding fold-tuple-optimizations
 supply [[goals-limit=1]] unit-propagation-inner-loop-wl-loop-D-heur-fast[intro] length-ll-fs-heur-def[simp]
 apply (annot-snat-const ‹TYPE (64)›)
 by sepref

lemma le-unat64-max-minus-4-unat64-max: ‹a ≤ snat64-max − MIN-HEADER-SIZE ⟹ Suc a <
max-snat 64›
 by (auto simp: snat64-max-def max-snat-def)
```

definition *cut-watch-list-heur2-inv* **where**
‹*cut-watch-list-heur2-inv L n* = (λ(*j*, *w*, *W*). *j* ≤ *w* ∧ *w* ≤ *n* ∧ *nat-of-lit L* < *length W*)›

lemma *cut-watch-list-heur2-alt-def*:
‹*cut-watch-list-heur2* = (λ*j w L S*$_0$. *do* {
let (*W*, *S*) = *extract-watchlist-wl-heur S*$_0$;
ASSERT (*W* = *get-watched-wl-heur S*$_0$);
ASSERT(*j* ≤ *length* (*W* ! *nat-of-lit L*) ∧ *j* ≤ *w* ∧ *nat-of-lit L* < *length W* ∧
w ≤ *length* (*W* ! (*nat-of-lit L*)));
let n = *length* (*W*!(*nat-of-lit L*));
(*j*, *w*, *W*) ← *WHILE*$_T$$^{cut\text{-}watch\text{-}list\text{-}heur2\text{-}inv\ L\ n}$
(λ(*j*, *w*, *W*). *w* < *n*)
(λ(*j*, *w*, *W*). *do* {
ASSERT(*w* < *length* (*W*!(*nat-of-lit L*)));
RETURN (*j*+*1*, *w*+*1*, *W*[*nat-of-lit L* := (*W*!(*nat-of-lit L*))[*j* := *W*!(*nat-of-lit L*)!*w*]])
})
(*j*, *w*, *W*);
ASSERT(*j* ≤ *length* (*W* ! *nat-of-lit L*) ∧ *nat-of-lit L* < *length W*);
let W = *W*[*nat-of-lit L* := *take j* (*W* ! *nat-of-lit L*)];
RETURN (*update-watchlist-wl-heur W S*)
})›
unfolding *cut-watch-list-heur2-inv-def cut-watch-list-heur2-def*
by (*auto simp*: *state-extractors Let-def intro*!: *ext split*: *isasat-int-splits*)

lemma *cut-watch-list-heur2I*:
‹*length* (*a1'd* ! *nat-of-lit baa*) ≤ *snat64-max* − *MIN-HEADER-SIZE* ⟹
cut-watch-list-heur2-inv baa (*length* (*a1'd* ! *nat-of-lit baa*))
(*a1'e*, *a1'f*, *a2'f*) ⟹
a1'f < *length-ll a2'f* (*nat-of-lit baa*) ⟹
ez ≤ *bba* ⟹
Suc a1'e < *max-snat 64*›
‹*length* (*a1'd* ! *nat-of-lit baa*) ≤ *snat64-max* − *MIN-HEADER-SIZE* ⟹
cut-watch-list-heur2-inv baa (*length* (*a1'd* ! *nat-of-lit baa*))
(*a1'e*, *a1'f*, *a2'f*) ⟹
a1'f < *length-ll a2'f* (*nat-of-lit baa*) ⟹
ez ≤ *bba* ⟹
Suc a1'f < *max-snat 64*›
‹*cut-watch-list-heur2-inv baa* (*length* (*a1'd* ! *nat-of-lit baa*))
(*a1'e*, *a1'f*, *a2'f*) ⟹ *nat-of-lit baa* < *length a2'f*›
‹*cut-watch-list-heur2-inv baa* (*length* (*a1'd* ! *nat-of-lit baa*))
(*a1'e*, *a1'f*, *a2'f*) ⟹ *a1'f* < *length-ll a2'f* (*nat-of-lit baa*) ⟹
a1'e < *length* (*a2'f* ! *nat-of-lit baa*)›
by (*auto simp*: *max-snat-def snat64-max-def cut-watch-list-heur2-inv-def length-ll-def*)

sepref-def *cut-watch-list-heur2-fast-code*
is ‹*uncurry3 cut-watch-list-heur2*›
:: ‹[λ(((*j*, *w*), *L*), *S*). *length* (*watched-by-int S L*) ≤ *snat64-max*−*MIN-HEADER-SIZE*]$_a$
sint64-nat-assn^k *$*_a$ *sint64-nat-assn*^{k} *$*_a$ *unat-lit-assn*^{k} *$*_a$
isasat-bounded-assn^d → *isasat-bounded-assn*›
supply [[*goals-limit=1*]] *cut-watch-list-heur2I*[*intro*] *length-ll-def*[*simp*]
unfolding *cut-watch-list-heur2-alt-def length-ll-def*[*symmetric*]
nth-rll-def[*symmetric*] *watched-by-alt-def*
op-list-list-take-alt-def[*symmetric*]
op-list-list-upd-alt-def[*symmetric*]
apply (*annot-snat-const* ‹*TYPE* (*64*)›)

by *sepref*

sepref-def *unit-propagation-inner-loop-wl-D-fast-code*
is ‹*uncurry unit-propagation-inner-loop-wl-D-heur*›
:: ‹[λ(*L*, *S*). *length* (*get-clauses-wl-heur S*) ≤ *snat64-max*]$_a$
unat-lit-assn^k *$*_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *PR-CONST-def unit-propagation-inner-loop-wl-D-heur-def*
by *sepref*

lemma [*sepref-fr-rules*]: ‹(*Mreturn o Tuple17-get-d*, *RETURN o literals-to-update-wl-heur*) ∈ *isasat-bounded-assn*^{k} →$_a$ *sint64-nat-assn*›
supply [*split*] = *isasat-int-splits*
unfolding *isasat-bounded-assn-def*
by *sepref-to-hoare vcg'*

lemma *select-and-remove-from-literals-to-update-wl-heur-alt-def*:
‹*select-and-remove-from-literals-to-update-wl-heur S* = *do* {
ASSERT(*literals-to-update-wl-heur S* < *length* (*fst* (*get-trail-wl-heur S*)));
ASSERT(*literals-to-update-wl-heur S* + *1* ≤ *unat32-max*);
let (*j*, *T*) = *extract-literals-to-update-wl-heur S*;
ASSERT (*j* = *literals-to-update-wl-heur S*);
L ← *isa-trail-nth* (*get-trail-wl-heur T*) *j*;
RETURN (*update-literals-to-update-wl-heur* (*j* + *1*) *T*, −*L*)
}›
by (*cases S*) (*auto simp*: *select-and-remove-from-literals-to-update-wl-heur-def state-extractors*
intro!: *ext split*: *isasat-int-splits*)

sepref-def *select-and-remove-from-literals-to-update-wlfast-code*
is ‹*select-and-remove-from-literals-to-update-wl-heur*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn* ×$_a$ *unat-lit-assn*›
supply [[*goals-limit=1*]]
unfolding *select-and-remove-from-literals-to-update-wl-heur-alt-def isasat-trail-nth-st-def*[*symmetric*]
unfolding *fold-tuple-optimizations*
supply [[*goals-limit* = *1*]]
apply (*annot-snat-const* ‹*TYPE* (*64*)›)
by *sepref*

lemma *unit-propagation-outer-loop-wl-D-heur-fast*:
‹*length* (*get-clauses-wl-heur x*) ≤ *snat64-max* ⟹
unit-propagation-outer-loop-wl-D-heur-inv x s' ⟹
length (*get-clauses-wl-heur a1'*) =
length (*get-clauses-wl-heur s'*) ⟹
length (*get-clauses-wl-heur s'*) ≤ *snat64-max*›
by (*auto simp*: *unit-propagation-outer-loop-wl-D-heur-inv-def*)

lemma *unit-propagation-outer-loop-wl-D-invI*:
‹*unit-propagation-outer-loop-wl-D-heur-inv* S_0 *S* ⟹
isa-length-trail-pre (*get-trail-wl-heur S*)›
unfolding *unit-propagation-outer-loop-wl-D-heur-inv-def*
by *blast*

sepref-def *unit-propagation-outer-loop-wl-D-fast-code*
is ‹*unit-propagation-outer-loop-wl-D-heur*›

:: ‹[λS. length (get-clauses-wl-heur S) ≤ snat64-max]$_a$ isasat-bounded-assn^d → isasat-bounded-assn›
supply [[goals-limit=1]] unit-propagation-outer-loop-wl-D-heur-fast[intro] of-nat-snat[sepref-import-param]
unit-propagation-outer-loop-wl-D-invI[intro]
unfolding unit-propagation-outer-loop-wl-D-heur-def isasat-length-trail-st-def[symmetric]
by sepref

sepref-register unit-propagation-outer-loop-wl-D-heur

experiment begin

export-llvm
length-ll-fs-heur-fast-code
unit-propagation-inner-loop-wl-loop-D-fast
cut-watch-list-heur2-fast-code
unit-propagation-inner-loop-wl-D-fast-code
isa-trail-nth-fast-code
select-and-remove-from-literals-to-update-wlfast-code
unit-propagation-outer-loop-wl-D-fast-code

end

end
theory IsaSAT-Decide-Defs
imports IsaSAT-Setup IsaSAT-VMTF IsaSAT-Bump-Heuristics IsaSAT-Phasing
begin

Chapter 19

Decide

definition *isa-bump-find-next-undef* **where** ‹
isa-bump-find-next-undef x M = (*case x of Bump-Heuristics hstable focused foc tobmp* ⇒
if foc then do {
L ← *isa-vmtf-find-next-undef focused M*;
RETURN (*L*, *Bump-Heuristics hstable* (*update-next-search L focused*) *foc tobmp*)
} *else do* {
(*L*, *hstable*) ← *isa-acids-find-next-undef hstable M*;
RETURN (*L*, *Bump-Heuristics hstable focused foc tobmp*)
})›

definition *isa-vmtf-find-next-undef-upd*
:: ‹*trail-pol* ⇒ *bump-heuristics* ⇒
((*trail-pol* × *bump-heuristics*) × *nat option*)*nres*›
where
‹*isa-vmtf-find-next-undef-upd* = (λ*M vm*. *do*{
(*L*, *vm*) ← *isa-bump-find-next-undef vm M*;
RETURN ((*M*, *vm*), *L*)
})›

definition *get-saved-phase-option-heur-pre* :: ‹*nat option* ⇒ *restart-heuristics* ⇒ *bool*› **where**
‹*get-saved-phase-option-heur-pre L* = (λ*heur*.
L ≠ *None* ⟶ *get-next-phase-heur-pre-stats True* (*the L*) *heur*)›

definition *lit-of-found-atm* **where**
‹*lit-of-found-atm* φ *L* = *SPEC* (λ*K*. (*L* = *None* ⟶ *K* = *None*) ∧
(*L* ≠ *None* ⟶ *K* ≠ *None* ∧ *atm-of* (*the K*) = *the L*))›

definition *find-unassigned-lit-wl-D-heur*
:: ‹*isasat* ⇒ (*isasat* × *nat literal option*) *nres*›
where
‹*find-unassigned-lit-wl-D-heur* = (λ*S*. *do* {
let M = *get-trail-wl-heur S*;
let vm = *get-vmtf-heur S*;
let heur = *get-heur S*;
let heur = *set-fully-propagated-heur heur*;
((*M*, *vm*), *L*) ← *isa-vmtf-find-next-undef-upd M vm*;
ASSERT(*get-saved-phase-option-heur-pre* (*L*) (*get-content heur*));
L ← *lit-of-found-atm heur L*;
let S = *set-trail-wl-heur M S*;
let S = *set-vmtf-wl-heur vm S*;
let S = *set-heur-wl-heur heur S*;

```
    RETURN (S, L)
  })⟩

definition decide-lit-wl-heur :: ⟨nat literal ⇒ isasat ⇒ isasat nres⟩ where
  ⟨decide-lit-wl-heur = (λL' S. do {
      let M = get-trail-wl-heur S;
      let stats = get-stats-heur S;
      ASSERT(isa-length-trail-pre M);
      let j = isa-length-trail M;
      let S = set-literals-to-update-wl-heur j S;
      ASSERT(cons-trail-Decided-tr-pre (L', M));
      let M = cons-trail-Decided-tr L' M;
      let S = set-trail-wl-heur M S;
      let stats = incr-decision stats;
      let S = set-stats-wl-heur stats S;
      RETURN S})⟩

definition mop-get-saved-phase-heur-st :: ⟨nat ⇒ isasat ⇒ bool nres⟩ where
   ⟨mop-get-saved-phase-heur-st = (λL S. mop-get-saved-phase-heur L (get-heur S))⟩

definition decide-wl-or-skip-D-heur
  :: ⟨isasat ⇒ (bool × isasat) nres⟩
where
  ⟨decide-wl-or-skip-D-heur S = (do {
    (S, L) ← find-unassigned-lit-wl-D-heur S;
    case L of
      None ⇒ RETURN (True, S)
    | Some L ⇒ do {
        T ← decide-lit-wl-heur L S;
        RETURN (False, T)}
  })
⟩

definition get-next-phase-st :: ⟨bool ⇒ nat ⇒ isasat ⇒ (bool) nres⟩ where
  ⟨get-next-phase-st = (λb L S.
     (get-next-phase-heur b L (get-heur S)))⟩

definition find-unassigned-lit-wl-D-heur2
  :: ⟨isasat ⇒ (isasat × nat option) nres⟩
where
  ⟨find-unassigned-lit-wl-D-heur2 = (λS. do {
      ((M, vm), L) ← isa-vmtf-find-next-undef-upd (get-trail-wl-heur S) (get-vmtf-heur S);
    RETURN (set-heur-wl-heur (set-fully-propagated-heur (get-heur S)) (set-trail-wl-heur M (set-vmtf-wl-heur
vm S)), L)
    })⟩

definition decide-wl-or-skip-D-heur' where
  ⟨decide-wl-or-skip-D-heur' = (λS. do {
      (S, L) ← find-unassigned-lit-wl-D-heur2 S;
      ASSERT(get-saved-phase-option-heur-pre L (get-restart-heuristics (get-heur S)));
      case L of
       None ⇒ RETURN (True, S)
     | Some L ⇒ do {
        L ← do {
            b ← get-next-phase-st (get-target-opts S = TARGET-ALWAYS ∨
              (get-target-opts S = TARGET-STABLE-ONLY ∧ get-restart-phase S = STABLE-MODE))
```

```
L S;
            RETURN (if b then Pos L else Neg L)};
        T ← decide-lit-wl-heur L S;
        RETURN (False, T)
      }
    })
›


lemma decide-wl-or-skip-D-heur'-decide-wl-or-skip-D-heur:
  ‹decide-wl-or-skip-D-heur' S ≤ ⇓Id (decide-wl-or-skip-D-heur S)›
proof −
  have [iff]:
    ‹{K. (∃ y. K = Some y) ∧ atm-of (the K) = x2d} = {Some (Pos x2d), Some (Neg x2d)}› for x2d
    apply (auto simp: atm-of-eq-atm-of)
    apply (case-tac y)
    apply auto
    done
  have H: ‹do {
    L ←do {ASSERT φ; P};
    Q L} =
    do {ASSERT φ; L ← P; Q L} › for P Q φ
    by auto
  have H: ‹A ≤ ⇓Id B ⟹ B ≤ ⇓Id A ⟹A = B› for A B
    by auto
  have K: ‹RES {Some (Pos x2), Some (Neg x2)} ≤ ⇓ {(x, y). x = Some y} (RES {Pos x2, Neg x2})›
    ‹RES {(Pos x2), (Neg x2)} ≤ ⇓ {(y, x). x = Some y} (RES {Some (Pos x2), Some (Neg x2)})›
for x2
    by (auto intro!: RES-refine)
  have [simp]: ‹IsaSAT-Decide-Defs.get-saved-phase-option-heur-pre a (get-restart-heuristics (set-fully-propagated-heur
(S))) =
    IsaSAT-Decide-Defs.get-saved-phase-option-heur-pre a (get-restart-heuristics (S))› for S a
   by (cases S)(auto simp: IsaSAT-Decide-Defs.get-saved-phase-option-heur-pre-def get-next-phase-heur-pre-stats-def
      get-next-phase-pre-def set-fully-propagated-heur-def set-fully-propagated-heur-stats-def)
  have S: ‹decide-wl-or-skip-D-heur S =
      (do {
                ((M, vm), L) ← isa-vmtf-find-next-undef-upd (get-trail-wl-heur S) (get-vmtf-heur S);
                 ASSERT (IsaSAT-Decide-Defs.get-saved-phase-option-heur-pre L (get-restart-heuristics
(get-heur S)));
               case L of None ⇒ RETURN (True, set-heur-wl-heur (set-fully-propagated-heur (get-heur
S)) (set-vmtf-wl-heur vm (set-trail-wl-heur M S)))
                 | Some L ⇒ do {
                   - ← SPEC (λ-::bool. True);
                   L ←RES {Pos L, Neg L};
                    T ← decide-lit-wl-heur L (set-heur-wl-heur (set-fully-propagated-heur (get-heur S))
(set-vmtf-wl-heur vm (set-trail-wl-heur M S)));
                   RETURN (False, T)
                 }})› for S a b c d e f g h i  j k l m n p q r
    unfolding decide-wl-or-skip-D-heur-def find-unassigned-lit-wl-D-heur-def Let-def
    apply (auto intro!: bind-cong[OF refl] simp: lit-of-found-atm-def split: option.splits)
    apply (rule H)
    subgoal
      apply (refine-rcg K)
      apply auto
      done
    subgoal
```

```
      apply (refine-rcg K)
      apply auto
      done
    done
  have [refine]: ‹get-saved-phase-option-heur-pre x2c (get-restart-heuristics XX) ⟹
    x2c = Some x'a ⟹ XX=YY ⟹
    get-next-phase-heur b x'a YY ≤ (SPEC (λ-::bool. True))› for x'a x1d x1e x1f x1g x2g b XX x2c YY
    by (auto simp: get-next-phase-heur-def get-saved-phase-option-heur-pre-def get-next-phase-pre-def
      get-next-phase-heur-stats-def get-next-phase-stats-def get-next-phase-heur-pre-stats-def
      split: prod.splits)
  have [refine]: ‹xa =  x'a ⟹ RETURN (if xb then Pos xa else Neg xa)
      ≤ ⇓ Id (RES {Pos x'a, Neg x'a})› for xb x'a xa
    by auto
  have [refine]: ‹decide-lit-wl-heur L S
    ≤ ⇓ Id
      (decide-lit-wl-heur La Sa)› if ‹(L, La) ∈ Id› ‹(S, Sa) ∈ Id› for L La S Sa
    using that by auto
  have [intro!]: ‹get-saved-phase-option-heur-pre (snd pa) (get-restart-heuristics l) ⟹
    get-saved-phase-option-heur-pre (snd pa) (get-restart-heuristics (set-fully-propagated-heur l))› for pa
l
   by (cases l; cases pa) (auto simp: get-saved-phase-option-heur-pre-def get-next-phase-heur-pre-stats-def
      get-next-phase-pre-def set-fully-propagated-heur-stats-def
      set-fully-propagated-heur-def)
  show ?thesis
    apply (cases S, simp only: S)
    unfolding find-unassigned-lit-wl-D-heur-def
      nres-monad3 prod.case find-unassigned-lit-wl-D-heur-def
      prod.case decide-wl-or-skip-D-heur'-def get-next-phase-st-def
      find-unassigned-lit-wl-D-heur2-def
      case-prod-beta snd-conv fst-conv bind-to-let-conv
    apply (subst Let-def)
    apply (refine-vcg
      same-in-Id-option-rel)
    subgoal by auto
    subgoal by auto
    subgoal by auto
    apply assumption back
    subgoal by auto
    subgoal by (auto simp: set-fully-propagated-heur-def split: prod.splits)
    subgoal by auto
    subgoal by auto
    subgoal by auto
    done
qed

end
theory IsaSAT-Decide
  imports IsaSAT-Decide-Defs
begin

lemma (in -)not-is-None-not-None: ‹¬is-None s ⟹ s ≠ None›
  by (auto split: option.splits)


definition bump-find-next-undef where ‹
  bump-find-next-undef A x M = (case x of Bump-Heuristics hstable focused foc tobmp ⇒
```

if foc then do {
L ← *vmtf-find-next-undef* $\mathcal{A}$ *focused M*;
RETURN (*L*, *Bump-Heuristics hstable* (*update-next-search L focused*) *foc tobmp*)
} *else do* {
(*L*, *hstable*) ← *acids-find-next-undef* $\mathcal{A}$ *hstable M*;
RETURN (*L*, *Bump-Heuristics hstable focused foc tobmp*)
})›

definition *bump-find-next-undef-upd*
:: ‹*nat multiset* ⇒ (*nat*,*nat*)*ann-lits* ⇒ *bump-heuristics* ⇒
(((*nat*,*nat*)*ann-lits* × *bump-heuristics*) × *nat option*)*nres*›
where
‹*bump-find-next-undef-upd* $\mathcal{A}$ = (λ*M vm*. *do*{
(*L*, *vm*) ← *bump-find-next-undef* $\mathcal{A}$ *vm M*;
RETURN ((*M*, *vm*), *L*)
})›

lemma *isa-bump-find-next-undef-bump-find-next-undef*:
‹(*uncurry isa-bump-find-next-undef*, *uncurry* (*bump-find-next-undef* $\mathcal{A}$)) ∈
Id ×$_r$ *trail-pol* $\mathcal{A}$ →$_f$ ⟨⟨*nat-rel*⟩*option-rel* ×$_r$ *Id*⟩*nres-rel* ›
unfolding *isa-bump-find-next-undef-def bump-find-next-undef-def uncurry-def*
defined-atm-def[*symmetric*]
apply (*intro frefI nres-relI*)
apply (*case-tac* ‹*x*›, *case-tac* ‹*fst x*›, *case-tac* ‹*y*›, *case-tac* ‹*fst y*›, *hypsubst*, *clarsimp simp only*: *fst-conv tuple4.case*)
apply (*refine-rcg isa-vmtf-find-next-undef-vmtf-find-next-undef*[*THEN fref-to-Down-curry*]
isa-acids-find-next-undef-acids-find-next-undef[*THEN fref-to-Down-curry*])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done

lemma *isa-vmtf-find-next-undef-vmtf-find-next-undef*:
‹(*uncurry isa-vmtf-find-next-undef-upd*, *uncurry* (*bump-find-next-undef-upd* $\mathcal{A}$)) ∈
trail-pol $\mathcal{A}$ ×$_r$ *Id* →$_f$
⟨*trail-pol* $\mathcal{A}$ ×$_f$ *Id* ×$_f$ ⟨*nat-rel*⟩*option-rel*⟩*nres-rel* ›
unfolding *isa-vmtf-find-next-undef-upd-def isa-vmtf-find-next-undef-upd-def uncurry-def*
defined-atm-def[*symmetric*] *bump-find-next-undef-upd-def*
apply (*intro frefI nres-relI*)
apply (*refine-rcg isa-bump-find-next-undef-bump-find-next-undef*[*THEN fref-to-Down-curry*])
subgoal by *auto*
subgoal by (*auto simp*: *update-next-search-def split*: *prod.splits*)
done
term *isa-bump-find-next-undef*
term *bump-find-next-undef*
lemma *bump-find-next-undef-ref*:
assumes
vmtf: ‹*x* ∈ *bump-heur* $\mathcal{A}$ *M*›
shows ‹*bump-find-next-undef* $\mathcal{A}$ *x M*
≤ ⇓ *Id* (*SPEC* (λ(*L*, *bmp*).
(*L* ≠ *None* ⟶ *bmp* ∈ *bump-heur* $\mathcal{A}$ (*Decided* (*Pos* (*the L*)) # *M*)) ∧
(*L* = *None* ⟶ *bmp* ∈ *bump-heur* $\mathcal{A}$ *M*) ∧
(*L* = *None* ⟶ (∀ *L*∈#$\mathcal{L}_{all}$ $\mathcal{A}$. *defined-lit M L*)) ∧

```
    (L ≠ None ⟶ Pos (the L) ∈# \<L>_all \<A> ∧ undefined-lit M (Pos (the L))))))›
  using assms
  unfolding bump-find-next-undef-def
  apply (cases ‹get-focused-heuristics x›; cases ‹get-stable-heuristics x›)
  apply (cases x, simp only: tuple4.case)
  by (refine-vcg lhs-step-If)
   (auto intro!: vmtf-find-next-undef-ref[THEN order-trans]
      acids-find-next-undef[THEN order-trans] dest: vmtf-consD
     simp: bump-heur-def update-next-search-def)

definition find-undefined-atm
  :: ‹nat multiset ⇒ (nat,nat) ann-lits ⇒ bump-heuristics ⇒
       (((nat,nat) ann-lits × bump-heuristics) × nat option) nres›
where
  ‹find-undefined-atm \<A> M - = SPEC(λ((M', vm), L).
     (L ≠ None ⟶ Pos (the L) ∈# \<L>_all \<A> ∧ undefined-atm M (the L)) ∧
  (L = None ⟶ (∀ K∈# \<L>_all \<A>. defined-lit M K)) ∧ M = M' ∧
  (L = None ⟶ vm ∈ bump-heur \<A> M)∧
  (L ≠ None ⟶ vm ∈ bump-heur \<A> (Decided (Pos (the L)) # M)))›

definition lit-of-found-atm-D-pre where
‹lit-of-found-atm-D-pre \<A> = (λ((φ, -), L). L ≠ None ⟶
       (the L ≤ unat32-max div 2 ∧ the L ∈# \<A> ∧ phase-save-heur-rel \<A> φ ))›

lemma lit-of-found-atm-D-pre:
  ‹heuristic-rel \<A> heur ⟹ isasat-input-bounded \<A> ⟹ (L ≠ None ⟹ the L ∈# \<A>) ⟹
    get-saved-phase-option-heur-pre (L) (get-content heur)›
  by (auto simp: lit-of-found-atm-D-pre-def phase-saving-def heuristic-rel-def phase-save-heur-rel-def
   get-saved-phase-option-heur-pre-def get-next-phase-pre-def heuristic-rel-stats-def get-next-phase-heur-pre-stats-def
    atms-of-\<L>_all-\<A>_in in-\<L>_all-atm-of-in-atms-of-iff dest: bspec[of - - ‹Pos (the L)›])

definition find-unassigned-lit-wl-D-heur-pre where
  ‹find-unassigned-lit-wl-D-heur-pre S ⟷
    (
      ∃ T U.
        (S, T) ∈ state-wl-l None ∧
        (T, U) ∈ twl-st-l None ∧
        twl-struct-invs U ∧
        literals-are-\<L>_in (all-atms-st S) S ∧
        get-conflict-wl S = None
    )›

definition twl-st-heur-decide-find :: ‹nat literal ⇒ (isasat × nat twl-st-wl) set› where
[unfolded Let-def]: ‹twl-st-heur-decide-find L =
  {(S, T).
  let M' = get-trail-wl-heur S; N' = get-clauses-wl-heur S; D' = get-conflict-wl-heur S;
    W' = get-watched-wl-heur S; j = literals-to-update-wl-heur S; outl = get-outlearned-heur S;
    cach = get-conflict-cach S; clvls = get-count-max-lvls-heur S;
    vm = get-vmtf-heur S;
    vdom = get-aivdom S; heur = get-heur S; old-arena = get-old-arena S;
    lcount = get-learned-count S;
    occs = get-occs S in
  let M = get-trail-wl T; LM = Decided (L) # get-trail-wl T;
      N = get-clauses-wl T;  D = get-conflict-wl T;
      Q = literals-to-update-wl T;
      W = get-watched-wl T; N0 = get-init-clauses0-wl T; U0 = get-learned-clauses0-wl T;
```

NS = *get-subsumed-init-clauses-wl* T; US = *get-subsumed-learned-clauses-wl* T;
NEk = *get-kept-unit-init-clss-wl* T; UEk = *get-kept-unit-learned-clss-wl* T;
NE = *get-unkept-unit-init-clss-wl* T; UE = *get-unkept-unit-learned-clss-wl* T *in*
$(M', M) \in$ *trail-pol* (*all-atms-st* T) $\wedge$
valid-arena N' N (*set* (*get-vdom-aivdom vdom*)) $\wedge$
$(D', D) \in$ *option-lookup-clause-rel* (*all-atms-st* T) $\wedge$
$(D = None \longrightarrow j \leq length\ M) \wedge$
$Q =$ *uminus* ‘# *lit-of* ‘# *mset* (*drop* j (*rev* M)) $\wedge$
$(W', W) \in \langle Id \rangle$*map-fun-rel* ($D_0$ (*all-atms-st* T)) $\wedge$
vm $\in$ *bump-heur* (*all-atms-st* T) *LM* $\wedge$
no-dup M $\wedge$
clvls $\in$ *counts-maximum-level* M D $\wedge$
cach-refinement-empty (*all-atms-st* T) *cach* $\wedge$
out-learned M D *outl* $\wedge$
clss-size-corr N *NE UE NEk UEk NS US N0 U0 lcount* $\wedge$
vdom-m (*all-atms-st* T) W $N \subseteq$ *set* (*get-vdom-aivdom vdom*) $\wedge$
aivdom-inv-dec vdom (*dom-m* N) $\wedge$
isasat-input-bounded (*all-atms-st* T) $\wedge$
isasat-input-nempty (*all-atms-st* T) $\wedge$
old-arena $= [] \wedge$
heuristic-rel (*all-atms-st* T) *heur* $\wedge$
(*occs*, *empty-occs-list* (*all-atms-st* T)) $\in$ *occurrence-list-ref*
}›

abbreviation *twl-st-heur-decide-find'''*
:: ‹*nat literal* ⇒ *nat* ⇒ (*isasat* × *nat twl-st-wl*) *set*›
where
‹*twl-st-heur-decide-find'''* L $r \equiv \{(S, T).\ (S, T) \in$ *twl-st-heur-decide-find* $L \wedge$
length (*get-clauses-wl-heur* S) $= r\}$›

lemma *vmtf-find-next-undef-upd*:
‹(*uncurry* (*bump-find-next-undef-upd* $\mathcal{A}$), *uncurry* (*find-undefined-atm* $\mathcal{A}$)) $\in$
[$\lambda(M, vm).\ vm \in$ *bump-heur* $\mathcal{A}$ $M]_f$ *Id* $\times_f$ *Id* $\to \langle Id \times_f Id \times_f \langle$*nat-rel*$\rangle$*option-rel*$\rangle$*nres-rel*›
unfolding *bump-find-next-undef-upd-def find-undefined-atm-def*
update-next-search-def uncurry-def
apply (*intro frefI nres-relI*)
apply (*clarify*)
apply (*rule bind-refine-spec*)
prefer *2*
apply (*rule bump-find-next-undef-ref*[*simplified*])
by (*auto intro*!: *RETURN-SPEC-refine simp*: *image-image defined-atm-def*[*symmetric*])

lemma *find-unassigned-lit-wl-D'-find-unassigned-lit-wl-D*:
‹(*find-unassigned-lit-wl-D-heur*, *find-unassigned-lit-wl*) $\in$
[*find-unassigned-lit-wl-D-heur-pre*]$_f$
$\{(S, T).\ (S, T) \in$ *twl-st-heur'''* $r \wedge$ *learned-clss-count* $S = u\} \to$
$\langle\{((T, L), (T', L')).\ (L \neq None \longrightarrow (T, T') \in$ *twl-st-heur-decide-find'''* (*the* L) $r) \wedge$
$(L = None \longrightarrow (T, T') \in$ *twl-st-heur'''* $r) \wedge L = L' \wedge$ *learned-clss-count* $T = u \wedge$
$(L \neq None \longrightarrow$ *undefined-lit* (*get-trail-wl* T') (*the* L) $\wedge$ *the* $L \in\#$ *all-lits-st* $T') \wedge$
get-conflict-wl $T' = None\}\rangle$*nres-rel*›
proof −
have [*simp*]: ‹*undefined-lit* M (*Pos* (*atm-of* y)) = *undefined-lit* M y› **for** M y
by (*auto simp*: *defined-lit-map*)
have [*simp*]: ‹*defined-atm* M (*atm-of* y) = *defined-lit* M y› **for** M y
by (*auto simp*: *defined-lit-map defined-atm-def*)

have *ID-R*: ‹*Id* $\times_r$ ⟨*Id*⟩*option-rel* = *Id*›
by *auto*

define *unassigned-atm* **where**
‹*unassigned-atm S L* ⟷ (∃ *M N D NE UE NS US WS Q*.
S = (*M*, *N*, *D*, *NE*, *UE*, *NS*, *US*, *WS*, *Q*) ∧
(*L* ≠ *None* ⟶
undefined-lit M (*the L*) ∧ *the L* ∈# $\mathcal{L}_{all}$ (*all-atms-st S*) ∧
atm-of (*the L*) ∈# *all-atms-st S*) ∧
(*L* = *None* ⟶ (∄ *L'*. *undefined-lit M L'* ∧
atm-of L' ∈# *all-atms-st S*)))›
for *L* :: ‹*nat literal option*› **and** *S* :: ‹*nat twl-st-wl*›

have *all-lits-st-atm*: ‹*L* ∈# *all-lits-st S* ⟷ *atm-of L* ∈# *all-atms-st S*› **for** *L S*
unfolding *all-lits-st-def all-atms-st-def in-all-lits-of-mm-ain-atms-of-iff*
all-lits-def all-atms-def **by** (*metis atm-of-all-lits-of-mm*(*1*))

have *find-unassigned-lit-wl-D-alt-def*:
‹*find-unassigned-lit-wl S* = *do* {
L ← *SPEC*(*unassigned-atm S*);
L ← *RES* {*L*, *map-option uminus L*};
SPEC(λ((*M*, *N*, *D*, *NE*, *UE*, *N0*, *U0*, *WS*, *Q*), *L'*).
S = (*M*, *N*, *D*, *NE*, *UE*, *N0*, *U0*, *WS*, *Q*) ∧ *L* = *L'*)
}› **for** *S*
unfolding *find-unassigned-lit-wl-def RES-RES-RETURN-RES unassigned-atm-def*
RES-RES-RETURN-RES all-lits-def in-all-lits-of-mm-ain-atms-of-iff
in-$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *in-set-all-atms-iff all-lits-st-atm*
by (*cases S*) *auto*

have *isa-vmtf-find-next-undef-upd*:
‹*isa-vmtf-find-next-undef-upd* (*get-trail-wl-heur S*)
(*get-vmtf-heur S*)
≤ ⇓ {(((*M*, *vm*), *A*), *L*). *A* = *map-option atm-of L* ∧
unassigned-atm (*bt*, *bu*, *bv*, *bw*, *bx*, *by*, *bz*, *baa*, *bab*) *L* ∧
(*L* ≠ *None* ⟶ *vm* ∈ *bump-heur* (*all-atms-st* (*bt*, *bu*, *bv*, *bw*, *bx*, *by*, *bz*, *baa*, *bab*)) (*Decided*
(*Pos* (*the A*)) # *bt*)) ∧
(*L* = *None* ⟶ *vm* ∈ *bump-heur* (*all-atms-st* (*bt*, *bu*, *bv*, *bw*, *bx*, *by*, *bz*, *baa*, *bab*)) *bt*) ∧
(*L* ≠ *None* ⟶ *the A* ∈# *all-atms-st* (*bt*, *bu*, *bv*, *bw*, *bx*, *by*, *bz*, *baa*, *bab*)) ∧
(*M*, *bt*) ∈ *trail-pol* (*all-atms-st* (*bt*, *bu*, *bv*, *bw*, *bx*, *by*, *bz*, *baa*, *bab*))}
(*SPEC* (*unassigned-atm* (*bt*, *bu*, *bv*, *bw*, *bx*, *by*, *bz*, *baa*, *bab*)))›
(**is** ‹- ≤ ⇓ *?find* -›)
if
pre: ‹*find-unassigned-lit-wl-D-heur-pre* (*bt*, *bu*, *bv*, *bw*, *bx*, *by*, *bz*, *baa*, *bab*)› **and**
T: ‹(*S*,
bt, *bu*, *bv*, *bw*, *bx*, *by*, *bz*, *baa*, *bab*)
∈ *twl-st-heur*› **and**
‹*r* =
length
(*get-clauses-wl-heur*
S)›
for *a aa ab ac ad b ae af ag ba ah ai aj ak al am bb an bc ao ap aq bd ar as at*
au av aw be ax ay az bf bg bh bi bj bk bl bm bn bo bp bq br bs bt bu bv
bw bx by bz heur baa bab stats S
proof −
let *?*$\mathcal{A}$ = ‹*all-atms-st* (*bt*, *bu*, *bv*, *bw*, *bx*, *by*, *bz*, *baa*, *bab*)›

```
  have pol:
    ‹(get-trail-wl-heur S, bt) ∈ trail-pol ?A›
    using that by (cases bz; auto simp: twl-st-heur-def)
  have [intro]:
     ‹Multiset.Ball (\mathcal{L}_{all} ?A) (defined-lit bt) ⟹
atm-of L' ∈# ?A ⟹
undefined-lit bt L' ⟹ False› for L'
    by (auto simp: atms-of-ms-def
  all-lits-of-mm-union ran-m-def all-lits-of-mm-add-mset \mathcal{L}_{all}-union
  eq-commute[of - ‹the (fmlookup - -)›] \mathcal{L}_{all}-atm-of-all-lits-of-m
 atms-of-def \mathcal{L}_{all}-add-mset
dest!: multi-member-split
 )

  show ?thesis
    apply (rule order.trans)
    apply (rule isa-vmtf-find-next-undef-vmtf-find-next-undef[of ?A, THEN fref-to-Down-curry,
of - - bt ‹get-vmtf-heur S›])
    subgoal by fast
    subgoal
using pol by (cases ‹get-vmtf-heur S›) (auto simp: twl-st-heur-def all-atms-def[symmetric])
    apply (rule order.trans)
    apply (rule ref-two-step')
    apply (rule vmtf-find-next-undef-upd[THEN fref-to-Down-curry, of ?A bt ‹get-vmtf-heur S›])
    subgoal using T by (auto simp: all-atms-def twl-st-heur-def)
    subgoal by auto
    subgoal using pre
      apply (cases bab)
apply (auto 5 0 simp: find-undefined-atm-def unassigned-atm-def conc-fun-RES all-atms-def[symmetric]
  mset-take-mset-drop-mset'
  intro!: RES-refine intro: )
apply (auto intro:  simp: defined-atm-def)
apply (rule-tac x = ‹Some (Pos ya)› in exI)
apply (auto intro: simp: defined-atm-def  in-\mathcal{L}_{all}-atm-of-\mathcal{A}_{in}
  in-set-all-atms-iff)
apply (rule-tac x = ‹Some (Pos y)› in exI)
apply (auto intro: simp: defined-atm-def  in-\mathcal{L}_{all}-atm-of-\mathcal{A}_{in}
  in-set-all-atms-iff)
done
  done
qed

 have lit-of-found-atm: ‹lit-of-found-atm ao' x2a
≤ ⇓ {(L, L'). L = L' ∧ map-option atm-of  L = x2a}
  (RES {L, map-option uminus L})›
  if
    ‹find-unassigned-lit-wl-D-heur-pre (bt, bu, bv, bw, bx, by, bz, baa, bab)› and
    ‹(S, bt, bu, bv, bw, bx, by, bz, baa, bab) ∈ twl-st-heur› and
    ‹r = length (get-clauses-wl-heur S)› and
    ‹(x, L) ∈ ?find bt bu bv bw bx by bz baa bab› and
    ‹x1 = (x1a, x2)› and
    ‹x = (x1, x2a)›
   for a aa ab ac ad b ae af ag ba ah ai aj ak al am bb an bc ao ap aq bd ar as at
     au av aw be ax ay az bf bg bh bi bj bk bl bm bn bo bp bq br bs bt bu bv
     bw bx by bz x L x1 x1a x2 x2a heur baa bab ao' stats S
 proof −
```

show *?thesis*
using *that* **unfolding** *lit-of-found-atm-def*
by (*auto simp*: *atm-of-eq-atm-of twl-st-heur-def intro*!: *RES-refine*)
qed
have [*simp*]: ‹*vmtf* $\mathcal{A}$ (*Decided* (*Pos* (*atm-of ap*)) # *aa*) = *vmtf* $\mathcal{A}$ (*Decided ap* # *aa*)›
‹*vmtf* $\mathcal{A}$ (*Decided* (−*ap*) # *aa*) = *vmtf* $\mathcal{A}$ (*Decided ap* # *aa*)› **for** $\mathcal{A}$ *ap aa*
unfolding *vmtf-def vmtf-*$\mathcal{L}_{all}$*-def*
by *auto*
have [*simp*]: ‹*acids* $\mathcal{A}$ (*Decided* (*Pos* (*atm-of ap*)) # *aa*) = *acids* $\mathcal{A}$ (*Decided ap* # *aa*)›
‹*acids* $\mathcal{A}$ (*Decided* (−*ap*) # *aa*) = *acids* $\mathcal{A}$ (*Decided ap* # *aa*)› **for** $\mathcal{A}$ *ap aa*
unfolding *acids-def defined-lit-map*
by *auto*
have [*simp*]: ‹*bump-heur* $\mathcal{A}$ (*Decided* (*Pos* (*atm-of ap*)) # *aa*)= *bump-heur* $\mathcal{A}$ (*Decided ap* # *aa*)›
‹*bump-heur* $\mathcal{A}$ (*Decided* (−*ap*) # *aa*) = *bump-heur* $\mathcal{A}$ (*Decided ap* # *aa*)› **for** $\mathcal{A}$ *ap aa bc*
unfolding *bump-heur-def*
by *auto*
have [*dest*]: ‹*find-unassigned-lit-wl-D-heur-pre* (*ca*, *cb*, *cc*, *cd*, *ce*, *cf*, *cg*, *ch*, *ci*, *cx*, *cy*) ⟹ *cc* = *None*›
for *ca cb cc cd ce cf cg ch ci cy cx*
unfolding *find-unassigned-lit-wl-D-heur-pre-def* **by** *auto*
show *?thesis*
unfolding *find-unassigned-lit-wl-D-heur-def find-unassigned-lit-wl-D-alt-def find-undefined-atm-def*
ID-R Let-def
apply (*intro frefI nres-relI*)
apply *clarify*
apply *refine-vcg*
apply (*rule isa-vmtf-find-next-undef-upd*; *assumption*)
subgoal
by (*rule lit-of-found-atm-D-pre*)
(*auto simp add*: *twl-st-heur-def in-*$\mathcal{L}_{all}$*-atm-of-in-atms-of-iff Ball-def image-image*
mset-take-mset-drop-mset′ all-atms-def[*symmetric*] *unassigned-atm-def*
simp del: *twl-st-of-wl.simps dest*!: *intro*!: *RETURN-RES-refine*)
apply (*rule lit-of-found-atm*; *assumption*)
subgoal for *a aa ab ac ad b ae af ag ba ah ai aj ak al am bb an bc ao L L′*
by (*cases am*)
(*clarsimp-all simp*: *twl-st-heur-def twl-st-heur-decide-find-def unassigned-atm-def atm-of-eq-atm-of*
uminus-$\mathcal{A}_{in}$*-iff learned-clss-count-def*
all-lits-st-alt-def[*symmetric*]
simp del: *twl-st-of-wl.simps dest*!: *intro*!: *RETURN-RES-refine*;
auto simp: *atm-of-eq-atm-of uminus-*$\mathcal{A}_{in}$*-iff all-atms-st-def*; *fail*)+
done
qed

lemma *nofail-get-next-phase*:
‹*get-next-phase-heur-pre-stats True L* (*get-restart-heuristics* φ) ⟹
nofail (*get-next-phase-heur b L* φ)›
by (*cases* φ) (*auto simp*: *phase-save-heur-rel-def phase-saving-def get-next-phase-heur-def get-next-phase-heur-stats-def*
get-next-phase-heur-pre-stats-def get-next-phase-stats-def
nofail-def bind-ASSERT-eq-if $\mathcal{L}_{all}$*-add-mset atms-of-def get-next-phase-pre-def split*: *if-splits*
dest!: *multi-member-split*)

lemma *all-atms-st-cons-trail-Decided*[*simp*]:
‹*all-atms-st* (*cons-trail-Decided x′a x1b*, *oth*) = *all-atms-st* (*x1b*, *oth*)› **and**
all-atms-st-cons-trail-empty-Q:
‹*NO-MATCH* {#} *Q* ⟹

```
    all-atms-st (x1b, N, D, NS, US, NEk, UEk, NE, UE, N0, U0, Q, W) = all-atms-st (x1b, N, D,
NS, US, NEk, UEk, NE, UE, N0, U0, {#}, W)›
  by (cases oth) (auto simp: cons-trail-Decided-def all-atms-st-def)

lemma decide-wl-or-skip-D-heur-decide-wl-or-skip-D:
  ‹(decide-wl-or-skip-D-heur, decide-wl-or-skip) ∈
    {(S, T). (S, T) ∈ twl-st-heur''' r ∧ learned-clss-count S =u} →f
    ⟨bool-rel ×f {(S, T). (S, T) ∈ twl-st-heur''' r ∧ learned-clss-count S =u}⟩ nres-rel›
  (is ‹- ∈ ?A →f -›)
proof −
  have [simp]:
    ‹rev (cons-trail-Decided L M) = rev M @ [Decided L]›
    ‹no-dup (cons-trail-Decided L M) = no-dup (Decided L # M)›
    ‹bump-heur A (cons-trail-Decided L M) = bump-heur A (Decided L # M)›
    for M L A
    by (auto simp: cons-trail-Decided-def)

  have final: ‹decide-lit-wl-heur xb x1a
 ≤ SPEC
    (λT.  do {
                RETURN (False, T)}
  ≤ SPEC
     (λc. (c, False, decide-lit-wl x'a x1)
   ∈ bool-rel ×f ?A))›
    if
      ‹(x, y) ∈ {(S, T). (S, T) ∈ twl-st-heur''' r ∧ learned-clss-count S = u}› and
      ‹(xa, x')
       ∈ {((T, L), T', L').
    (L ≠ None ⟶ (T, T') ∈ twl-st-heur-decide-find''' (the L) r) ∧
    (L = None ⟶ (T, T') ∈ twl-st-heur''' r) ∧
    L = L' ∧
    learned-clss-count T = u ∧
    (L ≠ None ⟶
     undefined-lit (get-trail-wl T') (the L) ∧ the L ∈# all-lits-st T') ∧
    get-conflict-wl T' = None}› and
      st:
        ‹x' = (x1, x2)›
        ‹xa = (x1a, x2a)›
        ‹x2a = Some xb›
        ‹x2 = Some x'a› and
      ‹(xb, x'a) ∈ nat-lit-lit-rel›
    for x y xa x' x1 x2 x1a x2a xb x'a
  proof −
    show ?thesis
      unfolding decide-lit-wl-heur-def
        decide-lit-wl-def
      apply refine-vcg
      subgoal
        by (rule isa-length-trail-pre[of - ‹get-trail-wl x1› ‹all-atms-st x1›])
   (use that(2) in ‹auto simp: twl-st-heur-decide-find-def twl-st-heur-def st all-atms-def[symmetric]›)
      subgoal
        by (rule cons-trail-Decided-tr-pre[of - ‹get-trail-wl x1› ‹all-atms-st x1›])
   (use that(2) in ‹auto simp: twl-st-heur-decide-find-def st all-atms-def[symmetric]
          all-lits-st-alt-def[symmetric]›)
      subgoal
        using that(2) unfolding cons-trail-Decided-def[symmetric] st
```

```
      apply (clarsimp simp: twl-st-heur-def)[]
      apply (clarsimp simp add: twl-st-heur-def twl-st-heur-decide-find-def all-atms-def[symmetric]
   isa-length-trail-length-u[THEN fref-to-Down-unRET-Id] out-learned-def
        all-lits-st-alt-def[symmetric] all-atms-st-cons-trail-empty-Q
  intro!: cons-trail-Decided-tr[THEN fref-to-Down-unRET-uncurry]
    isa-vmtf-consD)
      by (auto simp add: twl-st-heur-def all-atms-def[symmetric] learned-clss-count-def
   isa-length-trail-length-u[THEN fref-to-Down-unRET-Id] out-learned-def
        all-lits-st-alt-def[symmetric] all-atms-st-cons-trail-empty-Q
  intro!: cons-trail-Decided-tr[THEN fref-to-Down-unRET-uncurry]
    isa-vmtf-consD)
     done
 qed

 have decide-wl-or-skip-alt-def: ‹decide-wl-or-skip S = (do {
   ASSERT(decide-wl-or-skip-pre S);
   (S, L) ← find-unassigned-lit-wl S;
   case L of
     None ⇒ RETURN (True, S)
   | Some L ⇒ RETURN (False, decide-lit-wl L S)
 })› for S
 unfolding decide-wl-or-skip-def by auto
 show ?thesis
   supply [[goals-limit=1]]
   unfolding decide-wl-or-skip-D-heur-def decide-wl-or-skip-alt-def decide-wl-or-skip-pre-def
    decide-l-or-skip-pre-def twl-st-of-wl.simps[symmetric]
   apply (intro nres-relI frefI same-in-Id-option-rel)
   apply (refine-vcg find-unassigned-lit-wl-D'-find-unassigned-lit-wl-D[of r u, THEN fref-to-Down])
   subgoal for x y
     unfolding decide-wl-or-skip-pre-def find-unassigned-lit-wl-D-heur-pre-def
decide-wl-or-skip-pre-def decide-l-or-skip-pre-def decide-or-skip-pre-def
      apply normalize-goal+
      apply (rule-tac x = xa in exI)
      apply (rule-tac x = xb in exI)
      apply auto
      done
   apply (rule same-in-Id-option-rel)
   subgoal by (auto simp del: simp: twl-st-heur-def)
   subgoal by auto
   by (rule final; assumption?)
qed


lemma bind-triple-unfold:
  ‹do {
    ((M, vm), L) ← (P :: - nres);
    f ((M, vm), L)
} =
do {
    x ← P;
    f x
}›
  by (intro bind-cong) auto

lemma decide-wl-or-skip-D-heur'-decide-wl-or-skip-D-heur2:
  ‹(decide-wl-or-skip-D-heur', decide-wl-or-skip-D-heur) ∈ Id →f ⟨Id⟩nres-rel›
```

by (*intro frefI nres-relI*) (*use decide-wl-or-skip-D-heur'-decide-wl-or-skip-D-heur* **in** *auto*)

end
theory *IsaSAT-Decide-LLVM*
imports *IsaSAT-Decide-Defs IsaSAT-VMTF-State-LLVM IsaSAT-Setup-LLVM IsaSAT-Rephase-State-LLVM*
begin
lemma *decide-lit-wl-heur-alt-def*:
‹*decide-lit-wl-heur* = (λ*L' S*. *do* {
let (*M*, *S*) = *extract-trail-wl-heur S*;
let (*stats*, *S*) = *extract-stats-wl-heur S*;
ASSERT(*isa-length-trail-pre M*);
let j = *isa-length-trail M*;
let S = *update-literals-to-update-wl-heur j S*;
ASSERT(*cons-trail-Decided-tr-pre* (*L'*, *M*));
let M = *cons-trail-Decided-tr L' M*;
let stats = *incr-decision stats*;
let S = *update-trail-wl-heur M S*;
let S = *update-stats-wl-heur stats S*;
RETURN S})›
by (*auto simp*: *decide-lit-wl-heur-def state-extractors split*: *isasat-int-splits intro*!: *ext*)

sepref-def *decide-lit-wl-fast-code*
is ‹*uncurry decide-lit-wl-heur*›
:: ‹*unat-lit-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *decide-lit-wl-heur-alt-def*
by *sepref*

sepref-register *find-unassigned-lit-wl-D-heur decide-lit-wl-heur*

sepref-register *isa-vmtf-find-next-undef*

sepref-def *isa-vmtf-find-next-undef-code* **is**
‹*uncurry isa-vmtf-find-next-undef*› :: ‹*vmtf-assn*^{k} $*_a$ *trail-pol-fast-assn*^{k} $\rightarrow_a$ *atom.option-assn*›
unfolding *isa-vmtf-find-next-undef-def vmtf-assn-def*
unfolding *atom.fold-option*
apply (*rewrite* **in** ‹*WHILEIT* - ⌑› *short-circuit-conv*)
supply [[*goals-limit* = *1*]]
apply *annot-all-atm-idxs*
by *sepref*

sepref-def *isa-bump-find-next-undef-code* **is**
‹*uncurry isa-bump-find-next-undef*› :: ‹*heuristic-bump-assn*^{d} $*_a$ *trail-pol-fast-assn*^{k} $\rightarrow_a$ *atom.option-assn*
$\times_a$ *heuristic-bump-assn*›
unfolding *isa-bump-find-next-undef-def*
unfolding *atom.fold-option*
supply [[*goals-limit* = *1*]]
apply *annot-all-atm-idxs*
by *sepref*

sepref-register *update-next-search*
sepref-def *update-next-search-code* **is**
‹*uncurry* (*RETURN oo update-next-search*)› :: ‹*atom.option-assn*^{k} $*_a$ *vmtf-assn*^{d} $\rightarrow_a$ *vmtf-assn*›
unfolding *update-next-search-def vmtf-assn-def*
by *sepref*

sepref-register *isa-vmtf-find-next-undef-upd mop-get-saved-phase-heur get-next-phase-st*

sepref-def *isa-vmtf-find-next-undef-upd-code* **is**
  ‹*uncurry isa-vmtf-find-next-undef-upd*›
    :: ‹*trail-pol-fast-assn*^{d} $*_a$ *heuristic-bump-assn*^{d} $\rightarrow_a$ (*trail-pol-fast-assn* $\times_a$ *heuristic-bump-assn*) $\times_a$ *atom.option-assn*›
  unfolding *isa-vmtf-find-next-undef-upd-def*
  by *sepref*

lemma *find-unassigned-lit-wl-D-heur2-alt-def*:
  ‹*find-unassigned-lit-wl-D-heur2* = (λ*S*. *do* {
    let (*M*, *S*) = *extract-trail-wl-heur S*;
    let (*vm*, *S*) = *extract-vmtf-wl-heur S*;
    let (*heur*, *S*) = *extract-heur-wl-heur S*;
      ((*M*, *vm*), *L*) ← *isa-vmtf-find-next-undef-upd M vm*;
    RETURN (*update-heur-wl-heur* (*set-fully-propagated-heur heur*) (*update-trail-wl-heur M* (*update-vmtf-wl-heur vm S*)), *L*)
    })›
   by (*auto simp*: *find-unassigned-lit-wl-D-heur2-def state-extractors split*: *isasat-int-splits intro*!: *ext*)
sepref-register *find-unassigned-lit-wl-D-heur2*
sepref-def *find-unassigned-lit-wl-D-heur-impl*
  is ‹*find-unassigned-lit-wl-D-heur2*›
  :: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn* $\times_a$ *atom.option-assn*›
  unfolding *find-unassigned-lit-wl-D-heur2-alt-def*
  by *sepref*

sepref-definition *get-next-phase-heur-stats-impl′*
  is ‹*uncurry2* (λ*S C′ D′*. *get-next-phase-heur C′ D′ S*)›
  :: ‹[*uncurry2* (λ*S C D*. *True*)]$_a$ *heuristic-assn*^{k} $*_a$ *bool1-assn*^{k} $*_a$ *atom-assn*^{k} $\rightarrow$ *bool1-assn*›
  by *sepref*

definition *get-next-phase-st′-impl* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
  ‹*get-next-phase-st′-impl* = (λ*N C D*. *read-heur-wl-heur-code* (*get-next-phase-heur-stats-impl C D*) *N*)›

definition *get-next-phase-st′* :: ‹-› **where**
  ‹*get-next-phase-st′ N C D* = (*get-next-phase-st C D N*)›

global-interpretation *get-next-phase*: *read-heur-param-adder2* **where**
  R = *bool1-rel* **and**
  R′ = *atom-rel* **and**
  f′ = ‹ λ*C D S*. *get-next-phase-heur C D S*› **and**
  f = ‹λ*C D S*. *get-next-phase-heur-stats-impl C D S*› **and**
  x-assn = ‹*bool1-assn*› **and**
  P = ‹(λ- - -. *True*)›
  rewrites
    ‹(λ*N C D*. *read-heur-wl-heur-code* (*get-next-phase-heur-stats-impl C D*) *N*) = *get-next-phase-st′-impl*›
and
    ‹(λ*N C D*. *read-heur-wl-heur* (*get-next-phase-heur C D*) *N*) = *get-next-phase-st′*›
  apply *unfold-locales*
  apply (*rule get-next-phase-heur-stats-impl′.refine*[*unfolded get-next-phase-heur-stats-impl′-def*])
  subgoal by (*auto simp*: *get-next-phase-st′-impl-def*)
 subgoal by (*auto simp*: *read-all-st-def get-next-phase-st-def get-next-phase-st′-def split*: *isasat-int-splits intro*!: *ext*)
  done

lemmas [*sepref-fr-rules*] = *get-next-phase.refine*
lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
get-next-phase-st′-impl-def[*unfolded read-all-st-code-def*]

sepref-def *get-next-phase-st-impl*
is ‹*uncurry2 get-next-phase-st*›
:: ‹*bool1-assn*^{k} $*_a$ *atom-assn*^{k} $*_a$ *isasat-bounded-assn*^{k} $\rightarrow_a$ *bool1-assn*›
unfolding *get-next-phase-st′-def*[*symmetric*]
by *sepref*

sepref-def *decide-wl-or-skip-D-fast-code*
is ‹*decide-wl-or-skip-D-heur*›
:: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ *bool1-assn* $\times_a$ *isasat-bounded-assn*›
supply[[*goals-limit=1*]]
apply (*rule hfref-refine-with-pre*[*OF decide-wl-or-skip-D-heur′-decide-wl-or-skip-D-heur*, *unfolded Down-id-eq*])
unfolding *decide-wl-or-skip-D-heur′-def option.case-eq-if atom.fold-option*
by *sepref*

experiment begin

export-llvm
decide-lit-wl-fast-code
isa-vmtf-find-next-undef-code
update-next-search-code
isa-vmtf-find-next-undef-upd-code
decide-wl-or-skip-D-fast-code

end

end
theory *IsaSAT-Other-Defs*
imports *IsaSAT-Conflict-Analysis-Defs IsaSAT-Backtrack-Defs IsaSAT-Decide-Defs*
begin

Chapter 20

Combining Together: the Other Rules

definition *cdcl-twl-o-prog-wl-D-heur*
:: ‹*isasat* ⇒ (*bool* × *isasat*) *nres*›
where
‹*cdcl-twl-o-prog-wl-D-heur S* =
do {
if get-conflict-wl-is-None-heur S
then decide-wl-or-skip-D-heur S
else do {
if count-decided-st-heur S > *0*
then do {
T ← *skip-and-resolve-loop-wl-D-heur S*;
ASSERT(*length* (*get-clauses-wl-heur S*) = *length* (*get-clauses-wl-heur T*));
ASSERT(*get-learned-count S* = *get-learned-count T*);
U ← *backtrack-wl-D-nlit-heur T*;
U ← *isasat-current-status U*; — Print some information every once in a while
RETURN (*False*, *U*)
}
else RETURN (*True*, *S*)
}
}
›

end
theory *IsaSAT-CDCL-Defs*
imports *IsaSAT-Propagate-Conflict-Defs IsaSAT-Other-Defs IsaSAT-Show*
begin

Combining Together: Full Strategy **definition** *cdcl-twl-stgy-prog-wl-D-heur*
:: ‹*isasat* ⇒ *isasat nres*›
where
‹*cdcl-twl-stgy-prog-wl-D-heur* S_0 =
do {
do {
(*brk*, *T*) ← $WHILE_T$
(λ(*brk*, -). ¬*brk*)
(λ(*brk*, *S*).
do {
T ← *unit-propagation-outer-loop-wl-D-heur S*;

```
        cdcl-twl-o-prog-wl-D-heur T
      })
      (False, S_0);
    RETURN T
  }
 }
 ›
definition cdcl-twl-stgy-prog-break-wl-D-heur :: ‹isasat ⇒ isasat nres›
where
 ‹cdcl-twl-stgy-prog-break-wl-D-heur S_0 =
 do {
  b ← RETURN (isasat-fast S_0);
  (b, brk, T) ← WHILE_T^λ(b, brk, T). True
      (λ(b, brk, -). b ∧ ¬brk)
      (λ(b, brk, S).
      do {
        ASSERT(isasat-fast S);
        T ← unit-propagation-outer-loop-wl-D-heur S;
        ASSERT(isasat-fast T);
        (brk, T) ← cdcl-twl-o-prog-wl-D-heur T;
        b ← RETURN (isasat-fast T);
        RETURN(b, brk, T)
      })
      (b, False, S_0);
  if brk then RETURN T
  else cdcl-twl-stgy-prog-wl-D-heur T
 }›


definition cdcl-twl-stgy-prog-bounded-wl-heur :: ‹isasat ⇒ (bool × isasat) nres›
where
 ‹cdcl-twl-stgy-prog-bounded-wl-heur S_0 =
 do {
  b ← RETURN (isasat-fast S_0);
  (b, brk, T) ← WHILE_T^λ(b, brk, T). True
      (λ(b, brk, -). b ∧ ¬brk)
      (λ(b, brk, S).
      do {
        ASSERT(isasat-fast S);
        T ← unit-propagation-outer-loop-wl-D-heur S;
        ASSERT(isasat-fast T);
        (brk, T) ← cdcl-twl-o-prog-wl-D-heur T;
        b ← RETURN (isasat-fast T);
        RETURN(b, brk, T)
      })
      (b, False, S_0);
  RETURN (brk, T)
 }›

end
theory IsaSAT-Other
 imports IsaSAT-Conflict-Analysis IsaSAT-Backtrack IsaSAT-Decide
  IsaSAT-Other-Defs
begin
```

Chapter 21

Combining Together: the Other Rules

lemma *cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D*:
 ‹(*cdcl-twl-o-prog-wl-D-heur*, *cdcl-twl-o-prog-wl*) ∈
  {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur* ∧ *length* (*get-clauses-wl-heur S*) = *r* ∧ *learned-clss-count S* = *u*} →$_f$
   ⟨*bool-rel* ×$_f$ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur* ∧
     length (*get-clauses-wl-heur S*) ≤ *r* + *MAX-HEADER-SIZE*+1 + *unat32-max div 2* ∧
       learned-clss-count S ≤ *Suc u*}⟩*nres-rel*›
proof −
 have *H*: ‹(*x*, *y*) ∈ {(*S*, *T*).
          (*S*, *T*) ∈ *twl-st-heur* ∧
          length (*get-clauses-wl-heur S*) =
          length (*get-clauses-wl-heur x*)} ⟹
        (*x*, *y*)
        ∈ {(*S*, *T*).
          (*S*, *T*) ∈ *twl-st-heur-conflict-ana* ∧
          length (*get-clauses-wl-heur S*) =
          length (*get-clauses-wl-heur x*)}› **for** *x y*
  by (*auto simp*: *twl-st-heur-state-simp twl-st-heur-twl-st-heur-conflict-ana*)
 have *H2*: ‹(*x*, *y*) ∈ {(*S*, *T*).
  (*S*, *T*) ∈ *twl-st-heur* ∧
  length (*get-clauses-wl-heur S*) = *r* ∧ *learned-clss-count S* = *u*} ⟹
      (*x*, *y*) ∈ *twl-st-heur-conflict-ana*′ *r* (*get-learned-count x*)› **for** *x y*
  by (*auto simp*: *twl-st-heur-state-simp twl-st-heur-twl-st-heur-conflict-ana*)
 have *H3*: ‹(*x*, *y*)
  ∈ {(*S*, *T*).
  (*S*, *T*) ∈ *twl-st-heur* ∧
  length (*get-clauses-wl-heur S*) = *r* ∧ *learned-clss-count S* = *u*} ⟹
  (*x*, *y*) ∈ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur*‴ *r* ∧ *learned-clss-count S* = *u*}› **for** *x y*
  by *auto*
 have *UUa*: ‹(*U*, *Ua*)
    ∈ {(*S*, *T*).
      (*S*, *T*) ∈ *twl-st-heur* ∧
      length (*get-clauses-wl-heur S*) ≤ *3* + *1* + *r* + *unat32-max div 2* ∧
      learned-clss-count S ≤ *Suc u*} ⟹
    (*U*, *Ua*)
    ∈ {(*S*, *Tb*).
      (*S*, *Tb*) ∈ *twl-st-heur* ∧
  length (*get-clauses-wl-heur S*) ≤ *3* + *1* + *r* + *unat32-max div 2* ∧ *learned-clss-count S* ≤ *Suc u*}›
for *U Ua*

by *auto*
show *?thesis*
unfolding *cdcl-twl-o-prog-wl-D-heur-def cdcl-twl-o-prog-wl-def*
get-conflict-wl-is-None
apply (*intro frefI nres-relI*)
apply (*refine-vcg*
decide-wl-or-skip-D-heur-decide-wl-or-skip-D[**where** *r*=*r* **and** *u*=*u*, *THEN fref-to-Down*, *THEN order-trans*]
skip-and-resolve-loop-wl-D-heur-skip-and-resolve-loop-wl-D[**where** *r*=*r*, *THEN fref-to-Down*]
backtrack-wl-D-nlit-backtrack-wl-D[**where** *r*=*r* **and** *u*=*u*, *THEN fref-to-Down*]
isasat-current-status-id[*THEN fref-to-Down*, *THEN order-trans*])
subgoal
by (*auto simp*: *twl-st-heur-state-simp*
get-conflict-wl-is-None-heur-get-conflict-wl-is-None[*THEN fref-to-Down-unRET-Id*])
apply (*rule H3*; *assumption*)
subgoal by (*rule conc-fun-R-mono*) *auto*
subgoal by (*auto simp*: *twl-st-heur-state-simp twl-st-heur-count-decided-st-alt-def*)
apply (*rule H2*; *assumption*)
subgoal by (*auto simp*: *twl-st-heur-state-simp twl-st-heur-twl-st-heur-conflict-ana*)
subgoal by (*auto simp*: *twl-st-heur-state-simp*)
subgoal by (*auto simp*: *twl-st-heur-state-simp dest*!: *get-learned-count-learned-clss-countD2*)
apply (*rule UUa*; *assumption*)
subgoal by (*auto simp*: *conc-fun-RES RETURN-def learned-clss-count-def*)
subgoal by (*auto simp*: *twl-st-heur-state-simp*)
done
qed

lemma *cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D2*:
‹(*cdcl-twl-o-prog-wl-D-heur*, *cdcl-twl-o-prog-wl*) ∈
{(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur*} →$_f$
⟨*bool-rel* ×$_f$ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur*}⟩*nres-rel*›
apply (*intro frefI nres-relI*)
apply (*rule cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D*[*THEN fref-to-Down*, *THEN order-trans*])
apply (*auto intro*!: *conc-fun-R-mono*)
done

end
theory *IsaSAT-CDCL*
imports *IsaSAT-Propagate-Conflict IsaSAT-Other IsaSAT-Show*
IsaSAT-CDCL-Defs
begin

Combining Together: Full Strategy **lemma** *cdcl-twl-stgy-prog-wl-D-heur-cdcl-twl-stgy-prog-wl-D*:
‹(*cdcl-twl-stgy-prog-wl-D-heur*, *cdcl-twl-stgy-prog-wl*) ∈ *twl-st-heur* →$_f$ ⟨*twl-st-heur*⟩*nres-rel*›
proof −
have *H*[*refine0*]: ‹(*x*, *y*) ∈ {(*S*, *T*).
(*S*, *T*) ∈ *twl-st-heur* ∧
length (*get-clauses-wl-heur S*) =
length (*get-clauses-wl-heur x*)} ⟹
(*x*, *y*)
∈ {(*S*, *T*).
(*S*, *T*) ∈ *twl-st-heur* ∧ *get-learned-count S* = *get-learned-count x*}› **for** *x y*
by (*auto simp*: *twl-st-heur-state-simp twl-st-heur-twl-st-heur-conflict-ana*)
show *?thesis*
unfolding *cdcl-twl-stgy-prog-wl-D-heur-def cdcl-twl-stgy-prog-wl-def*
apply (*intro frefI nres-relI*)

subgoal for *x y*
apply (*refine-vcg*
unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D'[*THEN twl-st-heur''D-twl-st-heurD*, *THEN fref-to-Down*]
cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D2[*THEN fref-to-Down*])
subgoal by (*auto simp*: *twl-st-heur-state-simp*)
subgoal by (*auto simp*: *twl-st-heur-state-simp twl-st-heur'-def*)
subgoal by (*auto simp*: *twl-st-heur'-def*)
subgoal by (*auto simp*: *twl-st-heur-state-simp*)
subgoal by (*auto simp*: *twl-st-heur-state-simp*)
done
done
qed

lemma *cdcl-twl-stgy-prog-early-wl-heur-cdcl-twl-stgy-prog-early-wl-D*:
assumes *r*: ‹*r* ≤ *snat64-max*›
shows ‹(*cdcl-twl-stgy-prog-bounded-wl-heur*, *cdcl-twl-stgy-prog-early-wl*) ∈
{(*S*,*T*). (*S*, *T*) ∈ *twl-st-heur''' r*} →$_f$
⟨*bool-rel* ×$_r$ *twl-st-heur*⟩*nres-rel*›
proof −
have *A*[*refine0*]: ‹*RETURN* (*isasat-fast x*) ≤ ⇓
{(*b*, *b'*). *b* = *b'* ∧ (*b* = (*isasat-fast x*))} (*RES UNIV*)›
for *x*
by (*auto intro*: *RETURN-RES-refine*)
have *twl-st-heur''*: ‹(*x1e*, *x1b*) ∈ *twl-st-heur* ⟹
(*x1e*, *x1b*)
∈ *twl-st-heur''*
(*dom-m* (*get-clauses-wl x1b*))
(*length* (*get-clauses-wl-heur x1e*))
(*get-learned-count x1e*)›
for *x1e x1b*
by (*auto simp*: *twl-st-heur'-def*)
have *twl-st-heur'''*: ‹(*x1e*, *x1b*) ∈ *twl-st-heur''* 𝒟 *r lcount* ⟹
(*x1e*, *x1b*)
∈ {(*S*, *Taa*).
(*S*, *Taa*) ∈ *twl-st-heur* ∧
length (*get-clauses-wl-heur S*) = *r* ∧
learned-clss-count S = (λ(*a*,*b*,*c*,*d*,*e*). *a* + *b* + *c* + *d*+*e*) (*lcount*)}›
for *x1e x1b r* 𝒟 *lcount*
by (*auto simp*: *twl-st-heur'-def learned-clss-count-def clss-size-lcountUEk-def*
clss-size-lcountUE-def clss-size-lcountU0-def clss-size-lcount-def clss-size-lcountUS-def
split: *prod.splits*)
have *H*: ‹*SPEC* (λ-::*bool*. *True*) = *RES UNIV*› **by** *auto*
show *?thesis*
supply[[*goals-limit=1*]] *isasat-fast-length-leD*[*dest*] *twl-st-heur'-def*[*simp*]
unfolding *cdcl-twl-stgy-prog-bounded-wl-heur-def*
cdcl-twl-stgy-prog-early-wl-def H
apply (*intro frefI nres-relI*)
apply (*refine-rcg*
cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D[*THEN fref-to-Down*]
unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D'[*THEN fref-to-Down*]
WHILEIT-refine[**where** *R* = ‹{((*ebrk*, *brk*, *T*), (*ebrk'*, *brk'*, *T'*)).
(*ebrk* = *ebrk'*) ∧ (*brk* = *brk'*) ∧ (*T*, *T'*) ∈ *twl-st-heur* ∧
(*ebrk* ⟶ *isasat-fast T*) ∧ (*length* (*get-clauses-wl-heur T*) ≤ *snat64-max*)}›])

subgoal using *r* **by** *auto*
subgoal by *fast*
subgoal by *auto*
apply (*rule twl-st-heur''*; *auto*; *fail*)
subgoal by (*auto simp*: *isasat-fast-def dest*: *get-learned-count-learned-clss-countD*)
apply (*rule twl-st-heur'''*; *assumption*)
subgoal
apply *clarsimp*
by (*auto simp*: *isasat-fast-def snat64-max-def unat32-max-def*
dest:)
subgoal by *auto*
done
qed

end
theory *IsaSAT-Other-LLVM*
imports *IsaSAT-Other-Defs IsaSAT-Conflict-Analysis-LLVM IsaSAT-Backtrack-LLVM IsaSAT-Decide-LLVM*
begin

sepref-register *get-conflict-wl-is-None decide-wl-or-skip-D-heur skip-and-resolve-loop-wl-D-heur*
backtrack-wl-D-nlit-heur isasat-current-status count-decided-st-heur get-conflict-wl-is-None-heur

lemma *cdcl-twl-o-prog-wl-D-heurI1*:
‹*get-learned-count x* = *get-learned-count xc* ⟹
learned-clss-count x < *unat64-max* ⟹ *learned-clss-count xc* ≤ *unat64-max*›
using *get-learned-count-learned-clss-countD2*[*of xc x*]
by (*auto dest*: *get-learned-count-learned-clss-countD2*)

lemma *cdcl-twl-o-prog-wl-D-heurI*:
‹ *get-learned-count x* = *get-learned-count xc* ⟹
learned-clss-count x < *unat64-max* ⟹ *learned-clss-count xc* < *unat64-max*›
using *get-learned-count-learned-clss-countD2*[*of xc x*]
by *auto*

sepref-def *cdcl-twl-o-prog-wl-D-fast-code*
is ‹*cdcl-twl-o-prog-wl-D-heur*›
:: ‹[*isasat-fast*]$_a$
isasat-bounded-assn^d → *bool1-assn* ×$_a$ *isasat-bounded-assn*›
unfolding *cdcl-twl-o-prog-wl-D-heur-def PR-CONST-def*
unfolding *get-conflict-wl-is-None get-conflict-wl-is-None-heur-alt-def*[*symmetric*]
supply [[*goals-limit* = *1*]] *isasat-fast-def*[*simp*] *cdcl-twl-o-prog-wl-D-heurI*[*intro*]
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

declare
cdcl-twl-o-prog-wl-D-fast-code.refine[*sepref-fr-rules*]

sepref-register
cdcl-twl-o-prog-wl-D-heur

lemma *isasat-fast-alt-def*: ‹*isasat-fast S* = (*length-clauses-heur S* ≤ *9223372034707292156* ∧
clss-size-lcount (*get-learned-count S*) < *18446744073709551615* − *clss-size-lcountUE* (*get-learned-count*
S) ∧
clss-size-lcount (*get-learned-count S*) + *clss-size-lcountUE* (*get-learned-count S*) < *18446744073709551615*
− *clss-size-lcountUS* (*get-learned-count S*) ∧

clss-size-lcount (*get-learned-count S*) +
clss-size-lcountUE (*get-learned-count S*) + *clss-size-lcountUS* (*get-learned-count S*) < *18446744073709551615* − *clss-size-lcountU0* (*get-learned-count S*) ∧
clss-size-lcount (*get-learned-count S*) +
clss-size-lcountUE (*get-learned-count S*) + *clss-size-lcountUS* (*get-learned-count S*) + *clss-size-lcountU0* (*get-learned-count S*) < *18446744073709551615* − *clss-size-lcountUEk* (*get-learned-count S*)∧
clss-size-lcount (*get-learned-count S*) +
clss-size-lcountUE (*get-learned-count S*) + *clss-size-lcountUS* (*get-learned-count S*) + *clss-size-lcountU0* (*get-learned-count S*) + *clss-size-lcountUEk* (*get-learned-count S*) < *18446744073709551615*)›
by (*cases S*; *auto simp*: *isasat-fast-def snat64-max-def unat32-max-def length-clauses-heur-def unat64-max-def learned-clss-count-def clss-size-lcountU0-def*)

sepref-def *isasat-fast-impl*
is ‹*RETURN o isasat-fast*›
:: ‹*isasat-bounded-assn*^{k} →$_a$ *bool1-assn*›
unfolding *isasat-fast-alt-def short-circuit-conv*
apply (*rewrite at* ‹- < ¤› *unat-const-fold*[**where** *'a=64*])+
apply (*rewrite at* ‹- < ¤ − -› *unat-const-fold*[**where** *'a=64*])+
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

end
theory *IsaSAT-CDCL-LLVM*
imports *IsaSAT-CDCL-Defs IsaSAT-Propagate-Conflict-LLVM IsaSAT-Other-LLVM*
begin

sepref-def *cdcl-twl-stgy-prog-wl-D-code*
is ‹*cdcl-twl-stgy-prog-bounded-wl-heur*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *bool1-assn* ×$_a$ *isasat-bounded-assn*›
unfolding *cdcl-twl-stgy-prog-bounded-wl-heur-def PR-CONST-def*
supply [[*goals-limit* = *1*]] *isasat-fast-length-leD*[*dest*]
by *sepref*

declare *cdcl-twl-stgy-prog-wl-D-code.refine*[*sepref-fr-rules*]

export-llvm *cdcl-twl-stgy-prog-wl-D-code* **file** ‹*code/isasat.ll*›

end
theory *IsaSAT-Restart-Defs*
imports
Watched-Literals.WB-Sort Watched-Literals.Watched-Literals-Watch-List-Simp IsaSAT-Rephase-State IsaSAT-Setup IsaSAT-VMTF IsaSAT-Sorting IsaSAT-Proofs
begin

lemma *unbounded-id*: ‹*unbounded* (*id* :: *nat* ⇒ *nat*)›
by (*auto simp*: *bounded-def*) *presburger*

global-interpretation *twl-restart-ops id*
by *unfold-locales*

global-interpretation *twl-restart id*
by *standard* (*rule unbounded-id*)

definition *twl-st-heur-restart* :: ‹(*isasat* × *nat twl-st-wl*) *set*› **where**

[*unfolded Let-def*]: ‹*twl-st-heur-restart* =
  {(*S*,*T*).
  let M′ = *get-trail-wl-heur S*; *N′* = *get-clauses-wl-heur S*; *D′* = *get-conflict-wl-heur S*;
    W′ = *get-watched-wl-heur S*; *j* = *literals-to-update-wl-heur S*; *outl* = *get-outlearned-heur S*;
    cach = *get-conflict-cach S*; *clvls* = *get-count-max-lvls-heur S*;
    vm = *get-vmtf-heur S*;
    vdom = *get-aivdom S*; *heur* = *get-heur S*; *old-arena* = *get-old-arena S*;
    lcount = *get-learned-count S*; *occs* = *get-occs S in*
    let M = *get-trail-wl T*; *N* = *get-clauses-wl T*; *D* = *get-conflict-wl T*;
      Q = *literals-to-update-wl T*;
      W = *get-watched-wl T*; *N0* = *get-init-clauses0-wl T*; *U0* = *get-learned-clauses0-wl T*;
      NS = *get-subsumed-init-clauses-wl T*; *US* = *get-subsumed-learned-clauses-wl T*;
      NEk = *get-kept-unit-init-clss-wl T*; *UEk* = *get-kept-unit-learned-clss-wl T*;
      NE = *get-unkept-unit-init-clss-wl T*; *UE* = *get-unkept-unit-learned-clss-wl T in*
    (*M′*, *M*) ∈ *trail-pol* (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) ∧
    valid-arena N′ N (*set* (*get-vdom-aivdom vdom*)) ∧
    (*D′*, *D*) ∈ *option-lookup-clause-rel* (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) ∧
    (*D* = *None* ⟶ *j* ≤ *length M*) ∧
    Q = *uminus* ‘# *lit-of* ‘# *mset* (*drop j* (*rev M*)) ∧
    (*W′*, *W*) ∈ ⟨*Id*⟩*map-fun-rel* (*D*$_0$ (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*))) ∧
    vm ∈ *bump-heur* (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) *M* ∧
    no-dup M ∧
    clvls ∈ *counts-maximum-level M D* ∧
    cach-refinement-empty (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) *cach* ∧
    out-learned M D outl ∧
    clss-size-corr-restart N NE {#} *NEk UEk NS* {#} *N0* {#} *lcount* ∧
    vdom-m (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) *W N* ⊆ *set* (*get-vdom-aivdom vdom*) ∧
    aivdom-inv-dec vdom (*dom-m N*) ∧
    isasat-input-bounded (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) ∧
    isasat-input-nempty (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) ∧
    old-arena = [] ∧
      heuristic-rel (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) *heur* ∧
    (*occs*, *empty-occs-list* (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*))) ∈ *occurrence-list-ref*
  }›

abbreviation *twl-st-heur′′′′* **where**
  ‹*twl-st-heur′′′′ r* ≡ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur* ∧ *length* (*get-clauses-wl-heur S*) ≤ *r*}›

abbreviation *twl-st-heur′′′′uu* **where**
  ‹*twl-st-heur′′′′uu r u* ≡ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur* ∧ *length* (*get-clauses-wl-heur S*) ≤ *r* ∧
    learned-clss-count S ≤ *u*}›

abbreviation *twl-st-heur-restart′′′* **where**
  ‹*twl-st-heur-restart′′′ r* ≡
    {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart* ∧ *length* (*get-clauses-wl-heur S*) = *r*}›

abbreviation *twl-st-heur-restart′′′′* **where**
  ‹*twl-st-heur-restart′′′′ r* ≡
    {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart* ∧ *length* (*get-clauses-wl-heur S*) ≤ *r*}›

definition *twl-st-heur-restart-ana* :: ‹*nat* ⇒ (*isasat* × *nat twl-st-wl*) *set*› **where**
‹*twl-st-heur-restart-ana r* =
  {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart* ∧ *length* (*get-clauses-wl-heur S*) = *r*}›

abbreviation *twl-st-heur-restart-ana′* :: ‹-› **where**

‹*twl-st-heur-restart-ana′ r u* ≡
{(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart-ana r* ∧ *learned-clss-count S* ≤ *u*}›

definition *empty-Q* :: ‹*isasat* ⇒ *isasat nres*› **where**
‹*empty-Q* = (λ*S*. *do*{
j ← *mop-isa-length-trail* (*get-trail-wl-heur S*);
RETURN (*set-heur-wl-heur* (*restart-info-restart-done-heur* (*get-heur S*)) (*set-literals-to-update-wl-heur j S*))
})›

definition *remove-all-annot-true-clause-one-imp-heur*
:: ‹*nat* × *clss-size* × *arena* ⇒ (*clss-size* × *arena*) *nres*›
where
‹*remove-all-annot-true-clause-one-imp-heur* = (λ(*C*, *j*, *N*). *do* {
case arena-status N C of
DELETED ⇒ *RETURN* (*j*, *N*)
| *IRRED* ⇒ *RETURN* (*j*, *extra-information-mark-to-delete N C*)
| *LEARNED* ⇒ *RETURN* (*clss-size-decr-lcount j*, *extra-information-mark-to-delete N C*)
})›

definition *number-clss-to-keep* :: ‹*isasat* ⇒ *nat nres*› **where**
‹*number-clss-to-keep* = (λ*S*.
RES UNIV)›

definition *number-clss-to-keep-impl* :: ‹*isasat* ⇒ *nat nres*› **where**
‹*number-clss-to-keep-impl* = (λ*S*.
RETURN (*length-tvdom-aivdom* (*get-aivdom S*) >> *2*))›

definition (**in** −) *MINIMUM-DELETION-LBD* :: *nat* **where**
‹*MINIMUM-DELETION-LBD* = *3*›

definition *trail-update-reason-at* :: ‹- ⇒ - ⇒ *trail-pol* ⇒ -› **where**
‹*trail-update-reason-at* ≡ (λ*L C* (*M*, *val*, *lvls*, *reason*, *k*). (*M*, *val*, *lvls*, *reason*[*atm-of L* := *C*], *k*))›

abbreviation *trail-get-reason* :: ‹*trail-pol* ⇒ -› **where**
‹*trail-get-reason* ≡ (λ(*M*, *val*, *lvls*, *reason*, *k*). *reason*)›

definition *replace-reason-in-trail* :: ‹*nat literal* ⇒ -› **where**
‹*replace-reason-in-trail L C* = (λ*M*. *do* {
ASSERT(*atm-of L* < *length* (*trail-get-reason M*));
RETURN (*trail-update-reason-at L 0 M*)
})›

definition *isasat-replace-annot-in-trail*
:: ‹*nat literal* ⇒ *nat* ⇒ *isasat* ⇒ *isasat nres*›
where
‹*isasat-replace-annot-in-trail L C* = (λ*S*. *do* {
let lcount = *clss-size-resetUS0* (*get-learned-count S*);
M ← *replace-reason-in-trail L C* (*get-trail-wl-heur S*);
RETURN (*set-trail-wl-heur M* (*set-learned-count-wl-heur lcount S*))
})›

definition *remove-one-annot-true-clause-one-imp-wl-D-heur*
:: ‹*nat* ⇒ *isasat* ⇒ (*nat* × *isasat*) *nres*›
where

‹*remove-one-annot-true-clause-one-imp-wl-D-heur* = (λ*i* S_0. *do* {
(*L*, *C*) ← *do* {
L ← *isa-trail-nth* (*get-trail-wl-heur* S_0) *i*;
C ← *get-the-propagation-reason-pol* (*get-trail-wl-heur* S_0) *L*;
RETURN (*L*, *C*)};
ASSERT(*C* ≠ *None* ∧ *i* + *1* ≤ *Suc* (*unat32-max div 2*));
if the C = *0 then RETURN* (*i*+*1*, S_0)
else do {
ASSERT(*C* ≠ *None*);
S ← *isasat-replace-annot-in-trail L* (*the C*) S_0;
log-del-clause-heur S (*the C*);
ASSERT(*mark-garbage-pre* (*get-clauses-wl-heur S*, *the C*) ∧ *arena-is-valid-clause-vdom* (*get-clauses-wl-heur S*) (*the C*) ∧ *learned-clss-count S* ≤ *learned-clss-count* S_0);
S ← *mark-garbage-heur4* (*the C*) *S*;
— *S* ← *remove-all-annot-true-clause-imp-wl-D-heur L S*;
RETURN (*i*+*1*, *S*)
}
})›

definition *cdcl-twl-full-restart-wl-D-GC-prog-heur-post* :: ‹*isasat* ⇒ *isasat* ⇒ *bool*› **where**
‹*cdcl-twl-full-restart-wl-D-GC-prog-heur-post S T* ⟷
(∃ *S*′ *T*′. (*S*, *S*′) ∈ *twl-st-heur-restart* ∧ (*T*, *T*′) ∈ *twl-st-heur-restart* ∧
cdcl-twl-full-restart-wl-GC-prog-post S′ *T*′)›

definition *remove-one-annot-true-clause-imp-wl-D-heur-inv*
:: ‹*isasat* ⇒ (*nat* × *isasat*) ⇒ *bool*› **where**
‹*remove-one-annot-true-clause-imp-wl-D-heur-inv S* = (λ(*i*, *T*).
(∃ *S*′ *T*′. (*S*, *S*′) ∈ *twl-st-heur-restart* ∧ (*T*, *T*′) ∈ *twl-st-heur-restart* ∧
remove-one-annot-true-clause-imp-wl-inv S′ (*i*, *T*′) ∧
learned-clss-count T ≤ *learned-clss-count S*))›

definition *empty-US* :: ‹′*v twl-st-wl* ⇒ ′*v twl-st-wl*› **where**
‹*empty-US* = (λ(*M*′, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*).
(*M*′, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*))›

definition *trail-zeroed-until-state* **where**
‹*trail-zeroed-until-state S* = *trail-zeroed-until* (*get-trail-wl-heur S*)›

definition *trail-set-zeroed-until-state* **where**
‹*trail-set-zeroed-until-state z S* = (*let M* = *get-trail-wl-heur S in set-trail-wl-heur* (*trail-set-zeroed-until z M*) *S*)›

definition *remove-one-annot-true-clause-imp-wl-D-heur* :: ‹*isasat* ⇒ *isasat nres*›
where
‹*remove-one-annot-true-clause-imp-wl-D-heur* = (λ*S*. *do* {
ASSERT((*isa-length-trail-pre o get-trail-wl-heur*) *S*);
k ← (*if count-decided-st-heur S* = *0*
then RETURN (*isa-length-trail* (*get-trail-wl-heur S*))
else get-pos-of-level-in-trail-imp (*get-trail-wl-heur S*) *0*);
let start = *trail-zeroed-until-state S*;
(*i*, *T*) ← *WHILE*$_T$[*remove-one-annot-true-clause-imp-wl-D-heur-inv S*]
(λ(*i*, *S*). *i* < *k*)
(λ(*i*, *S*). *do* { (*j*, *S*) ← *remove-one-annot-true-clause-one-imp-wl-D-heur i S*; *RETURN* (*j*, *trail-set-zeroed-until-state j S*)})
(*start*, *S*);

```
    ASSERT (remove-one-annot-true-clause-imp-wl-D-heur-inv S (i, T));
    let T = trail-set-zeroed-until-state i T;
    RETURN (empty-US-heur T)
  })›

definition GC-units-required :: ‹isasat ⇒ bool› where
  ‹GC-units-required T ⟷ units-since-last-GC-st T ≥ get-GC-units-opt T›


definition FLAG-no-restart :: ‹8 word› where
  ‹FLAG-no-restart = 0›

definition FLAG-restart :: ‹8 word› where
  ‹FLAG-restart = 1›

definition FLAG-Reduce-restart :: ‹8 word› where
  ‹FLAG-Reduce-restart = 3›

definition FLAG-GC-restart :: ‹8 word› where
  ‹FLAG-GC-restart = 2›

definition FLAG-Inprocess-restart :: ‹8 word› where
  ‹FLAG-Inprocess-restart = 4›


definition max-restart-decision-lvl :: nat where
  ‹max-restart-decision-lvl = 300›

definition max-restart-decision-lvl-code :: ‹32 word› where
  ‹max-restart-decision-lvl-code = 300›

definition GC-required-heur :: ‹isasat ⇒ nat ⇒ bool nres› where
  ‹GC-required-heur S n = do {
    n ← RETURN (full-arena-length-st S);
    wasted ← RETURN (wasted-bytes-st S);
    RETURN (3*wasted > ((of-nat n)>>2))
  }›

definition minimum-number-between-restarts :: ‹64 word› where
  ‹minimum-number-between-restarts = 50›

definition upper-restart-bound-reached :: ‹isasat ⇒ bool› where
  ‹upper-restart-bound-reached = (λS. get-global-conflict-count S ≥ next-reduce-schedule-st S)›

definition should-subsume-st :: ‹isasat ⇒ bool› where
  ‹should-subsume-st S ⟷ get-subsumption-opts S ∧
      (get-global-conflict-count S > next-subsume-schedule-st S)›

definition should-eliminate-pure-st :: ‹isasat ⇒ bool› where
  ‹should-eliminate-pure-st S ⟷
      (get-global-conflict-count S > next-pure-lits-schedule-st S)›

definition should-inprocess-st :: ‹isasat ⇒ bool› where
  ‹should-inprocess-st S ⟷
      (should-subsume-st S ∨ should-eliminate-pure-st S)›
```

definition *iterate-over-VMTFC* **where**
‹*iterate-over-VMTFC* = (λ*f* (*I* :: 'a ⇒ *bool*) *P* (*ns* :: (*nat*, *nat*) *vmtf-node list*, *n*) *x*. *do* {
(-, *x*) ← *WHILE*$_T^{\lambda(n,\ x).\ I\ x}$
(λ(*n*, *x*). *n* ≠ *None* ∧ *P x*)
(λ(*n*, *x*). *do* {
ASSERT(*n* ≠ *None*);
let A = *the n*;
ASSERT(*A* < *length ns*);
ASSERT(*A* ≤ *unat32-max div 2*);
x ← *f A x*;
RETURN (*get-next* ((*ns* ! *A*)), *x*)
})
(*n*, *x*);
RETURN x
})›

definition *iterate-over-VMTF* :: ‹-› **where**
iterate-over-VMTF-alt-def: ‹*iterate-over-VMTF f I* = *iterate-over-VMTFC f I* (λ-. *True*)›

definition *arena-header-size* :: ‹*arena* ⇒ *nat* ⇒ *nat*› **where**
‹*arena-header-size arena C* =
(*if arena-length arena C* > *4 then MAX-HEADER-SIZE else MIN-HEADER-SIZE*)›

definition *update-restart-phases* :: ‹*isasat* ⇒ *isasat nres*› **where**
‹*update-restart-phases* = (λ*S*. *do* {
let heur = *get-heur S*;
let lcount = *get-global-conflict-count S*;
let vm = *get-vmtf-heur S*;
let vm = *switch-bump-heur vm*;
heur ← *RETURN* (*incr-restart-phase heur*);
heur ← *RETURN* (*incr-restart-phase-end lcount heur*);
heur ← *RETURN* (*if current-restart-phase heur* = *STABLE-MODE then heuristic-reluctant-enable heur else heuristic-reluctant-disable heur*);
heur ← *RETURN* (*swap-emas heur*);
RETURN (*set-heur-wl-heur heur* (*set-vmtf-wl-heur vm S*))
})›

definition *restart-abs-wl-heur-pre* :: ‹*isasat* ⇒ *bool* ⇒ *bool*› **where**
‹*restart-abs-wl-heur-pre S brk* ⟷ (∃ *T last-GC last-Restart*. (*S*, *T*) ∈ *twl-st-heur* ∧ *restart-abs-wl-pre T last-GC last-Restart brk*)›

lemma *valid-arena-header-size*:
‹*valid-arena arena N vdom* ⟹ *C* ∈# *dom-m N* ⟹ *arena-header-size arena C* = *header-size* (*N* ∝ *C*)›
by (*auto simp*: *arena-header-size-def header-size-def arena-lifting*)

definition *rewatch-heur-st-pre* :: ‹*isasat* ⇒ *bool*› **where**
‹*rewatch-heur-st-pre S* ⟷ (∀ *i* < *length* (*get-tvdom S*). *get-tvdom S* ! *i* ≤ *snat64-max*)›

lemma *isasat-GC-clauses-wl-D-rewatch-pre*:
assumes
‹*length* (*get-clauses-wl-heur x*) ≤ *snat64-max*› **and**
‹*length* (*get-clauses-wl-heur xc*) ≤ *length* (*get-clauses-wl-heur x*)› **and**

```
    ‹∀ i ∈ set (get-tvdom xc). i ≤ length (get-clauses-wl-heur x)›
  shows ‹rewatch-heur-st-pre xc›
  using assms
  unfolding rewatch-heur-st-pre-def all-set-conv-all-nth
  by auto

end
theory IsaSAT-Restart-Reduce-Defs
  imports IsaSAT-Restart-Defs
    IsaSAT-Bump-Heuristics
begin
```

We first fix the function that proves termination. We don't take the "smallest" function possible (other possibilites that are growing slower include $\lambda n.\ n >> 50$).

```
definition (in −) find-local-restart-target-level-int-inv where
  ‹find-local-restart-target-level-int-inv bmp cs =
     (λ(brk, i). i ≤ length cs ∧ length cs < unat32-max)›


definition find-local-restart-target-level-int
   :: ‹trail-pol ⇒ bump-heuristics ⇒ nat nres›
where
  ‹find-local-restart-target-level-int =
     (λ(M, xs, lvls, reasons, k, cs, zeored) bmp. do {
     let m = current-vmtf-array-nxt-score bmp;
     (brk, i) ← WHILE_T^find-local-restart-target-level-int-inv bmp cs
        (λ(brk, i). ¬brk ∧ i < length cs)
        (λ(brk, i). do {
           ASSERT (i < length cs);
           let t = (cs ! i);
    ASSERT(t < length M);
    let L = atm-of (M ! t);
           u ← access-focused-vmtf-array bmp L;
           let brk = stamp u < m;
           RETURN (brk, if brk then i else i+1)
         })
        (False, 0);
    RETURN i
   })›
```

find-decomp-wl-st-int is the wrong function here, because unlike in the backtrack case, we also have to update the queue of literals to update. This is done in the function *empty-Q*.

```
definition find-local-restart-target-level-st :: ‹isasat ⇒ nat nres› where
  ‹find-local-restart-target-level-st S = do {
    find-local-restart-target-level-int (get-trail-wl-heur S) (get-vmtf-heur S)
  }›

definition cdcl-twl-local-restart-wl-D-heur
   :: ‹isasat ⇒ isasat nres›
where
  ‹cdcl-twl-local-restart-wl-D-heur = (λS. do {
      ASSERT(restart-abs-wl-heur-pre S False);
      lvl ← find-local-restart-target-level-st S;
      b ← RETURN (lvl = count-decided-st-heur S);
      if b
```

```
      then RETURN S
      else do {
        S ← find-decomp-wl-st-int lvl S;
        S ← empty-Q S;
        incr-restart-stat S
      }
    }
  )›

definition reorder-vdom-wl :: ‹'v twl-st-wl ⇒ 'v twl-st-wl nres› where
  ‹reorder-vdom-wl S = do {
    ASSERT (mark-to-delete-clauses-wl-pre S);
    RETURN S
  }›

definition sort-clauses-by-score :: ‹arena ⇒ isasat-aivdom ⇒ isasat-aivdom nres› where
  ‹sort-clauses-by-score arena vdom = do {
      ASSERT(∀ i∈set (get-vdom-aivdom vdom). valid-sort-clause-score-pre-at arena i);
      ASSERT(∀ i∈set (get-avdom-aivdom vdom). valid-sort-clause-score-pre-at arena i);
      ASSERT(∀ i∈set (get-tvdom-aivdom vdom). valid-sort-clause-score-pre-at arena i);
    SPEC(λvdom'. mset (get-vdom-aivdom vdom) = mset (get-vdom-aivdom vdom') ∧
      mset (get-avdom-aivdom vdom) = mset (get-avdom-aivdom vdom') ∧
      mset (get-ivdom-aivdom vdom) = mset (get-ivdom-aivdom vdom') ∧
      mset (get-tvdom-aivdom vdom) = mset (get-tvdom-aivdom vdom'))
  }›

definition (in −) quicksort-clauses-by-score-avdom :: ‹arena ⇒ vdom ⇒ vdom nres› where
  ‹quicksort-clauses-by-score-avdom arena =
     (full-quicksort-ref clause-score-ordering (clause-score-extract arena))›

definition remove-deleted-clauses-from-avdom-inv :: ‹- › where
 ‹remove-deleted-clauses-from-avdom-inv N avdom0 = (λ(i, j, avdom). i ≤ j ∧ j ≤ length (get-avdom-aivdom
avdom0) ∧ length (get-avdom-aivdom avdom) = length (get-avdom-aivdom avdom0) ∧
   mset (take i (get-avdom-aivdom avdom) @ drop j (get-avdom-aivdom avdom)) ⊆# mset (get-avdom-aivdom
avdom0) ∧
    mset (take i (get-avdom-aivdom avdom) @ drop j (get-avdom-aivdom avdom)) ∩# dom-m N = mset
(get-avdom-aivdom avdom0) ∩# dom-m N ∧
    get-vdom-aivdom avdom = get-vdom-aivdom avdom0 ∧
    get-ivdom-aivdom avdom = get-ivdom-aivdom avdom0 ∧
    distinct (get-tvdom-aivdom avdom) ∧
  set (get-tvdom-aivdom avdom) ⊆ set (take i (get-avdom-aivdom avdom)) ∧
  length (get-tvdom-aivdom avdom) ≤ i ∧
   (∀ C ∈ set (get-tvdom-aivdom avdom). C ∈# dom-m N ∧ ¬irred N C ∧ length (N ∝ C) ≠ 2))›

definition is-candidate-for-removal where
  ‹is-candidate-for-removal C N = do {
    ASSERT (C ∈# dom-m N);
    SPEC (λb :: bool. b ⟶ ¬irred N C ∧ length (N ∝ C) ≠ 2)
  }›

definition remove-deleted-clauses-from-avdom :: ‹- › where
‹remove-deleted-clauses-from-avdom N avdom0 = do {
  let n = length (get-avdom-aivdom avdom0);
  let avdom0' = get-avdom-aivdom avdom0;
  (i, j, avdom) ← WHILE_T remove-deleted-clauses-from-avdom-inv N avdom0
    (λ(i, j, avdom). j < n)
```

```
    (λ(i, j, avdom). do {
      ASSERT(j < length (get-avdom-aivdom avdom));
      if (get-avdom-aivdom avdom ! j) ∈# dom-m N then do {
        let C = get-avdom-aivdom avdom ! j;
        let avdom = swap-avdom-aivdom avdom i j;
        should-push ← is-candidate-for-removal C N;
        if should-push then RETURN (i+1, j+1, push-to-tvdom C avdom)
        else RETURN (i+1, j+1, avdom)
      }
     else RETURN (i, j+1, avdom)
    })
    (0, 0, empty-tvdom avdom0);
  ASSERT(i ≤ length (get-avdom-aivdom avdom));
  RETURN (take-avdom-aivdom i avdom)
}›


definition isa-is-candidate-for-removal where
  ‹isa-is-candidate-for-removal M C arena = do {
    ASSERT(arena-act-pre arena C);
    L ← mop-arena-lit arena C;
    lbd ← mop-arena-lbd arena C;
    length ← mop-arena-length arena C;
    status ← mop-arena-status arena C;
    used ← mop-marked-as-used arena C;
    D ← get-the-propagation-reason-pol M L;
    let can-del =
      lbd > MINIMUM-DELETION-LBD ∧
      status = LEARNED ∧
      length ≠ 2 ∧
      used = 0 ∧
      (D ≠ Some C);
    RETURN can-del
  }›


definition isa-gather-candidates-for-reduction :: ‹trail-pol ⇒ arena ⇒ - ⇒ (arena × -) nres› where
‹isa-gather-candidates-for-reduction M arena avdom0 = do {
  ASSERT(length (get-avdom-aivdom avdom0) ≤ length arena);
  ASSERT(length (get-avdom-aivdom avdom0) ≤ length (get-vdom-aivdom avdom0));
  let n = length (get-avdom-aivdom avdom0);
  (arena, i, j, avdom) ← WHILE_T λ(-, i, j, -). i ≤ j ∧ j ≤ n
    (λ(arena, i, j, avdom). j < n)
    (λ(arena, i, j, avdom). do {
      ASSERT(j < n);
    ASSERT(arena-is-valid-clause-vdom arena (get-avdom-aivdom avdom!j) ∧ j < length (get-avdom-aivdom
avdom) ∧ i < length (get-avdom-aivdom avdom));
     ASSERT (length (get-tvdom-aivdom avdom) ≤ i);
     if arena-status arena (get-avdom-aivdom avdom ! j) ≠ DELETED then do{
        ASSERT(arena-act-pre arena (get-avdom-aivdom avdom ! j));
        should-push ← isa-is-candidate-for-removal M (get-avdom-aivdom avdom ! j) arena;
        let arena = mark-unused arena (get-avdom-aivdom avdom ! j);
        if should-push then  RETURN (arena, i+1, j+1, push-to-tvdom (get-avdom-aivdom avdom ! j)
(swap-avdom-aivdom avdom i j))
       else RETURN (arena, i+1, j+1, swap-avdom-aivdom avdom i j)
      }
```

```
    else RETURN (arena, i, j+1, avdom)
   }) (arena, 0, 0, empty-tvdom avdom0);
  ASSERT(i ≤ length (get-avdom-aivdom avdom));
  RETURN (arena, take-avdom-aivdom i avdom)
}›

definition (in −) sort-vdom-heur :: ‹isasat ⇒ isasat nres› where
  ‹sort-vdom-heur = (λS. do {
    let vdom = get-aivdom S;
    let M' = get-trail-wl-heur S;
    let arena = get-clauses-wl-heur S;
    ASSERT(length (get-avdom-aivdom vdom) ≤ length arena);
    ASSERT(length (get-vdom-aivdom vdom) ≤ length arena);
    (arena', vdom) ← isa-gather-candidates-for-reduction M' arena vdom;
    ASSERT(valid-sort-clause-score-pre arena (get-vdom-aivdom vdom));
    ASSERT(EQ (length arena) (length arena'));
    ASSERT(length (get-avdom-aivdom vdom) ≤ length arena);
    vdom ← sort-clauses-by-score arena' vdom;
    RETURN (set-clauses-wl-heur arena' (set-aivdom-wl-heur vdom S))
    })›

definition partition-main-clause where
  ‹partition-main-clause arena = partition-main clause-score-ordering (clause-score-extract arena)›

definition partition-clause where
  ‹partition-clause arena = partition-between-ref clause-score-ordering  (clause-score-extract arena)›


definition mark-to-delete-clauses-wl-D-heur-pre :: ‹isasat ⇒ bool› where
  ‹mark-to-delete-clauses-wl-D-heur-pre S ⟷
  (∃ S'. (S, S') ∈ twl-st-heur-restart ∧ mark-to-delete-clauses-wl-pre S')›

definition find-largest-lbd-and-size
  :: ‹nat ⇒ isasat ⇒ (nat × nat) nres›
where
‹find-largest-lbd-and-size  = (λl S. do {
    ASSERT(length (get-tvdom S) ≤ length (get-clauses-wl-heur S));
    (i, lbd, sze) ← WHILE_T^λ(i, - :: nat, -::nat). i ≤ length (get-tvdom S)
      (λ(i, lbd, sze). i < l ∧ i < length (get-tvdom S))
      (λ(i, lbd, sze). do {
        ASSERT(i < length (get-tvdom S));
        ASSERT(access-tvdom-at-pre S i);
        let C = get-tvdom S ! i;
        ASSERT(clause-not-marked-to-delete-heur-pre (S, C));
        b ← mop-clause-not-marked-to-delete-heur S C;
        if ¬b then RETURN (i+1, lbd, sze)
        else do {
          lbd2 ← mop-arena-lbd-st S C;
          sze' ← mop-arena-length-st S C;
          RETURN (i+1, max lbd (lbd2), max sze sze')
        }
      })
      (0, 0 :: nat , 0 :: nat);
    RETURN (lbd, sze)
  })›
```

```
definition mark-to-delete-clauses-wl-D-heur
  :: ‹isasat ⇒ isasat nres›
where
‹mark-to-delete-clauses-wl-D-heur  = (λS0. do {
    ASSERT(mark-to-delete-clauses-wl-D-heur-pre S0);
    S ← sort-vdom-heur S0;
    l ← number-clss-to-keep S;
    (lbd, sze) ← find-largest-lbd-and-size l S;
    ASSERT(length (get-tvdom S) ≤ length (get-clauses-wl-heur S0));
    (i, T) ← WHILE_T^λ-. True
      (λ(i, S). i < length (get-tvdom S))
      (λ(i, T). do {
        ASSERT(i < length (get-tvdom T));
        ASSERT(access-tvdom-at-pre T i);
        let C = get-tvdom T ! i;
        ASSERT(clause-not-marked-to-delete-heur-pre (T, C));
        b ← mop-clause-not-marked-to-delete-heur T C;
        if ¬b then RETURN (i, delete-index-vdom-heur i T)
        else do {
          ASSERT(access-lit-in-clauses-heur-pre ((T, C), 0));
          ASSERT(length (get-clauses-wl-heur T) ≤ length (get-clauses-wl-heur S0));
          ASSERT(length (get-tvdom T) ≤ length (get-clauses-wl-heur T));
          L ← mop-access-lit-in-clauses-heur T C 0;
          D ← get-the-propagation-reason-pol (get-trail-wl-heur T) L;
          length ← mop-arena-length (get-clauses-wl-heur T) C;
          let can-del = (D ≠ Some C) ∧
             length ≠ 2;
          if can-del
          then
            do {
              wasted ← mop-arena-length-st T C;
              - ← log-del-clause-heur T C;
              T ← mop-mark-garbage-heur3 C i (incr-wasted-st (of-nat wasted) T);
              RETURN (i, T)
            }
          else do {
            RETURN (i+1, T)
  }
      }
      })
      (l, S);
    ASSERT(length (get-tvdom T) ≤ length (get-clauses-wl-heur S0));
    incr-reduction-stat (set-stats-size-limit-st lbd sze T)
  })›
```

```
definition mark-to-delete-clauses-GC-wl-D-heur-pre :: ‹isasat ⇒ bool› where
  ‹mark-to-delete-clauses-GC-wl-D-heur-pre S ⟷
  (∃S'. (S, S') ∈ twl-st-heur-restart ∧ mark-to-delete-clauses-GC-wl-pre S')›
```

The duplication is unfortunate. The only difference is the precondition.

```
definition mark-to-delete-clauses-GC-wl-D-heur
  :: ‹isasat ⇒ isasat nres›
where
‹mark-to-delete-clauses-GC-wl-D-heur  = (λS0. do {
```

```
    ASSERT(mark-to-delete-clauses-GC-wl-D-heur-pre S0);
    S ← sort-vdom-heur S0;
    l ← number-clss-to-keep S;
    (lbd, sze) ← find-largest-lbd-and-size l S;
    ASSERT(length (get-tvdom S) ≤ length (get-clauses-wl-heur S0));
    (i, T) ← WHILE_T^λ-. True
      (λ(i, S). i < length (get-tvdom S))
      (λ(i, T). do {
        ASSERT(i < length (get-tvdom T));
        ASSERT(access-tvdom-at-pre T i);
        let C = get-tvdom T ! i;
        ASSERT(clause-not-marked-to-delete-heur-pre (T, C));
        b ← mop-clause-not-marked-to-delete-heur T C;
        if ¬b then RETURN (i, delete-index-vdom-heur i T)
        else do {
          ASSERT(access-lit-in-clauses-heur-pre ((T, C), 0));
          ASSERT(length (get-clauses-wl-heur T) ≤ length (get-clauses-wl-heur S0));
          ASSERT(length (get-tvdom T) ≤ length (get-clauses-wl-heur T));
          length ← mop-arena-length (get-clauses-wl-heur T) C;
          let can-del = length ≠ 2;
          if can-del
          then
            do {
              wasted ← mop-arena-length-st T C;
              - ← log-del-clause-heur T C;
              T ← mop-mark-garbage-heur3 C i (incr-wasted-st (of-nat wasted) T);
              RETURN (i, T)
            }
          else do {
            RETURN (i+1, T)
  }
        }
      })
      (l, S);
    ASSERT(length (get-tvdom T) ≤ length (get-clauses-wl-heur S0));
    incr-restart-stat (set-stats-size-limit-st lbd sze T)
  })›
```

definition *reduceint* :: ‹*64 word*› **where**
‹*reduceint = 1000*›

Approximatively *sqrt p* is $(2::'a)^{\textit{word-log2 p >> 1}}$

definition *schedule-next-reduction-st* :: ‹*isasat ⇒ isasat*› **where**

```
‹schedule-next-reduction-st S =
 (let (delta :: 64 word) = 1 + 2 << (word-log2 (max 1 (get-reductions-count S)) >> 1);
    delta = delta * reduceint;
    irred = (get-irredundant-count-st S) >> 10;
    extra = if irred < 10 then 1 else of-nat (word-log2 (irred)) >> 1;
    delta = extra * delta in
schedule-next-reduce-st delta S)›
```

definition *cdcl-twl-mark-clauses-to-delete* **where**

```
‹cdcl-twl-mark-clauses-to-delete S = do {
- ← ASSERT (mark-to-delete-clauses-wl-D-heur-pre S);
- ← RETURN (IsaSAT-Profile.start-reduce);
```

```
  T ← mark-to-delete-clauses-wl-D-heur S;
  - ← RETURN (IsaSAT-Profile.stop-reduce);
  RETURN (schedule-next-reduction-st (clss-size-resetUS0-st T))
  }›


definition cdcl-twl-restart-wl-heur where
‹cdcl-twl-restart-wl-heur S = do {
    cdcl-twl-local-restart-wl-D-heur S
  }›

definition isasat-GC-clauses-prog-copy-wl-entry
  :: ‹arena ⇒ (nat watcher) list list ⇒ nat literal ⇒
        (arena × isasat-aivdom) ⇒ (arena × (arena × isasat-aivdom)) nres›
where
‹isasat-GC-clauses-prog-copy-wl-entry = (λN0 W A (N', aivdom). do {
    ASSERT(nat-of-lit A < length W);
    ASSERT(length (W ! nat-of-lit A) ≤ length N0);
    let le = length (W ! nat-of-lit A);
    (i, N, N', aivdom) ← WHILE_T
      (λ(i, N, N', aivdom). i < le)
      (λ(i, N, (N', aivdom)). do {
        ASSERT(i < length (W ! nat-of-lit A));
        let C = fst (W ! nat-of-lit A ! i);
        ASSERT(arena-is-valid-clause-vdom N C);
        let st = arena-status N C;
        if st ≠ DELETED then do {
          ASSERT(arena-is-valid-clause-idx N C);
          ASSERT(length N' +
            (if arena-length N C > 4 then MAX-HEADER-SIZE else MIN-HEADER-SIZE) +
            arena-length N C ≤ length N0);
          ASSERT(length N = length N0);
           ASSERT(length (get-vdom-aivdom aivdom) < length N0);
          ASSERT(length (get-avdom-aivdom aivdom) < length N0);
          ASSERT(length (get-ivdom-aivdom aivdom) < length N0);
          ASSERT(length (get-tvdom-aivdom aivdom) < length N0);
       let D = length N' + (if arena-length N C > 4 then MAX-HEADER-SIZE else MIN-HEADER-SIZE);
          N' ← fm-mv-clause-to-new-arena C N N';
          ASSERT(mark-garbage-pre (N, C));
   RETURN (i+1, extra-information-mark-to-delete N C, N',
        (if st = LEARNED then add-learned-clause-aivdom-strong D aivdom else add-init-clause-aivdom-strong
D aivdom))
        } else RETURN (i+1, N, (N', aivdom))
      }) (0, N0, (N', aivdom));
    RETURN (N, (N', aivdom))
  })›

definition isasat-GC-clauses-prog-single-wl
  :: ‹arena ⇒  (arena × isasat-aivdom) ⇒ (nat watcher) list list ⇒ nat ⇒
        (arena × (arena × isasat-aivdom) × (nat watcher) list list) nres›
where
‹isasat-GC-clauses-prog-single-wl = (λN0 N' WS A. do {
    let L = Pos A; ///////////////////
    ASSERT(nat-of-lit L < length WS);
    ASSERT(nat-of-lit (−L) < length WS);
    (N, (N', aivdom)) ← isasat-GC-clauses-prog-copy-wl-entry N0 WS L N';
```

```
    let WS = WS[nat-of-lit L := []];
    ASSERT(length N = length N0);
    (N, N') ← isasat-GC-clauses-prog-copy-wl-entry N WS (−L) (N', aivdom);
    let WS = WS[nat-of-lit (−L) := []];
    RETURN (N, N', WS)
  })›

definition isasat-GC-clauses-prog-wl2 where
  ‹isasat-GC-clauses-prog-wl2 ≡ (λ(ns :: bump-heuristics) x0. do {
      (-, x) ← WHILE_T λ(n, x). length (fst x) = length (fst x0)
        (λ(n, -). n ≠ None)
        (λ(n, x). do {
          ASSERT(n ≠ None);
          let A = the n;
          ASSERT (A < length-bumped-vmtf-array ns);
          ASSERT(A ≤ unat32-max div 2);
          x ← (λ(arena_o, arena, W). isasat-GC-clauses-prog-single-wl arena_o arena W A) x;
          n ←  access-focused-vmtf-array ns A;
          RETURN (get-next n, x)
        })
        (Some (bumped-vmtf-array-fst ns), x0);
      RETURN x
    })›

definition isasat-GC-clauses-prog-wl :: ‹isasat ⇒ isasat nres› where
  ‹isasat-GC-clauses-prog-wl = (λS. do {
    let vm = get-vmtf-heur S;
    let N' = get-clauses-wl-heur S;
    let W' = get-watched-wl-heur S;
    let vdom = get-aivdom S;
    let old-arena = get-old-arena S;
    ASSERT(old-arena = []);
    (N, (N', vdom), WS) ← isasat-GC-clauses-prog-wl2 vm
       (N', (old-arena, empty-aivdom vdom), W');
    let S = set-watched-wl-heur WS S;
    let S = set-clauses-wl-heur N' S;
    let S = set-old-arena-wl-heur (take 0 N) S;
    let S = set-vmtf-wl-heur vm S;
    let S = set-stats-wl-heur (incr-GC (get-stats-heur S)) S;
    let S = set-aivdom-wl-heur vdom S;
    let heur = get-heur S;
    let heur = heuristic-reluctant-untrigger (set-zero-wasted heur);
    let S = set-heur-wl-heur heur S;
    RETURN S
  })›

definition isasat-GC-clauses-pre-wl-D :: ‹isasat ⇒ bool› where
‹isasat-GC-clauses-pre-wl-D S ⟷ (
  ∃ T. (S, T) ∈ twl-st-heur-restart ∧ cdcl-GC-clauses-pre-wl T
  )›

definition isasat-GC-clauses-wl-D :: ‹bool ⇒ isasat ⇒ isasat nres› where
‹isasat-GC-clauses-wl-D = (λinprocessing S. do {
  ASSERT(isasat-GC-clauses-pre-wl-D S);
  let b = True;
  if b then do {
```

```
    T ← isasat-GC-clauses-prog-wl S;
    ASSERT(length (get-clauses-wl-heur T) ≤ length (get-clauses-wl-heur S));
    ASSERT(∀ i ∈ set (get-tvdom T). i < length (get-clauses-wl-heur S));
    U ← rewatch-heur-and-reorder-st (empty-US-heur T);
    RETURN U
  }
  else RETURN S})›

end
theory IsaSAT-Restart
  imports
   Watched-Literals.WB-Sort Watched-Literals.Watched-Literals-Watch-List-Simp IsaSAT-Rephase-State
    IsaSAT-Setup IsaSAT-VMTF IsaSAT-Sorting IsaSAT-Proofs IsaSAT-Restart-Defs
    IsaSAT-Bump-Heuristics
begin
```

Chapter 22

Restarts

lemma *twl-st-heur-change-subsumed-clauses*:
  fixes *lcount lcount'* :: *clss-size*
  assumes ‹(*S*,
    (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*)) ∈ *twl-st-heur*›
    ‹*set-mset* (*all-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*)) = *set-mset* (*all-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS'*, *US'*, *N0*, *U0*, *Q*, *W*))›**and**
    ‹*clss-size-corr N NE UE NEk UEk NS' US' N0 U0 lcount'*›
  shows ‹(*set-learned-count-wl-heur lcount' S*,
    (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS'*, *US'*, *N0*, *U0*, *Q*, *W*)) ∈ *twl-st-heur*›
proof −
  note *cong* = *trail-pol-cong heuristic-rel-cong*
    option-lookup-clause-rel-cong isa-vmtf-cong heuristic-rel-cong
 note *cong2* = D_0*-cong phase-saving-cong cach-refinement-empty-cong vdom-m-cong isasat-input-nempty-cong*
   isasat-input-bounded-cong empty-occs-list-cong
  show *?thesis*
   supply[[*goals-limit=1*]]
   supply [*dest*!] = *cong*[*OF assms*(*2*)]
   using *assms*(*1*) *assms*(*3*) **apply** −
   unfolding *twl-st-heur-def in-pair-collect-simp prod.simps get-trail-wl.simps get-clauses-wl.simps*
     get-conflict-wl.simps literals-to-update-wl.simps get-watched-wl.simps
     IsaSAT-Setup.get-unkept-unit-learned-clss-wl.simps
     IsaSAT-Setup.get-kept-unit-init-clss-wl.simps
     IsaSAT-Setup.get-kept-unit-learned-clss-wl.simps get-subsumed-init-clauses-wl.simps
     get-subsumed-learned-clauses-wl.simps get-init-clauses0-wl.simps isasat-state-simp
      get-learned-clauses0-wl.simps IsaSAT-Setup.get-unkept-unit-init-clss-wl.simps
   apply *normalize-goal*+
   apply (*subst* (*asm*) *cong2*[*OF assms*(*2*)])+
   apply (*intro conjI*)
   apply ((*drule cong*[*OF assms*(*2*)])+; *assumption*)+
   done
qed

This is a list of comments (how does it work for glucose and cadical) to prepare the future refinement:

1. Reduction

   - every 2000+300*n (rougly since inprocessing changes the real number, cadical) (split over initialisation file); don't restart if level < 2 or if the level is less than the fast average

- curRestart * nbclausesbeforereduce; curRestart = (conflicts / nbclausesbeforereduce) + 1 (glucose)

2. Killed

   - half of the clauses that **can** be deleted (i.e., not used since last restart), not strictly LBD, but a probability of being useful.
   - half of the clauses

3. Restarts:

   - EMA-14, aka restart if enough clauses and slow_glue_avg * opts.restartmargin > fast_glue (file ema.cpp)
   - (lbdQueue.getavg() * K) > (sumLBD / conflictsRestarts), *conflictsRestarts > LOWER-BOUND-FO*
     && *lbdQueue.isvalid*() && *trail.size*() > *R* ∗ *trailQueue.getavg*()

declare *all-atms-def*[*symmetric*,*simp*]

lemma *twl-st-heur-restart-anaD*: ‹*x* ∈ *twl-st-heur-restart-ana r* ⟹ *x* ∈ *twl-st-heur-restart*›
by (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)

lemma *twl-st-heur-restartD*:
‹*x* ∈ *twl-st-heur-restart* ⟹ *x* ∈ *twl-st-heur-restart-ana* (*length* (*get-clauses-wl-heur* (*fst x*)))›
by (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)

named-theorems *twl-st-heur-restart*

lemma [*twl-st-heur-restart*]:
assumes ‹(*S*, *T*) ∈ *twl-st-heur-restart*›
shows ‹(*get-trail-wl-heur S*, *get-trail-wl T*) ∈ *trail-pol* (*all-init-atms-st T*)›
using *assms* **by** (*cases S*; *cases T*)
(*simp only*: *twl-st-heur-restart-def get-trail-wl.simps*
mem-Collect-eq prod.case get-clauses-wl.simps get-unit-init-clss-wl.simps all-init-atms-st-def
all-init-atms-def get-init-clauses0-wl.simps tuple17.inject get-unkept-unit-init-clss-wl.simps
get-kept-unit-init-clss-wl.simps
get-subsumed-init-clauses-wl.simps split: *isasat-int-splits*)

lemma *trail-pol-literals-are-in-*$\mathcal{L}_{in}$*-trail*:
‹(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$ ⟹ *literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *M*›
unfolding *literals-are-in-*$\mathcal{L}_{in}$*-trail-def trail-pol-def*
by *auto*

lemma *refine-generalise1*: ‹*A* ≤ *B* ⟹ *do* {*x* ← *B*; *C x*} ≤ *D* ⟹ *do* {*x* ← *A*; *C x*} ≤ (*D*:: *'a nres*)›
using *Refine-Basic.bind-mono*(*1*) *dual-order.trans* **by** *blast*

lemma *refine-generalise2*: *A* ≤ *B* ⟹ *do* {*x* ← *do* {*x* ← *B*; *A' x*}; *C x*} ≤ *D* ⟹
do {*x* ← *do* {*x* ← *A*; *A' x*}; *C x*} ≤ (*D*:: *'a nres*)
by (*simp add*: *refine-generalise1*)

lemma *trail-pol-no-dup*: ‹(*M*, *M'*) ∈ *trail-pol* $\mathcal{A}$ ⟹ *no-dup M'*›
by (*auto simp*: *trail-pol-def*)

definition *remove-all-annot-true-clause-imp-wl-D-pre*

:: ‹nat multiset ⇒ nat literal ⇒ nat twl-st-wl ⇒ bool›
where
‹remove-all-annot-true-clause-imp-wl-D-pre $\mathcal{A}$ L S ⟷ (L ∈# $\mathcal{L}_{all}$ $\mathcal{A}$)›

definition *remove-all-annot-true-clause-imp-wl-D-heur-pre* **where**
‹remove-all-annot-true-clause-imp-wl-D-heur-pre L S ⟷
(∃ S'. (S, S') ∈ twl-st-heur-restart
∧ remove-all-annot-true-clause-imp-wl-D-pre (all-init-atms-st S') L S')›

definition *five-uint64* :: ‹64 word› **where**
‹five-uint64 = 5›

definition *div2* **where** [*simp*]: ‹div2 n = n div 2›

definition *safe-minus* **where** ‹safe-minus a b = (if b ≥ a then 0 else a − b)›

definition *restart-flag-rel* :: ‹(8 word × restart-type) set› **where**
‹restart-flag-rel = {(FLAG-no-restart, NO-RESTART), (FLAG-restart, RESTART), (FLAG-GC-restart, GC),
(FLAG-Reduce-restart, GC), (FLAG-Inprocess-restart, INPROCESS)}›

lemma *clss-size-corr-restart-simp2*:
‹NO-MATCH {#} UE ⟹ clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c =
clss-size-corr-restart N NE {#} NEk UEk NS US N0 U0 c›
‹NO-MATCH {#} US ⟹ clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c =
clss-size-corr-restart N NE UE NEk UEk NS {#} N0 U0 c›
‹NO-MATCH {#} U0 ⟹ clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c =
clss-size-corr-restart N NE UE NEk UEk NS US N0 {#} c› **and**
clss-size-corr-restart-intro:
‹C ∈# dom-m N ⟹ ¬irred N C ⟹clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 c ⟹
clss-size-corr-restart (fmdrop C N) NE UE NEk UEk NS US' N0 U0' (clss-size-decr-lcount c)›
by (*auto simp*: *clss-size-corr-restart-def*)

lemma *mark-garbage-heur-wl*:
assumes
ST: ‹(S, T) ∈ twl-st-heur-restart› **and**
‹C ∈# dom-m (get-clauses-wl T)› **and**
‹¬ irred (get-clauses-wl T) C› **and** ‹i < length (get-tvdom S)› **and**
iC: ‹get-tvdom S ! i = C›
shows ‹(mark-garbage-heur3 C i S, mark-garbage-wl C T) ∈ twl-st-heur-restart›
proof −
show *?thesis*
using *assms distinct-mset-dom*[*of* ‹get-clauses-wl T›]
distinct-mset-mono[*of* ‹mset (get-avdom S)› ‹mset (get-vdom S)›]
apply (*cases S*; *cases T*)
apply (*simp add*: *twl-st-heur-restart-def mark-garbage-heur3-def mark-garbage-wl-def*)
by (*auto simp*: *twl-st-heur-restart-def mark-garbage-heur3-def mark-garbage-wl-def*
learned-clss-l-l-fmdrop size-remove1-mset-If clss-size-corr-restart-intro
simp: *all-init-atms-def all-init-lits-def aivdom-inv-dec-remove-clause*
simp del: *all-init-atms-def*[*symmetric*]
intro: *valid-arena-extra-information-mark-to-delete'*
intro!: *aivdom-inv-dec-remove-and-swap-inactive-tvdom*
aivdom-inv-dec-remove-and-swap-inactive
dest!: *in-set-butlastD in-vdom-m-fmdropD*
elim!: *in-set-upd-cases*)

qed

lemma *mark-garbage-heur-wl-ana*:
assumes
‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r*› **and**
‹*C* ∈# *dom-m* (*get-clauses-wl T*)› **and**
‹¬ *irred* (*get-clauses-wl T*) *C*› ‹*C* = *get-tvdom S* ! *i*› ‹*i* < *length* (*get-tvdom S*)›
shows ‹(*mark-garbage-heur3 C i S*, *mark-garbage-wl C T*) ∈ *twl-st-heur-restart-ana r*›
proof −
have [*intro*!]: ‹*distinct n* ⟹ *mset n* ⊆# *mset m* ⟹ *mset* (*removeAll* (*n* ! *i*) *n*) ⊆# *mset m*› **for** *n A m*
by (*metis filter-mset-eq-conv mset-filter removeAll-filter-not-eq subset-mset.dual-order.trans*)

show *?thesis*
using *assms*
using *assms distinct-mset-dom*[*of* ‹*get-clauses-wl T*›]
distinct-mset-mono[*of* ‹*mset* (*get-avdom S*)› ‹*mset* (*get-vdom S*)›]
apply (*cases S*; *cases T*)
apply (*simp add*: *twl-st-heur-restart-ana-def mark-garbage-heur3-def mark-garbage-wl-def*)
by (*auto simp*: *twl-st-heur-restart-def mark-garbage-heur3-def mark-garbage-wl-def*
learned-clss-l-l-fmdrop size-remove1-mset-If init-clss-l-fmdrop-irrelev aivdom-inv-dec-remove-clause
valid-arena-extra-information-mark-to-delete′ clss-size-corr-restart-intro
simp: *all-init-atms-def all-init-lits-def clss-size-corr-restart-simp2*
simp del: *all-init-atms-def*[*symmetric*] *clss-size-corr-restart-simp*
intro: *valid-arena-extra-information-mark-to-delete′*
intro!: *aivdom-inv-dec-remove-and-swap-inactive*
aivdom-inv-dec-remove-and-swap-inactive-tvdom
dest!: *in-set-butlastD in-vdom-m-fmdropD*
elim!: *in-set-upd-cases*)
qed

lemma *mark-unused-st-heur-ana*:
assumes
‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r*› **and**
‹*C* ∈# *dom-m* (*get-clauses-wl T*)›
shows ‹(*mark-unused-st-heur C S*, *T*) ∈ *twl-st-heur-restart-ana r*›
using *assms*
apply (*cases S*; *cases T*)
apply (*simp add*: *twl-st-heur-restart-ana-def mark-unused-st-heur-def*)
apply (*auto simp*: *twl-st-heur-restart-def mark-garbage-heur-def mark-garbage-wl-def*
learned-clss-l-l-fmdrop size-remove1-mset-If
simp: *all-init-atms-def all-init-lits-def*
simp del: *all-init-atms-def*[*symmetric*]
intro!: *valid-arena-mark-unused*
dest!: *in-set-butlastD in-vdom-m-fmdropD*
elim!: *in-set-upd-cases*)
done

lemma *twl-st-heur-restart-valid-arena*[*twl-st-heur-restart*]:
assumes
‹(*S*, *T*) ∈ *twl-st-heur-restart*›
shows ‹*valid-arena* (*get-clauses-wl-heur S*) (*get-clauses-wl T*) (*set* (*get-vdom S*))›
using *assms* **by** (*auto simp*: *twl-st-heur-restart-def*)

lemma *twl-st-heur-restart-get-avdom-nth-get-vdom*[*twl-st-heur-restart*]:
assumes

‹(S, T) ∈ twl-st-heur-restart› ‹i < length (get-avdom S)›
shows ‹get-avdom S ! i ∈ set (get-vdom S)›
using *assms* **by** (*auto 5 3 simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def aivdom-inv-dec-alt-def dest*!: *set-mset-mono*)

lemma *twl-st-heur-restart-get-tvdom-nth-get-vdom*[*twl-st-heur-restart*]:
assumes
‹(S, T) ∈ twl-st-heur-restart› ‹i < length (get-tvdom S)›
shows ‹get-tvdom S ! i ∈ set (get-vdom S)›
using *assms* **by** (*auto 5 3 simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def aivdom-inv-dec-alt-def dest*!: *set-mset-mono*)

lemma [*twl-st-heur-restart*]:
assumes
‹(S, T) ∈ twl-st-heur-restart› **and**
‹C ∈ set (get-avdom S)›
shows ‹clause-not-marked-to-delete-heur S C ⟷ (C ∈# dom-m (get-clauses-wl T))› **and**
‹C ∈# dom-m (get-clauses-wl T) ⟹ arena-lit (get-clauses-wl-heur S) C = get-clauses-wl T ∝ C ! 0›**and**
‹C ∈# dom-m (get-clauses-wl T) ⟹ arena-status (get-clauses-wl-heur S) C = LEARNED ⟷ ¬irred (get-clauses-wl T) C›
‹C ∈# dom-m (get-clauses-wl T) ⟹ arena-length (get-clauses-wl-heur S) C = length (get-clauses-wl T ∝ C)›
proof −
show ‹clause-not-marked-to-delete-heur S C ⟷ (C ∈# dom-m (get-clauses-wl T))›
using *assms*
by (*cases S*; *cases T*)
(*auto simp add*: *twl-st-heur-restart-def clause-not-marked-to-delete-heur-def*
arena-dom-status-iff(*1*) *aivdom-inv-dec-alt-def*
split: *prod.splits*)
assume *C*: ‹C ∈# dom-m (get-clauses-wl T)›
show ‹arena-lit (get-clauses-wl-heur S) C = get-clauses-wl T ∝ C ! 0›
using *assms C*
by (*cases S*; *cases T*)
(*auto simp add*: *twl-st-heur-restart-def clause-not-marked-to-delete-heur-def*
arena-lifting
split: *prod.splits*)
show ‹arena-status (get-clauses-wl-heur S) C = LEARNED ⟷ ¬irred (get-clauses-wl T) C›
using *assms C*
by (*cases S*; *cases T*)
(*auto simp add*: *twl-st-heur-restart-def clause-not-marked-to-delete-heur-def*
arena-lifting
split: *prod.splits*)
show ‹arena-length (get-clauses-wl-heur S) C = length (get-clauses-wl T ∝ C)›
using *assms C*
by (*cases S*; *cases T*)
(*auto simp add*: *twl-st-heur-restart-def clause-not-marked-to-delete-heur-def*
arena-lifting
split: *prod.splits*)
qed

lemma [*twl-st-heur-restart*]:
assumes
‹(S, T) ∈ twl-st-heur-restart-ana r› **and**
‹C ∈ set (get-avdom S)›

shows ‹*clause-not-marked-to-delete-heur S C* ⟷ (*C* ∈# *dom-m* (*get-clauses-wl T*))› **and**
‹*C* ∈# *dom-m* (*get-clauses-wl T*) ⟹ *arena-lit* (*get-clauses-wl-heur S*) *C* = *get-clauses-wl T* ∝ *C* ! *0*›**and**
‹*C* ∈# *dom-m* (*get-clauses-wl T*) ⟹ *arena-status* (*get-clauses-wl-heur S*) *C* = *LEARNED* ⟷ ¬*irred* (*get-clauses-wl T*) *C*›
‹*C* ∈# *dom-m* (*get-clauses-wl T*) ⟹ *arena-length* (*get-clauses-wl-heur S*) *C* = *length* (*get-clauses-wl T* ∝ *C*)›
using *assms*
by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart*)

lemma [*twl-st-heur-restart*]:
assumes
‹(*S*, *T*) ∈ *twl-st-heur-restart*› **and**
‹*C* ∈ *set* (*get-tvdom S*)›
shows ‹*clause-not-marked-to-delete-heur S C* ⟷ (*C* ∈# *dom-m* (*get-clauses-wl T*))› **and**
‹*C* ∈# *dom-m* (*get-clauses-wl T*) ⟹ *arena-lit* (*get-clauses-wl-heur S*) *C* = *get-clauses-wl T* ∝ *C* ! *0*›**and**
‹*C* ∈# *dom-m* (*get-clauses-wl T*) ⟹ *arena-status* (*get-clauses-wl-heur S*) *C* = *LEARNED* ⟷ ¬*irred* (*get-clauses-wl T*) *C*›
‹*C* ∈# *dom-m* (*get-clauses-wl T*) ⟹ *arena-length* (*get-clauses-wl-heur S*) *C* = *length* (*get-clauses-wl T* ∝ *C*)›
proof −
show ‹*clause-not-marked-to-delete-heur S C* ⟷ (*C* ∈# *dom-m* (*get-clauses-wl T*))›
using *assms*
by (*cases S*; *cases T*)
(*auto simp add*: *twl-st-heur-restart-def clause-not-marked-to-delete-heur-def*
arena-dom-status-iff(*1*) *aivdom-inv-dec-alt-def*
split: *prod.splits*)
assume *C*: ‹*C* ∈# *dom-m* (*get-clauses-wl T*)›
show ‹*arena-lit* (*get-clauses-wl-heur S*) *C* = *get-clauses-wl T* ∝ *C* ! *0*›
using *assms C*
by (*cases S*; *cases T*)
(*auto simp add*: *twl-st-heur-restart-def clause-not-marked-to-delete-heur-def*
arena-lifting
split: *prod.splits*)
show ‹*arena-status* (*get-clauses-wl-heur S*) *C* = *LEARNED* ⟷ ¬*irred* (*get-clauses-wl T*) *C*›
using *assms C*
by (*cases S*; *cases T*)
(*auto simp add*: *twl-st-heur-restart-def clause-not-marked-to-delete-heur-def*
arena-lifting
split: *prod.splits*)
show ‹*arena-length* (*get-clauses-wl-heur S*) *C* = *length* (*get-clauses-wl T* ∝ *C*)›
using *assms C*
by (*cases S*; *cases T*)
(*auto simp add*: *twl-st-heur-restart-def clause-not-marked-to-delete-heur-def*
arena-lifting
split: *prod.splits*)
qed

lemma [*twl-st-heur-restart*]:
assumes
‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r*› **and**
‹*C* ∈ *set* (*get-tvdom S*)›
shows ‹*clause-not-marked-to-delete-heur S C* ⟷ (*C* ∈# *dom-m* (*get-clauses-wl T*))› **and**
‹*C* ∈# *dom-m* (*get-clauses-wl T*) ⟹ *arena-lit* (*get-clauses-wl-heur S*) *C* = *get-clauses-wl T* ∝ *C* ! *0*›**and**

‹$C \in\# \mathit{dom\text{-}m}\ (\mathit{get\text{-}clauses\text{-}wl}\ T) \Longrightarrow \mathit{arena\text{-}status}\ (\mathit{get\text{-}clauses\text{-}wl\text{-}heur}\ S)\ C = \mathit{LEARNED} \longleftrightarrow \neg \mathit{irred}\ (\mathit{get\text{-}clauses\text{-}wl}\ T)\ C$›
‹$C \in\# \mathit{dom\text{-}m}\ (\mathit{get\text{-}clauses\text{-}wl}\ T) \Longrightarrow \mathit{arena\text{-}length}\ (\mathit{get\text{-}clauses\text{-}wl\text{-}heur}\ S)\ C = \mathit{length}\ (\mathit{get\text{-}clauses\text{-}wl}\ T \propto C)$›
using *assms*
by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart*)

lemma *number-clss-to-keep-impl-number-clss-to-keep*:
‹(*number-clss-to-keep-impl*, *number-clss-to-keep*) ∈ *Id* →$_f$ ⟨*nat-rel*⟩*nres-rel*›
by (*auto simp*: *number-clss-to-keep-impl-def number-clss-to-keep-def Let-def intro*!: *frefI nres-relI*)

lemma *in-set-delete-index-and-swapD*:
‹$x \in \mathit{set}\ (\mathit{delete\text{-}index\text{-}and\text{-}swap}\ \mathit{xs}\ i) \Longrightarrow x \in \mathit{set}\ \mathit{xs}$›
apply (*cases* ‹$i < \mathit{length}\ \mathit{xs}$›)
apply (*auto dest*!: *in-set-butlastD*)
by (*metis List.last-in-set in-set-upd-cases list.size*(*3*) *not-less-zero*)

lemma *delete-index-vdom-heur-twl-st-heur-restart-ana*:
‹$(S,\ T) \in \mathit{twl\text{-}st\text{-}heur\text{-}restart\text{-}ana}\ r \Longrightarrow i < \mathit{length}\ (\mathit{get\text{-}tvdom}\ S) \Longrightarrow$
$\mathit{get\text{-}tvdom}\ S\ !\ i \notin\# \mathit{dom\text{-}m}\ (\mathit{get\text{-}clauses\text{-}wl}\ T) \Longrightarrow$
$(\mathit{delete\text{-}index\text{-}vdom\text{-}heur}\ i\ S,\ T) \in \mathit{twl\text{-}st\text{-}heur\text{-}restart\text{-}ana}\ r$›
using *in-set-delete-index-and-swapD*[*of* - ‹*get-tvdom S*› *i*]
distinct-mset-mono[*of* ‹*mset* (*get-tvdom S*)› ‹*mset* (*get-vdom S*)›]
supply [[*goals-limit=1*]]
by (*clarsimp simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def delete-index-vdom-heur-def*)
(*auto intro*!: *aivdom-inv-dec-removed-inactive-tvdom*)

lemma *incr-wasted-st*:
assumes
‹$(S,\ T) \in \mathit{twl\text{-}st\text{-}heur\text{-}restart\text{-}ana}\ r$›
shows ‹$(\mathit{incr\text{-}wasted\text{-}st}\ C\ S,\ T) \in \mathit{twl\text{-}st\text{-}heur\text{-}restart\text{-}ana}\ r$›
using *assms*
apply (*cases S*; *cases T*)
apply (*simp add*: *twl-st-heur-restart-ana-def incr-wasted-st-def*)
apply (*auto simp*: *twl-st-heur-restart-def heuristic-rel-stats-def*
learned-clss-l-l-fmdrop size-remove1-mset-If phase-save-heur-rel-def
simp: *all-init-atms-def all-init-lits-def*
simp del: *all-init-atms-def*[*symmetric*]
intro!: *valid-arena-mark-unused*
dest!: *in-set-butlastD in-vdom-m-fmdropD*
elim!: *in-set-upd-cases*)
done

lemma *twl-st-heur-restart-same-annotD*:
‹$(S,\ T) \in \mathit{twl\text{-}st\text{-}heur\text{-}restart} \Longrightarrow \mathit{Propagated}\ L\ C \in \mathit{set}\ (\mathit{get\text{-}trail\text{-}wl}\ T) \Longrightarrow$
$\mathit{Propagated}\ L\ C' \in \mathit{set}\ (\mathit{get\text{-}trail\text{-}wl}\ T) \Longrightarrow C = C'$›
‹$(S,\ T) \in \mathit{twl\text{-}st\text{-}heur\text{-}restart} \Longrightarrow \mathit{Propagated}\ L\ C \in \mathit{set}\ (\mathit{get\text{-}trail\text{-}wl}\ T) \Longrightarrow$
$\mathit{Decided}\ L \in \mathit{set}\ (\mathit{get\text{-}trail\text{-}wl}\ T) \Longrightarrow \mathit{False}$›
by (*auto simp*: *twl-st-heur-restart-def dest*: *no-dup-no-propa-and-dec*
no-dup-same-annotD)

lemma $\mathcal{L}_{all}$*-mono*:
‹*set-mset* $\mathcal{A} \subseteq$ *set-mset* $\mathcal{B} \Longrightarrow L\ \in\#\ \mathcal{L}_{all}\ \mathcal{A} \Longrightarrow L\ \in\#\ \mathcal{L}_{all}\ \mathcal{B}$›
by (*auto simp*: $\mathcal{L}_{all}$*-def*)

lemma *all-lits-of-mm-mono2*:
‹*x* ∈# (*all-lits-of-mm A*) ⟹ *set-mset A* ⊆ *set-mset B* ⟹ *x* ∈# (*all-lits-of-mm B*)›
by (*auto simp*: *all-lits-of-mm-def*)

lemma $\mathcal{L}_{all}$*-init-all*:
‹*L* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1a*) ⟹ *L* ∈# $\mathcal{L}_{all}$ (*all-atms-st x1a*)›
using *all-init-lits-of-wl-all-lits-st* $\mathcal{L}_{all}$*-all-atms*
$\mathcal{L}_{all}$*-all-init-atms*(*2*) **by** *blast*

lemma *twl-st-heur-restartD2*:
‹*x* ∈ *twl-st-heur-restart* ⟹ *x* ∈ *twl-st-heur-restart-ana′* (*length* (*get-clauses-wl-heur* (*fst x*)))
(*learned-clss-count* (*fst x*))›
by (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)

lemma $\mathcal{L}_{all}$*-atm-of-all-init-lits-of-mm*:
‹*set-mset* ($\mathcal{L}_{all}$ (*atm-of* ‘# *all-init-lits N NUE*)) = *set-mset* (*all-init-lits N NUE*)›
by (*auto simp*: *all-init-lits-def* $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm*)

lemma *trail-pol-replace-annot-in-trail-spec*:
assumes
‹*atm-of x2* < *length* (*trail-get-reason* (*get-trail-wl-heur S*))› **and**
x2: ‹*atm-of x2* ∈# *all-init-atms-st* (*ys* @ *Propagated x2 C* # *zs*, *x2n′*)› **and**
‹(*S*, (*ys* @ *Propagated x2 C* # *zs*, *x2n′*))
∈ *twl-st-heur-restart-ana r*› **and**
M: ‹*M* = *get-trail-wl-heur S*›
shows
‹(*set-trail-wl-heur* (*trail-update-reason-at x2 0 M*) *S*,
(*ys* @ *Propagated x2 0* # *zs*, *x2n′*))
∈ *twl-st-heur-restart-ana r*›
proof −
let *?S* = ‹(*ys* @ *Propagated x2 C* # *zs*, *x2n′*)›
let *?A* = ‹*all-init-atms-st ?S*›
have *pol*: ‹(*get-trail-wl-heur S*, *ys* @ *Propagated x2 C* # *zs*)
∈ *trail-pol* (*all-init-atms-st ?S*)›
using *assms*(*3*) **unfolding** *twl-st-heur-restart-ana-def twl-st-heur-restart-def all-init-atms-st-def*
by (*cases x2n′*) *auto*
obtain *x y x1b x1c x1d x1e x2d x2e* **where**
tr: ‹*get-trail-wl-heur S* = (*x1b*, *x1c*, *x1d*, *x1e*, *x2d*)› **and**
x2d: ‹*x2d* = (*count-decided* (*ys* @ *Propagated x2 C* # *zs*), *y*, *x2e*)› **and**
reasons: ‹((*map lit-of* (*rev* (*ys* @ *Propagated x2 C* # *zs*)), *x1e*),
ys @ *Propagated x2 C* # *zs*)
∈ *ann-lits-split-reasons ?A*› **and**
pol: ‹∀ *L*∈#$\mathcal{L}_{all}$ *?A*. *nat-of-lit L* < *length x1c* ∧
x1c ! *nat-of-lit L* = *polarity* (*ys* @ *Propagated x2 C* # *zs*) *L*› **and**
n-d: ‹*no-dup* (*ys* @ *Propagated x2 C* # *zs*)› **and**
lvls: ‹∀ *L*∈#$\mathcal{L}_{all}$ *?A*. *atm-of L* < *length x1d* ∧
x1d ! *atm-of L* = *get-level* (*ys* @ *Propagated x2 C* # *zs*) *L*› **and**
‹*undefined-lit ys* (*lit-of* (*Propagated x2 C*))› **and**
‹*undefined-lit zs* (*lit-of* (*Propagated x2 C*))› **and**
inA:‹∀ *L*∈*set* (*ys* @ *Propagated x2 C* # *zs*). *lit-of L* ∈# $\mathcal{L}_{all}$ *?A*› **and**
cs: ‹*control-stack y* (*ys* @ *Propagated x2 C* # *zs*)› **and**
‹*literals-are-in-*$\mathcal{L}_{in}$*-trail ?A* (*ys* @ *Propagated x2 C* # *zs*)› **and**

‹length (ys @ Propagated x2 C # zs) < unat32-max› and
‹length (ys @ Propagated x2 C # zs) ≤ unat32-max div 2 + 1› and
‹count-decided (ys @ Propagated x2 C # zs) < unat32-max› and
‹length (map lit-of (rev (ys @ Propagated x2 C # zs))) =
length (ys @ Propagated x2 C # zs)› and
bounded: ‹isasat-input-bounded ?A› and
x1b: ‹x1b = map lit-of (rev (ys @ Propagated x2 C # zs))› and
‹zeroed-trail (ys @ Propagated x2 C # zs) x2e›
using pol **unfolding** trail-pol-alt-def
apply (cases ‹get-trail-wl-heur S›)
by blast
have lev-eq: ‹get-level (ys @ Propagated x2 C # zs) =
get-level (ys @ Propagated x2 0 # zs)›
‹count-decided (ys @ Propagated x2 C # zs) =
count-decided (ys @ Propagated x2 0 # zs)›
by (auto simp: get-level-cons-if get-level-append-if)
have [simp]: ‹atm-of x2 < length x1e›
using reasons x2 assms(1) **unfolding** tr **by** auto

have ‹((x1b, x1e[atm-of x2 := 0]), ys @ Propagated x2 0 # zs)
∈ ann-lits-split-reasons ?A›
using reasons n-d undefined-notin
by (auto simp: ann-lits-split-reasons-def x1b
DECISION-REASON-def atm-of-eq-atm-of)
moreover have n-d': ‹no-dup (ys @ Propagated x2 0 # zs)›
using n-d **by** auto
moreover have ‹∀ L∈#$\mathcal{L}_{all}$?A.
nat-of-lit L < length x1c ∧
x1c ! nat-of-lit L = polarity (ys @ Propagated x2 0 # zs) L›
using pol **by** (auto simp: polarity-def)
moreover have ‹∀ L∈#$\mathcal{L}_{all}$?A.
atm-of L < length x1d ∧
x1d ! atm-of L = get-level (ys @ Propagated x2 0 # zs) L›
using lvls **by** (auto simp: get-level-append-if get-level-cons-if)
moreover have ‹control-stack y (ys @ Propagated x2 0 # zs)›
using cs **apply** −
apply (subst control-stack-alt-def[symmetric, OF n-d'])
apply (subst (asm) control-stack-alt-def[symmetric, OF n-d])
unfolding control-stack'-def lev-eq
apply normalize-goal
apply (intro ballI conjI)
apply (solves auto)
unfolding set-append list.set(2) Un-iff insert-iff
apply (rule disjE, assumption)
subgoal for L
by (drule-tac x = L **in** bspec)
(auto simp: nth-append nth-Cons split: nat.splits)
apply (rule disjE[of ‹- = -›], assumption)
subgoal for L
by (auto simp: nth-append nth-Cons split: nat.splits)
subgoal for L
by (drule-tac x = L **in** bspec)
(auto simp: nth-append nth-Cons split: nat.splits)
done
moreover have ‹zeroed-trail (ys @ Propagated x2 0 # zs) x2e›
using ‹zeroed-trail (ys @ Propagated x2 C # zs) x2e›

```
    by (auto simp: zeroed-trail-def nth-append nth-Cons split: if-splits nat.splits)
  ultimately have
    ‹((x1b, x1c, x1d, x1e[atm-of x2 := 0], x2d), ys @ Propagated x2 0 # zs)
        ∈ trail-pol ?A›
    using n-d x2 inA bounded
    unfolding trail-pol-def x2d
    by simp

  moreover { fix aaa ca
    have ‹vmtf-\mathcal{L}_all (all-init-atms aaa ca) (ys @ Propagated x2 C # zs) =
       vmtf-\mathcal{L}_all (all-init-atms aaa ca) (ys @ Propagated x2 0 # zs)›
      by (auto simp: vmtf-\mathcal{L}_all-def)
    then have ‹vmtf (all-init-atms aaa ca) (ys @ Propagated x2 C # zs) =
      vmtf (all-init-atms aaa ca) (ys @ Propagated x2 0 # zs)›
      by (auto simp: vmtf-def  vmtf-def
image-iff)
    moreover have ‹vmtf (all-init-atms aaa ca) (get-unit-trail (ys @ Propagated x2 C # zs)) =
      vmtf (all-init-atms aaa ca) (get-unit-trail(ys @ Propagated x2 0 # zs))›
      by (auto simp: vmtf-def  get-unit-trail-def takeWhile-append vmtf-\mathcal{L}_all-def
image-iff)
    moreover have ‹acids (all-init-atms aaa ca) (get-unit-trail (ys @ Propagated x2 C # zs)) =
      acids (all-init-atms aaa ca) (get-unit-trail(ys @ Propagated x2 0 # zs))›
      apply (auto simp: acids-def get-unit-trail-def takeWhile-append vmtf-\mathcal{L}_all-def
image-iff)
      apply (metis (no-types, lifting) map-lit-of-eq-defined-litD valid-trail-reduction-eq-alt-def
        valid-trail-reduction-eq-change-annot)+
      done
    moreover have ‹acids (all-init-atms aaa ca) ((ys @ Propagated x2 C # zs)) =
      acids (all-init-atms aaa ca) ((ys @ Propagated x2 0 # zs))›
      apply (auto simp: acids-def get-unit-trail-def takeWhile-append vmtf-\mathcal{L}_all-def
image-iff)
      apply (metis (mono-tags, lifting) map-lit-of-eq-defined-litD valid-trail-reduction-eq-alt-def
        valid-trail-reduction-eq-change-annot)+
      done
    ultimately have ‹bump-heur (all-init-atms aaa ca) (ys @ Propagated x2 C # zs) =
      bump-heur (all-init-atms aaa ca) (ys @ Propagated x2 0 # zs)›
      by (auto simp: bump-heur-def)
  }
  moreover { fix D
    have ‹get-level (ys @ Propagated x2 C # zs) = get-level (ys @ Propagated x2 0 # zs)›
      by (auto simp: get-level-append-if get-level-cons-if)
    then have ‹counts-maximum-level (ys @ Propagated x2 C # zs) D =
      counts-maximum-level (ys @ Propagated x2 0 # zs) D› and
      ‹out-learned (ys @ Propagated x2 C # zs) = out-learned (ys @ Propagated x2 0 # zs)›
      by (auto simp: counts-maximum-level-def card-max-lvl-def
        out-learned-def intro!: ext)
  }
  ultimately show ?thesis
    using assms(3) unfolding twl-st-heur-restart-ana-def M
    by (cases x2n')
      (auto simp: twl-st-heur-restart-def all-init-atms-st-def tr trail-update-reason-at-def
        simp flip: mset-map drop-map)
qed

lemmas trail-pol-replace-annot-in-trail-spec2 =
  trail-pol-replace-annot-in-trail-spec[of ‹- -›, simplified]
```

lemma $\mathcal{L}_{all}$*-ball-all*:
‹$(\forall\, L \in\# \mathcal{L}_{all}\ (\textit{all-atms}\ N\ \textit{NUE}).\ P\ L) = (\forall\, L \in\# \textit{all-lits}\ N\ \textit{NUE}.\ P\ L)$›
‹$(\forall\, L \in\# \mathcal{L}_{all}\ (\textit{all-init-atms}\ N\ \textit{NUE}).\ P\ L) = (\forall\, L \in\# \textit{all-init-lits}\ N\ \textit{NUE}.\ P\ L)$›
by (*simp-all add*: $\mathcal{L}_{all}$*-all-atms-all-lits* $\mathcal{L}_{all}$*-all-init-atms*)

lemma *clss-size-corr-restart-intro3*[*intro*]:
‹*clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 lcount* $\Longrightarrow$
clss-size-corr-restart N NE UE NEk UEk NS $\{\#\}$ *N0* $\{\#\}$ (*clss-size-resetUS0 lcount*)›
by (*auto simp*: *clss-size-corr-restart-def clss-size-resetUS-def clss-size-def*
clss-size-resetU0-def clss-size-resetUE-def)

lemma *twl-st-heur-restart-ana-US-empty*:
‹*NO-MATCH* $\{\#\}$ *US* $\Longrightarrow$ *NO-MATCH* $\{\#\}$ *U0* $\Longrightarrow$ *NO-MATCH* $\{\#\}$ *UE* $\Longrightarrow$ (S, M, N, D, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, W, Q) $\in$ *twl-st-heur-restart-ana r* $\Longrightarrow$
(*set-learned-count-wl-heur* (*clss-size-resetUS0* (*get-learned-count S*)) S, M, N, D, *NE*, $\{\#\}$, *NEk*, *UEk*, *NS*, $\{\#\}$, *N0*, $\{\#\}$, W, Q)
$\in$ *twl-st-heur-restart-ana r*›
by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*
intro: *clss-size-corr-simp*)

fun *equality-except-trail-empty-US-wl* :: ‹'*v twl-st-wl* $\Rightarrow$ '*v twl-st-wl* $\Rightarrow$ *bool*› **where**
‹*equality-except-trail-empty-US-wl* (M, N, D, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *WS*, Q)
(M', N', D', *NE'*, *UE'*, *NEk'*, *UEk'*, *NS'*, *US'*, *N0'*, *U0'*, *WS'*, Q') $\longleftrightarrow$
$N = N' \wedge D = D' \wedge \textit{NE} = \textit{NE}' \wedge \textit{NEk} = \textit{NEk}' \wedge \textit{UEk} = \textit{UEk}' \wedge \textit{NS} = \textit{NS}' \wedge \textit{US} = \{\#\} \wedge \textit{UE} = \{\#\} \wedge$
$\textit{N0}' = \textit{N0} \wedge \textit{U0} = \{\#\} \wedge \textit{WS} = \textit{WS}' \wedge Q = Q'$›

lemma *equality-except-conflict-wl-get-clauses-wl*:
‹*equality-except-conflict-wl S Y* $\Longrightarrow$ *get-clauses-wl S* = *get-clauses-wl Y*› **and**
equality-except-conflict-wl-get-trail-wl:
‹*equality-except-conflict-wl S Y* $\Longrightarrow$ *get-trail-wl S* = *get-trail-wl Y*› **and**
equality-except-trail-empty-US-wl-get-conflict-wl:
‹*equality-except-trail-empty-US-wl S Y* $\Longrightarrow$ *get-conflict-wl S* = *get-conflict-wl Y*› **and**
equality-except-trail-empty-US-wl-get-clauses-wl:
‹*equality-except-trail-empty-US-wl S Y*$\Longrightarrow$ *get-clauses-wl S* = *get-clauses-wl Y*›
by (*cases S*; *cases Y*; *solves auto*)+

lemma *isasat-replace-annot-in-trail-replace-annot-in-trail-spec*:
‹((L, C), S), ((L', C'), S')) $\in$ *Id* $\times_f$ *Id* $\times_f$ *twl-st-heur-restart-ana'* r u $\Longrightarrow$
isasat-replace-annot-in-trail L C S $\leq$
$\Downarrow\{(U,\ U').\ (U,\ U') \in \textit{twl-st-heur-restart-ana}'\ r\ u \wedge$
get-clauses-wl-heur U = *get-clauses-wl-heur S* $\wedge$
get-learned-count U = *clss-size-resetUS0* (*get-learned-count S*) $\wedge$
get-vdom U = *get-vdom S* $\wedge$
equality-except-trail-empty-US-wl U' S'$\}$
(*replace-annot-wl L' C' S'*)›
unfolding *isasat-replace-annot-in-trail-def replace-annot-wl-def*
uncurry-def replace-reason-in-trail-def nres-monad3
apply (*cases S'*, *hypsubst*, *unfold prod.case*)
apply *refine-rcg*
subgoal
by (*auto simp*: *trail-pol-alt-def ann-lits-split-reasons-def* $\mathcal{L}_{all}$*-ball-all all-init-lits-of-wl-def*
twl-st-heur-restart-def twl-st-heur-restart-ana-def replace-annot-wl-pre-def
all-init-lits-alt-def(*2*))
subgoal for *x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f*

unfolding *replace-annot-wl-pre-def replace-annot-l-pre-def bind-to-let-conv Let-def*
apply (*clarify dest*!: *split-list*[*of* ‹*Propagated* - -›])
apply (*rule RETURN-SPEC-refine*)
apply (*rule-tac x* = ‹(*ys* @ *Propagated L 0* # *zs*, *x2*, *x1a*, *x2a*, {#}, *x2b*, *x1c*, *x2c*, {#}, *x2d*, {#}, *x2e*, *x1f*)› **in** *exI*)
apply (*intro conjI*)
prefer *2*
apply (*rule-tac x* = ‹*ys* @ *Propagated L 0* # *zs*› **in** *exI*)
apply (*intro conjI*)
apply *blast*
by (*auto intro*!: *trail-pol-replace-annot-in-trail-spec*[**where** *C*=*C*]
trail-pol-replace-annot-in-trail-spec2
simp: *atm-of-eq-atm-of all-init-atms-def replace-annot-wl-pre-def*
$\mathcal{L}_{all}$*-ball-all replace-annot-l-pre-def state-wl-l-def all-init-lits-of-wl-def*
all-init-lits-def ac-simps
twl-st-heur-restart-ana-US-empty learned-clss-count-def all-init-atms-st-def
simp del: *all-init-atms-def*[*symmetric*]; *fail*)+
done

lemma *get-pos-of-level-in-trail-le-decomp*:
assumes
‹(*S*, *T*) ∈ *twl-st-heur-restart*›
shows ‹*get-pos-of-level-in-trail* (*get-trail-wl T*) *0*
≤ *SPEC*
(λ*k*. ∃ *M1*. (∃ *M2 K*.
(*Decided K* # *M1*, *M2*)
∈ *set* (*get-all-ann-decomposition* (*get-trail-wl T*))) ∧
count-decided M1 = *0* ∧ *k* = *length M1*)›
unfolding *get-pos-of-level-in-trail-def*
proof (*rule SPEC-rule*)
fix *x*
assume *H*: ‹*x* < *length* (*get-trail-wl T*) ∧
is-decided (*rev* (*get-trail-wl T*) ! *x*) ∧
get-level (*get-trail-wl T*) (*lit-of* (*rev* (*get-trail-wl T*) ! *x*)) = *0* + *1*›
let *?M1* = ‹*rev* (*take x* (*rev* (*get-trail-wl T*)))›
let *?K* = ‹*Decided* ((*lit-of*(*rev* (*get-trail-wl T*) ! *x*)))›
let *?M2* = ‹*rev* (*drop* (*Suc x*) (*rev* (*get-trail-wl T*)))›
have *T*: ‹(*get-trail-wl T*) = *?M2* @ *?K* # *?M1*› **and**
K: ‹*Decided* (*lit-of ?K*) = *?K*›
apply (*subst append-take-drop-id*[*symmetric*, *of* - ‹*length* (*get-trail-wl T*) − *Suc x*›])
apply (*subst Cons-nth-drop-Suc*[*symmetric*])
using *H*
apply (*auto simp*: *rev-take rev-drop rev-nth*)
apply (*cases* ‹*rev* (*get-trail-wl T*) ! *x*›)
apply (*auto simp*: *rev-take rev-drop rev-nth*)
done
have *n-d*: ‹*no-dup* (*get-trail-wl T*)›
using *assms*(*1*)
by (*auto simp*: *twl-st-heur-restart-def*)
obtain *M2* **where**
‹(*?K* # *?M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*get-trail-wl T*))›
using *get-all-ann-decomposition-ex*[*of* ‹*lit-of ?K*› *?M1 ?M2*]
apply (*subst* (*asm*) *K*)
apply (*subst* (*asm*) *K*)
apply (*subst* (*asm*) *T*[*symmetric*])
by *blast*

moreover have ‹*count-decided ?M1 = 0*›
using *n-d H*
by (*subst* (*asm*)(*1*) *T*, *subst* (*asm*)(*11*) *T*, *subst T*) *auto*
moreover have ‹*x = length ?M1*›
using *n-d H* **by** *auto*
ultimately show ‹∃ *M1*. (∃ *M2 K*. (*Decided K # M1*, *M2*)
∈ *set* (*get-all-ann-decomposition* (*get-trail-wl T*))) ∧
count-decided M1 = 0 ∧ *x = length M1* ›
by *blast*
qed

lemma *twl-st-heur-restart-isa-length-trail-get-trail-wl*:
‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r* ⟹ *mop-isa-length-trail* (*get-trail-wl-heur S*) = *RETURN* (*length* (*get-trail-wl T*))›
unfolding *isa-length-trail-def twl-st-heur-restart-ana-def twl-st-heur-restart-def trail-pol-alt-def*
mop-isa-length-trail-def isa-length-trail-pre-def
by (*subgoal-tac* ‹(*case get-trail-wl-heur S of*
(*M'*, *xs*, *lvls*, *reasons*, *k*, *cs*) ⇒ *length M'* ≤ *unat32-max*)›)
(*cases S*;*auto dest*: *ann-lits-split-reasons-map-lit-of intro*!: *ASSERT-leI*; *fail*)+

lemma *twl-st-heur-restart-count-decided-st-alt-def*:
fixes *S* :: *isasat*
shows ‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r* ⟹ *count-decided-st-heur S = count-decided* (*get-trail-wl T*)›
unfolding *count-decided-st-def twl-st-heur-restart-ana-def trail-pol-def twl-st-heur-restart-def*
by (*cases T*) (*auto simp*: *count-decided-st-heur-def count-decided-pol-def*)

lemma *twl-st-heur-restart-trailD*:
‹(*S*, *T*) ∈ *twl-st-heur-restart-ana r* ⟹
(*get-trail-wl-heur S*, *get-trail-wl T*) ∈ *trail-pol* (*all-init-atms-st T*)›
by (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def all-init-atms-st-def*
get-unit-init-clss-wl-alt-def)

lemma *no-dup-nth-proped-dec-notin*:
‹*no-dup M* ⟹ *k < length M* ⟹ *M* ! *k = Propagated L C* ⟹ *Decided L* ∉ *set M*›
apply (*auto dest*!: *split-list simp*: *nth-append nth-Cons defined-lit-def in-set-conv-nth*
split: *if-splits nat.splits*)
by (*metis no-dup-no-propa-and-dec nth-mem*)

lemma *get-literal-and-reason*:
assumes
‹((*k*, *S*), *k'*, *T*) ∈ *nat-rel* ×*f twl-st-heur-restart-ana r*› **and**
‹*remove-one-annot-true-clause-one-imp-wl-pre k' T*› **and**
proped: ‹*is-proped* (*rev* (*get-trail-wl T*) ! *k'*)›
shows ‹*do* {
L ← *isa-trail-nth* (*get-trail-wl-heur S*) *k*;
C ← *get-the-propagation-reason-pol* (*get-trail-wl-heur S*) *L*;
RETURN (*L*, *C*)
} ≤ ⇓ {((*L*, *C*), *L'*, *C'*). *L = L'* ∧ *C' = the C* ∧ *C* ≠ *None*}
(*SPEC* (λ*p*. *rev* (*get-trail-wl T*) ! *k' = Propagated* (*fst p*) (*snd p*)))›
proof −
have *n-d*: ‹*no-dup* (*get-trail-wl T*)› **and**
res: ‹((*k*, *S*), *k'*, *T*) ∈ *nat-rel* ×*f twl-st-heur-restart*›
using *assms* **by** (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)
from *no-dup-nth-proped-dec-notin*[*OF this*(*1*), *of* ‹*length* (*get-trail-wl T*) − *Suc k'*›]
have *dec-notin*: ‹*Decided* (*lit-of* (*rev* (*fst T*) ! *k'*)) ∉ *set* (*fst T*)›

```
  using proped assms(2) by (cases T; cases ‹rev (get-trail-wl T) ! k'›)
   (auto simp: twl-st-heur-restart-def state-wl-l-def
    remove-one-annot-true-clause-one-imp-wl-pre-def twl-st-l-def
    remove-one-annot-true-clause-one-imp-pre-def rev-nth
    dest: no-dup-nth-proped-dec-notin)
have k': ‹k' < length (get-trail-wl T)› and [simp]: ‹fst T = get-trail-wl T›
  using proped assms(2)
  by (cases T; auto simp: twl-st-heur-restart-def state-wl-l-def
    remove-one-annot-true-clause-one-imp-wl-pre-def twl-st-l-def
    remove-one-annot-true-clause-one-imp-pre-def; fail)+
define k'' where ‹k'' ≡ length (get-trail-wl T) − Suc k'›
have k'': ‹k'' < length (get-trail-wl T)›
  using k' by (auto simp: k''-def)
have ‹rev (get-trail-wl T) ! k' = get-trail-wl T ! k''›
  using k' k'' by (auto simp: k''-def nth-rev)
then have ‹rev-trail-nth (fst T) k' ∈# \mathcal{L}_{all} (all-init-atms-st T)›
  using k'' assms nth-mem[OF k']
  by (auto simp: twl-st-heur-restart-ana-def rev-trail-nth-def get-unit-init-clss-wl-alt-def
    trail-pol-alt-def twl-st-heur-restart-def all-init-atms-st-def)
then have 1: ‹(SPEC (λp. rev (get-trail-wl T) ! k' = Propagated (fst p) (snd p))) =
  do {
    L ← RETURN (rev-trail-nth (fst T) k');
    ASSERT(L ∈# \mathcal{L}_{all} (all-init-atms-st T));
    C ← (get-the-propagation-reason (fst T) L);
    ASSERT(C ≠ None);
    RETURN (L, the C)
  }›
  using proped dec-notin k' nth-mem[OF k'] no-dup-same-annotD[OF n-d]
  apply (subst dual-order.eq-iff)
  apply (rule conjI)
  subgoal
    unfolding get-the-propagation-reason-def
    apply (cases ‹rev (get-trail-wl T) ! k'›)
    apply (auto simp: RES-RES-RETURN-RES rev-trail-nth-def
          get-the-propagation-reason-def lits-of-def rev-nth
      RES-RETURN-RES
          dest: split-list
    simp flip: k''-def
    intro!: le-SPEC-bindI[of - ‹Some (mark-of (get-trail-wl T ! k''))›])
    apply (cases ‹rev (get-trail-wl T) ! k'›)
    apply  (auto simp: RES-RES-RETURN-RES rev-trail-nth-def RES-ASSERT-moveout
        get-the-propagation-reason-def lits-of-def rev-nth
  RES-RETURN-RES
      simp flip: k''-def
      dest: split-list
      intro!: exI[of - ‹Some (mark-of (rev (get-trail-wl T) ! k'))›])
  apply (subst RES-ASSERT-moveout)
   by (auto 4 3 simp: RES-RETURN-RES image-iff
      dest: split-list
      intro!: exI[of - ‹Some (mark-of ((get-trail-wl T) ! k''))›])
  subgoal
    apply (cases ‹rev (get-trail-wl T) ! k'›)
    apply  (auto simp: RES-RES-RETURN-RES rev-trail-nth-def RES-ASSERT-moveout
        get-the-propagation-reason-def lits-of-def rev-nth
  RES-RETURN-RES
      simp flip: k''-def
```

dest: *split-list*
intro!: *exI*[*of* - ‹*Some* (*mark-of* (*rev* (*get-trail-wl T*) ! *k'*))›])
apply (*subst RES-ASSERT-moveout*)
by (*auto 4 3 simp*: *RES-RETURN-RES image-iff*
dest: *split-list*
intro!: *exI*[*of* - ‹*Some* (*mark-of* ((*get-trail-wl T*) ! *k''*))›])
done

show *?thesis*
supply *RETURN-as-SPEC-refine*[*refine2 del*]
apply (*subst 1*)
apply (*refine-rcg*
isa-trail-nth-rev-trail-nth[*THEN fref-to-Down-curry*, *unfolded comp-def*,
of - - - - ‹*all-init-atms-st T*›]
get-the-propagation-reason-pol[*THEN fref-to-Down-curry*, *unfolded comp-def*,
of ‹*all-init-atms-st T*›])
subgoal using *k'* **by** *auto*
subgoal using *assms* **by** (*cases S*; *auto dest*: *twl-st-heur-restart-trailD*)
subgoal by *auto*
subgoal for *K K'*
using *assms* **by** (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*
all-init-atms-st-def get-unit-init-clss-wl-alt-def)
subgoal
by *auto*
done
qed

lemma *red-in-dom-number-of-learned-ge1*: ‹*C'* ∈# *dom-m baa* ⟹ ¬ *irred baa C'* ⟹ *Suc 0* ≤ *size* (*learned-clss-l baa*)›
by (*auto simp*: *ran-m-def dest*!: *multi-member-split*)

definition *find-decomp-wl0* :: ‹'*v twl-st-wl* ⇒ '*v twl-st-wl* ⇒ *bool*› **where**
‹*find-decomp-wl0* = (λ(*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*) (*M'*, *N'*, *D'*, *NE'*, *UE'*, *NEk'*, *UEk'*, *NS'*, *US'*, *N0'*, *U0'*, *Q'*, *W'*).
(∃*K M2*. (*Decided K* # *M'*, *M2*) ∈ *set* (*get-all-ann-decomposition M*) ∧
count-decided M' = *0*) ∧
(*N'*, *D'*, *NE'*, *UE'*, *NEk'*, *UEk'*, *NS'*, *US*, *N0*, *U0*, *Q'*, *W'*) = (*N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US'*, *N0'*, *U0'*, *Q*, *W*))›
lemma *cdcl-twl-local-restart-wl-spec0-alt-def*:
‹*cdcl-twl-local-restart-wl-spec0* = (λ*S*. *do* {
ASSERT(*restart-abs-wl-pre2 S False*);
if count-decided (*get-trail-wl S*) > *0*
then do {
T ← *SPEC*(*find-decomp-wl0 S*);
RETURN (*empty-Q-wl T*)
} *else RETURN* (*empty-US-heur-wl S*)})›
by (*intro ext*; *case-tac S*)
(*auto 6 4 simp*: *cdcl-twl-local-restart-wl-spec0-def*
RES-RETURN-RES2 image-iff RES-RETURN-RES empty-US-heur-wl-def
find-decomp-wl0-def empty-Q-wl-def uncurry-def
intro!: *bind-cong*[*OF refl*]
dest: *get-all-ann-decomposition-exists-prepend*)

```
lemma cdcl-twl-local-restart-wl-spec0:
  assumes Sy: ‹(S, y) ∈ twl-st-heur-restart-ana' r u› and
    ‹get-conflict-wl y = None›
  shows ‹do {
      if count-decided-st-heur S > 0
      then do {
        S ← find-decomp-wl-st-int 0 S;
        empty-Q (empty-US-heur S)
      } else RETURN (empty-US-heur S)
    }
        ≤ ⇓ (twl-st-heur-restart-ana' r u) (cdcl-twl-local-restart-wl-spec0 y)›
proof −
  define upd :: ‹- ⇒ - ⇒ isasat ⇒ isasat› where
    ‹upd M' vm = (λ S. set-vmtf-wl-heur vm (set-trail-wl-heur M' S))›
     for M' :: trail-pol and vm

  have find-decomp-wl-st-int-alt-def:
    ‹find-decomp-wl-st-int = (λhighest S. do{
      (M', vm) ← isa-find-decomp-wl-imp (get-trail-wl-heur S) highest (get-vmtf-heur S);
      RETURN (upd M' vm S)
    })›
    unfolding upd-def find-decomp-wl-st-int-def
    by (auto intro!: ext)

  have [refine0]: ‹do {
   (M', vm) ←
     isa-find-decomp-wl-imp (get-trail-wl-heur S) 0 (get-vmtf-heur S);
   RETURN (upd M' vm S)
   } ≤ ⇓ {(S,
     T).
    (set-literals-to-update-wl-heur (isa-length-trail (get-trail-wl-heur S))
     (set-heur-wl-heur (restart-info-restart-done-heur (get-heur S)) S),
    (empty-Q-wl2 T)) ∈ twl-st-heur-restart-ana' r u ∧
    isa-length-trail-pre (get-trail-wl-heur S)} (SPEC (find-decomp-wl0 y))›
     (is ‹- ≤ ⇓ ?A -›)
    if
      ‹0 < count-decided-st-heur S› and
      ‹0 < count-decided (get-trail-wl y)›
  proof −
    have A:‹ ?A = {(S,
     T).
    (set-literals-to-update-wl-heur (length (get-trail-wl T))
     (set-heur-wl-heur (restart-info-restart-done-heur (get-heur S)) S),
    (empty-Q-wl2 T)) ∈ twl-st-heur-restart-ana' r u ∧
    isa-length-trail-pre (get-trail-wl-heur S)}›
    supply[[goals-limit=1]]
      apply (rule ; rule)
      subgoal for x
        apply clarify
 apply (frule twl-st-heur-restart-isa-length-trail-get-trail-wl)
        by (auto simp: trail-pol-def empty-Q-wl2-def mop-isa-length-trail-def)
      subgoal for x
        apply clarify
 apply (frule twl-st-heur-restart-isa-length-trail-get-trail-wl)
        by (auto simp: trail-pol-def empty-Q-wl2-def
            mop-isa-length-trail-def learned-clss-count-def)
```

```
    done

  let ?𝒜 = ‹all-init-atms-st y›
  have vm: ‹get-vmtf-heur S ∈ bump-heur ?𝒜 (get-trail-wl y)› (is ‹?vm ∈ -›) and
    n-d: ‹no-dup (get-trail-wl y)›
    using Sy
    by (auto simp: twl-st-heur-restart-def twl-st-heur-restart-ana-def
      all-init-atms-st-def get-unit-init-clss-wl-alt-def)

  have find-decomp-w-ns-pre:
    ‹find-decomp-w-ns-pre (all-init-atms-st y) ((get-trail-wl y, 0), ?vm)›
    using that assms vm unfolding find-decomp-w-ns-pre-def
    by (auto simp: twl-st-heur-restart-def twl-st-heur-restart-ana-def
      all-init-atms-st-def get-unit-init-clss-wl-alt-def
      dest: trail-pol-literals-are-in-𝓛_in-trail)
  have 1: ‹isa-find-decomp-wl-imp (get-trail-wl-heur S) 0 (get-vmtf-heur S) ≤
    ⇓ ({(M, M'). (M, M') ∈ trail-pol ?𝒜 ∧ count-decided M' = 0} ×_f Id)
      (find-decomp-w-ns ?𝒜 (get-trail-wl y) 0 vm')›
    apply (rule order-trans)
    apply (rule isa-find-decomp-wl-imp-find-decomp-wl-imp[THEN fref-to-Down-curry2,
      of ‹get-trail-wl y› 0 ?vm - - - ?𝒜])
    subgoal using that by auto
    subgoal
      using Sy vm
by (auto simp: twl-st-heur-restart-def twl-st-heur-restart-ana-def get-unit-init-clss-wl-alt-def
      all-init-atms-st-def)
    apply (rule order-trans, rule ref-two-step')
    apply (rule find-decomp-wl-imp-find-decomp-wl'[THEN fref-to-Down-curry2,
      of ?𝒜 ‹get-trail-wl y› 0 ?vm])
    subgoal by (rule find-decomp-w-ns-pre)
    subgoal by auto
    subgoal
      using n-d
      by (fastforce simp: find-decomp-w-ns-def conc-fun-RES Image-iff)
    done

 show ?thesis
    supply [[goals-limit=1]] unfolding A
    apply (rule bind-refine-res[OF - 1[unfolded find-decomp-w-ns-def conc-fun-RES]])
    apply (case-tac y, case S)
    apply clarify
    apply (rule RETURN-SPEC-refine)
    using assms
    by (auto simp: upd-def find-decomp-wl0-def
      intro!: RETURN-SPEC-refine clss-size-corr-simp simp: twl-st-heur-restart-def out-learned-def
    empty-Q-wl2-def twl-st-heur-restart-ana-def learned-clss-count-def
      all-init-atms-st-def
intro: isa-length-trail-pre dest: no-dup-appendD)
 qed
 have [simp]: ‹clss-size-corr-restart x1a x1c x1d NEk UEk x1e x1f x1g x1h (ck, cl, cd, cm, cn) ⟹
   clss-size-corr-restart x1a x1c x1d NEk UEk x1e {#} x1g {#} (ck, 0, cd, 0, 0)›
   for x1a x1c x1d x1e x1f x1g x1h ck cl cm cn NEk UEk cd
   by (auto simp: clss-size-corr-restart-def clss-size-def)

 have Sy': ‹(empty-US-heur S, empty-US-heur-wl y) ∈ twl-st-heur-restart-ana' r u›
   using Sy by (cases y; cases S; auto simp: twl-st-heur-restart-ana-def
```

empty-US-heur-wl-def twl-st-heur-restart-def empty-US-heur-def clss-size-resetUE-def
clss-size-lcountUEk-def
clss-size-resetUS-def clss-size-lcount-def clss-size-lcountU0-def clss-size-resetU0-def
learned-clss-count-def clss-size-def clss-size-lcountUS-def clss-size-lcountUE-def)
show *?thesis*
unfolding *find-decomp-wl-st-int-alt-def*
cdcl-twl-local-restart-wl-spec0-alt-def
apply *refine-vcg*
subgoal
using *Sy* **by** (*auto simp*: *twl-st-heur-restart-count-decided-st-alt-def*)
subgoal for *Sa T*
unfolding *empty-Q-def empty-Q-wl-def empty-US-heur-def empty-Q-wl2-def*
apply *clarify*
apply (*frule twl-st-heur-restart-isa-length-trail-get-trail-wl*)
by *refine-vcg*
(*cases ‹get-learned-count Sa›*, *auto simp add*: *mop-isa-length-trail-def twl-st-heur-restart-ana-def get-unit-init-clss-wl-alt-def*
twl-st-heur-restart-def learned-clss-count-def clss-size-resetUS-def clss-size-lcountUEk-def Let-def
clss-size-lcount-def clss-size-lcountU0-def clss-size-resetU0-def clss-size-resetUE-def
learned-clss-count-def clss-size-def clss-size-lcountUS-def clss-size-lcountUE-def)
subgoal
using *Sy′* **.**
done
qed

lemma *no-get-all-ann-decomposition-count-dec0*:
‹(∀ *M1*. (∀ *M2 K*. (*Decided K # M1*, *M2*) ∉ *set* (*get-all-ann-decomposition M*))) ⟷
count-decided M = 0›
apply (*induction M rule*: *ann-lit-list-induct*)
subgoal by *auto*
subgoal for *L M*
by *auto*
subgoal for *L C M*
by (*cases ‹get-all-ann-decomposition M›*) *fastforce+*
done

lemma *get-pos-of-level-in-trail-decomp-iff*:
assumes *‹no-dup M›*
shows *‹*((∃ *M1 M2 K*.
(*Decided K # M1*, *M2*)
∈ *set* (*get-all-ann-decomposition M*) ∧
count-decided M1 = 0 ∧ *k = length M1*)) ⟷
k < length M ∧ *count-decided M > 0* ∧ *is-decided* (*rev M* ! *k*) ∧ *get-level M* (*lit-of* (*rev M* ! *k*)) = *1›*
(**is** *‹?A ⟷ ?B›*)
proof
assume *?A*
then obtain *K M1 M2* **where**
decomp: *‹*(*Decided K # M1*, *M2*) ∈ *set* (*get-all-ann-decomposition M*)*›* **and**
[*simp*]: *‹count-decided M1 = 0›* **and**
k-M1: *‹length M1 = k›*
by *auto*
then have *‹k < length M›*
by *auto*
moreover have *‹rev M* ! *k = Decided K›*
using *decomp*

```
    by (auto dest!: get-all-ann-decomposition-exists-prepend
      simp: nth-append
      simp flip: k-M1)
  moreover have ‹get-level M (lit-of (rev M ! k)) = 1›
    using assms decomp
    by (auto dest!: get-all-ann-decomposition-exists-prepend
      simp: get-level-append-if nth-append
      simp flip: k-M1)
  ultimately show ?B
    using decomp by auto
next
  assume ?B
  define K where ‹K = lit-of (rev M ! k)›
  obtain M1 M2 where
    M: ‹M = M2 @ Decided K # M1› and
    k-M1: ‹length M1 = k›
    apply (subst (asm) append-take-drop-id[of ‹length M - Suc k›, symmetric])
    apply (subst (asm) Cons-nth-drop-Suc[symmetric])
    unfolding K-def
    subgoal using ‹?B› by auto
    subgoal using ‹?B› K-def by (cases ‹rev M ! k›) (auto simp: rev-nth)
    done
  moreover have ‹count-decided M1 = 0›
    using assms ‹?B› unfolding M
    by (auto simp: nth-append k-M1)
  ultimately show ?A
    using get-all-ann-decomposition-ex[of K M1 M2]
    unfolding M
    by auto
qed

lemma remove-all-learned-subsumed-clauses-wl-id:
  ‹(x2a, x2) ∈ twl-st-heur-restart-ana' r u ⟹
   RETURN (empty-US-heur x2a)
    ≤ ⇓ (twl-st-heur-restart-ana' r u)
      (remove-all-learned-subsumed-clauses-wl x2)›
   by (cases x2a; cases x2)
    (auto simp: twl-st-heur-restart-ana-def twl-st-heur-restart-def learned-clss-count-def
     remove-all-learned-subsumed-clauses-wl-def empty-US-heur-def
     intro: clss-size-corr-simp)

lemma mark-garbage-heur4-remove-and-add-cls-l:
  ‹(S, T) ∈ {(S, T). (S, T) ∈ twl-st-heur-restart-ana r ∧ learned-clss-count S ≤ u} ⟹ (C, C') ∈ Id
⟹
    mark-garbage-heur4 C S
      ≤ ⇓ {(S, T). (S, T) ∈ twl-st-heur-restart-ana r ∧ learned-clss-count S ≤ u}
          (remove-and-add-cls-wl C' T)›
  unfolding mark-garbage-heur4-def remove-and-add-cls-wl-def Let-def
  apply (cases T, hypsubst, unfold prod.case)
  apply refine-rcg
  subgoal
   by (auto simp add: twl-st-heur-restart-def twl-st-heur-restart-ana-def arena-lifting get-unit-init-clss-wl-alt-def
      dest!: clss-size-corr-restart-rew multi-member-split simp: ran-m-def)
  subgoal
    supply [[goals-limit=1]]
    apply (auto simp: learned-clss-count-def)
```

apply (*clarsimp simp add*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*
arena-lifting isasat-state-simp
valid-arena-extra-information-mark-to-delete'
all-init-atms-fmdrop-add-mset-unit learned-clss-l-l-fmdrop
learned-clss-l-l-fmdrop-irrelev aivdom-inv-dec-remove-clause
size-Diff-singleton red-in-dom-number-of-learned-ge1 learned-clss-count-def
clss-size-corr-restart-intro clss-size-corr-restart-simp3
dest: *in-vdom-m-fmdropD dest*: *in-diffD*
intro: *clss-size-corr-restart-intro*)
apply (*auto simp add*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*
arena-lifting
valid-arena-extra-information-mark-to-delete'
all-init-atms-fmdrop-add-mset-unit learned-clss-l-l-fmdrop
learned-clss-l-l-fmdrop-irrelev aivdom-inv-dec-remove-clause
size-Diff-singleton red-in-dom-number-of-learned-ge1 learned-clss-count-def
clss-size-corr-restart-intro clss-size-corr-restart-simp3
dest: *in-vdom-m-fmdropD dest*: *in-diffD*
intro: *clss-size-corr-restart-intro*)
done
done

lemma *remove-one-annot-true-clause-one-imp-wl-pre-fst-le-uint32*:
assumes ‹(*x*, *y*) ∈ *nat-rel* ×$_f$ {*p*. (*fst p*, *snd p*) ∈ *twl-st-heur-restart-ana r* ∧
learned-clss-count (*fst p*) ≤ *u*}› **and**
‹*remove-one-annot-true-clause-one-imp-wl-pre* (*fst y*) (*snd y*)›
shows ‹*fst x* + *1* ≤ *Suc* (*unat32-max div 2*)›
proof −
have [*simp*]: ‹*fst y* = *fst x*›
using *assms* **by** (*cases x*, *cases y*) *auto*
have ‹*fst x* < *length* (*get-trail-wl* (*snd y*))›
using *assms* **apply** −
unfolding
remove-one-annot-true-clause-one-imp-wl-pre-def
remove-one-annot-true-clause-one-imp-pre-def
by *normalize-goal*+ *auto*
moreover have ‹(*get-trail-wl-heur* (*snd x*), *get-trail-wl* (*snd y*)) ∈ *trail-pol* (*all-init-atms-st* (*snd y*))›
using *assms*
by (*cases x*, *cases y*) (*simp add*: *twl-st-heur-restart-ana-def*
twl-st-heur-restart-def all-init-atms-st-def)
ultimately show ‹*?thesis*›
by (*auto simp add*: *trail-pol-alt-def*)
qed

lemma *remove-one-annot-true-clause-one-imp-wl-alt-def*:
‹*remove-one-annot-true-clause-one-imp-wl* = (λ*i S*. *do* {
ASSERT(*remove-one-annot-true-clause-one-imp-wl-pre i S*);
ASSERT(*is-proped* (*rev* (*get-trail-wl S*) ! *i*));
(*L*, *C*) ← *SPEC*(λ(*L*, *C*). (*rev* (*get-trail-wl S*))!*i* = *Propagated L C*);
ASSERT(*Propagated L C* ∈ *set* (*get-trail-wl S*));
ASSERT(*L* ∈# *all-init-lits-of-wl S*);
if C = *0 then RETURN* (*i*+*1*, *S*)
else do {
ASSERT(*C* ∈# *dom-m* (*get-clauses-wl S*));
S ← *replace-annot-wl L C S*;
- ← *RETURN* (*log-clause S C*);
S ← *remove-and-add-cls-wl C S*;

```
      RETURN (i+1, S)
    }
  })›
  by (auto simp: remove-one-annot-true-clause-one-imp-wl-def log-clause-def cong: if-cong)


lemma log-clause-heur-log-clause2-ana:
  assumes ‹(S,T) ∈ twl-st-heur-restart-ana' r u› ‹(C,C') ∈ nat-rel›
  shows ‹log-clause-heur S C ≤⇓unit-rel (log-clause2 T C')›
proof −
  have [refine0]: ‹(0,0)∈nat-rel›
    by auto
  have length: ‹⇓ nat-rel ((RETURN ∘ (λc. length (get-clauses-wl T ∝ c))) C') ≤ SPEC (λc. (c, length
(get-clauses-wl T ∝ C')) ∈ {(a,b). a=b ∧ a = length (get-clauses-wl T ∝ C)})›
    by (use assms in auto)
  show ?thesis
    unfolding log-clause-heur-def log-clause2-def comp-def uncurry-def mop-arena-length-st-def
      mop-access-lit-in-clauses-heur-def
     apply (refine-vcg mop-arena-lit[where vdom = ‹set (get-vdom S)› and N = ‹get-clauses-wl T›,
THEN order-trans]
     mop-arena-length[where vdom = ‹set (get-vdom S)›, THEN fref-to-Down-curry, THEN order-trans,
unfolded prod.simps])
    apply assumption
    subgoal using assms by (auto simp: twl-st-heur-restart-ana-def twl-st-heur-restart-def)
    apply (rule length)
   subgoal by (use assms in ‹auto simp: twl-st-heur-restart-ana-def twl-st-heur-restart-def dest: arena-lifting(10)›)
    subgoal by auto
    subgoal using assms by (auto simp: twl-st-heur-restart-ana-def twl-st-heur-restart-def)
    apply assumption
    subgoal by (use assms in auto)
    apply (rule refl)
    subgoal by auto
    by auto
qed

lemma log-del-clause-heur-log-clause:
  assumes ‹(S,T) ∈ twl-st-heur-restart-ana' r u› ‹(C,C') ∈ nat-rel› ‹C ∈# dom-m (get-clauses-wl T)›
  shows ‹log-del-clause-heur S C ≤ SPEC (λc. (c, log-clause T C') ∈ unit-rel)›
    unfolding log-del-clause-heur-alt-def
    apply (rule log-clause-heur-log-clause2-ana[THEN order-trans, OF assms(1,2)])
    apply (rule order-trans)
    apply (rule ref-two-step')
    apply (rule log-clause2-log-clause[THEN fref-to-Down-curry])
    using assms by auto

lemma remove-one-annot-true-clause-one-imp-wl-D-heur-remove-one-annot-true-clause-one-imp-wl-D:
  ‹(uncurry remove-one-annot-true-clause-one-imp-wl-D-heur,
    uncurry remove-one-annot-true-clause-one-imp-wl) ∈
    nat-rel ×f twl-st-heur-restart-ana' r u →f
    ⟨nat-rel ×f twl-st-heur-restart-ana' r u⟩nres-rel›
  unfolding remove-one-annot-true-clause-one-imp-wl-D-heur-def
    remove-one-annot-true-clause-one-imp-wl-alt-def case-prod-beta uncurry-def
  apply (intro frefI nres-relI)
  subgoal for x y
  apply (refine-rcg get-literal-and-reason[where r=r]
    isasat-replace-annot-in-trail-replace-annot-in-trail-spec
```

[**where** *r*=*r* **and** *u*=*u*] *log-del-clause-heur-log-clause*
mark-garbage-heur4-remove-and-add-cls-l[**where** *r*=*r* **and** *u*=*u*])
subgoal
by (*auto simp*: *prod-rel-fst-snd-iff*)
subgoal unfolding *remove-one-annot-true-clause-one-imp-wl-pre-def*
by *auto*
subgoal
by (*rule remove-one-annot-true-clause-one-imp-wl-pre-fst-le-uint32*)
subgoal for *p pa*
by (*cases pa*)
(*auto simp*: *all-init-atms-def simp del*: *all-init-atms-def*[*symmetric*])
subgoal
by (*cases x*, *cases y*)
(*fastforce simp*: *twl-st-heur-restart-def*
trail-pol-alt-def)+
subgoal by *auto*
subgoal for *p pa*
by (*cases pa*; *cases p*; *cases x*; *cases y*)
(*auto simp*: *all-init-atms-def learned-clss-count-def simp del*: *all-init-atms-def*[*symmetric*])
apply (*solves auto*)
subgoal by *auto*
subgoal *in-dom-m* **for** *p pa S Sa*
unfolding *mark-garbage-pre-def*
arena-is-valid-clause-idx-def
prod.case
apply (*case-tac Sa*; *cases y*)
apply (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*)
done
subgoal for *p pa S Sa*
using *in-dom-m*
unfolding *mark-garbage-pre-def*
arena-is-valid-clause-idx-def
prod.case
apply (*rule-tac x* = ‹*get-clauses-wl Sa*› **in** *exI*)
apply (*rule-tac x* = ‹*set* (*get-vdom S*)› **in** *exI*)
apply (*case-tac Sa*; *cases y*)
apply (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*)
done
subgoal for *p pa S Sa*
unfolding *arena-is-valid-clause-vdom-def*
apply (*rule-tac x* = ‹*get-clauses-wl Sa*› **in** *exI*)
apply (*rule-tac x* = ‹*set* (*get-vdom S*)› **in** *exI*)
apply (*case-tac Sa*; *cases y*)
apply (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)
done
subgoal
by (*auto simp*: *prod-rel-fst-snd-iff dest*: *get-learned-count-learned-clss-countD*)
subgoal
by (*auto simp*: *prod-rel-fst-snd-iff dest*: *get-learned-count-learned-clss-countD*)
subgoal
by *auto*
subgoal
by (*cases x*, *cases y*) *fastforce*
done
done

lemma *remove-one-annot-trail-zeroed-until-state*:
assumes
‹(*x*, *y*) ∈ *twl-st-heur-restart-ana′ r u*› **and**
‹(*isa-length-trail-pre* ∘ *get-trail-wl-heur*) *x*› **and**
‹(*k*, *ka*) ∈ *nat-rel*› **and**
ka: ‹*ka* ∈ {*k*. (∃ *M1 M2 K*.
(*Decided K* # *M1*, *M2*)
∈ *set* (*get-all-ann-decomposition* (*get-trail-wl y*)) ∧
count-decided M1 = *0* ∧ *k* = *length M1*) ∨
count-decided (*get-trail-wl y*) = *0* ∧ *k* = *length* (*get-trail-wl y*)}›
shows ‹*trail-zeroed-until-state x* ≤ *ka*› (**is** *?A*) **and**
‹*j* < *trail-zeroed-until-state x* ⟹ *is-proped* (*rev* (*get-trail-wl y*) ! *j*)› (**is** ‹*?X* ⟹ *?B*›)**and**
‹*j* < *trail-zeroed-until-state x* ⟹ *mark-of* (*rev* (*get-trail-wl y*) ! *j*) = *0*› (**is** ‹*?X* ⟹ *?C*›)
proof −
define *A* **where** ‹*A* = (*all-init-atms* (*get-clauses-wl y*)
(*IsaSAT-Setup.get-unkept-unit-init-clss-wl y* +
IsaSAT-Setup.get-kept-unit-init-clss-wl y +
get-subsumed-init-clauses-wl y +
get-init-clauses0-wl y))›
have ‹(*get-trail-wl-heur x*, *get-trail-wl y*) ∈ *trail-pol A*›
using *assms*(*1*) **unfolding** *twl-st-heur-restart-ana-def twl-st-heur-restart-def A-def*
by *fast*
then show *?A*
‹*?X* ⟹ *?B*›
‹*?X* ⟹ *?C*›
using *ka*
apply (*auto simp*: *trail-pol-def zeroed-trail-def trail-zeroed-until-state-def*
trail-zeroed-until-def)
by (*meson annotated-lit.distinct-disc*(*1*) *get-pos-of-level-in-trail-decomp-iff leI*)
qed

lemma *remove-one-annot-true-clause-imp-wl-alt-def*:
‹*remove-one-annot-true-clause-imp-wl* = (λ*S*. *do* {
k ← *SPEC*(λ*k*. (∃ *M1 M2 K*. (*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*get-trail-wl* *S*)) ∧
count-decided M1 = *0* ∧ *k* = *length M1*)
∨ (*count-decided* (*get-trail-wl S*) = *0* ∧ *k* = *length* (*get-trail-wl S*)));
start ← *SPEC* (λ*i*. *i* ≤ *k* ∧ (∀ *j* < *i*. *is-proped* (*rev* (*get-trail-wl S*) ! *j*) ∧ *mark-of* (*rev* (*get-trail-wl* *S*) ! *j*) = *0*));
(*i*, *T*) ← *WHILE*$_T$*remove-one-annot-true-clause-imp-wl-inv S*
(λ(*i*, *S*). *i* < *k*)
(λ(*i*, *S*). *remove-one-annot-true-clause-one-imp-wl i S*)
(*start*, *S*);
ASSERT (*remove-one-annot-true-clause-imp-wl-inv S* (*i*, *T*));
T ← *RETURN T*;
remove-all-learned-subsumed-clauses-wl T
})›
unfolding *remove-one-annot-true-clause-imp-wl-def Let-def* **by** *auto*

definition *remove-one-annot-true-clause-imp-wl-D-heur* :: ‹*isasat* ⇒ *isasat nres*›
where
‹*remove-one-annot-true-clause-imp-wl-D-heur* = (λ*S*. *do* {
ASSERT((*isa-length-trail-pre o get-trail-wl-heur*) *S*);
k ← (*if count-decided-st-heur S* = *0*
then RETURN (*isa-length-trail* (*get-trail-wl-heur S*))
else get-pos-of-level-in-trail-imp (*get-trail-wl-heur S*) *0*);

let start = trail-zeroed-until-state S;
(i, T) ← WHILE$_T$*remove-one-annot-true-clause-imp-wl-D-heur-inv S*
(λ(i, S). i < k)
(λ(i, S). remove-one-annot-true-clause-one-imp-wl-D-heur i S)
(start, S);
ASSERT (remove-one-annot-true-clause-imp-wl-D-heur-inv S (i, T));
T ← RETURN (trail-set-zeroed-until-state i T);
RETURN (empty-US-heur T)
})›

lemma *remove-one-annot-true-clause-trail-set-zeroed-until-state*:
‹*(x, y) ∈ twl-st-heur-restart-ana′ r u ⟹*
(xa, x′) ∈ nat-rel ×$_f$ *twl-st-heur-restart-ana′ r (learned-clss-count x) ⟹*
x′ = (x1, x2) ⟹
xa = (x1a, x2a) ⟹
remove-one-annot-true-clause-imp-wl-inv y (x1, x2) ⟹
remove-one-annot-true-clause-imp-wl-D-heur-inv x (x1a, x2a) ⟹
(trail-set-zeroed-until-state x1a x2a, x2)
∈ twl-st-heur-restart-ana′ r u›
supply [[*goals-limit=1*]]
apply (*simp add*: *twl-st-heur-restart-ana-def trail-set-zeroed-until-state-def*
twl-st-heur-restart-def)
apply (*rule trail-set-zeroed-until-rel*)
apply (*solves simp*)
unfolding *remove-one-annot-true-clause-imp-wl-D-heur-inv-def*
remove-one-annot-true-clause-imp-wl-inv-def remove-one-annot-true-clause-imp-inv-def
prod.simps
apply *normalize-goal+*
apply (*simp add*: *zeroed-trail-def*)
done

find-theorems *trail-set-zeroed-until*

lemma *remove-one-annot-true-clause-imp-wl-D-heur-remove-one-annot-true-clause-imp-wl-D*:
‹(*remove-one-annot-true-clause-imp-wl-D-heur*, *remove-one-annot-true-clause-imp-wl*) ∈
twl-st-heur-restart-ana′ r u →$_f$
⟨*twl-st-heur-restart-ana′ r u*⟩*nres-rel*›
(**is** ‹- ∈ *?A →*$_f$ -›)
unfolding *remove-one-annot-true-clause-imp-wl-alt-def*
remove-one-annot-true-clause-imp-wl-D-heur-def
apply (*intro frefI nres-relI*)
subgoal for *x y*
apply (*refine-vcg*
WHILEIT-refine[**where** *R* = ‹*nat-rel ×*$_r$ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart-ana r* ∧ *learned-clss-count S* ≤ *learned-clss-count x*}›]
remove-one-annot-true-clause-one-imp-wl-D-heur-remove-one-annot-true-clause-one-imp-wl-D[*THEN fref-to-Down-curry*])
subgoal by (*auto simp*: *trail-pol-alt-def isa-length-trail-pre-def*
twl-st-heur-restart-def twl-st-heur-restart-ana-def)
subgoal by (*auto dest*: *twl-st-heur-restart-isa-length-trail-get-trail-wl*
simp: *twl-st-heur-restart-count-decided-st-alt-def mop-isa-length-trail-def*)
subgoal
apply (*rule order-trans*)
apply (*rule get-pos-of-level-in-trail-imp-get-pos-of-level-in-trail-CS*[*THEN fref-to-Down-curry*,
of ‹*get-trail-wl y*› *0* - - ‹*all-init-atms-st y*›])

subgoal by (*auto simp*: *get-pos-of-level-in-trail-pre-def*
twl-st-heur-restart-count-decided-st-alt-def)
subgoal by (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*
all-init-atms-st-def get-unit-init-clss-wl-alt-def)
subgoal
apply (*subst get-pos-of-level-in-trail-decomp-iff*)
apply (*solves* ‹*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*›)
apply (*auto simp*: *get-pos-of-level-in-trail-def*
twl-st-heur-restart-count-decided-st-alt-def)
done
done
subgoal by (*rule remove-one-annot-trail-zeroed-until-state*)
subgoal by (*rule remove-one-annot-trail-zeroed-until-state*)
subgoal by (*rule remove-one-annot-trail-zeroed-until-state*)
subgoal by *auto*
subgoal for *k k′ start T T′*
apply (*subst* (*asm*)(*11*) *surjective-pairing*)
apply (*subst* (*asm*)(*15*) *surjective-pairing*)
unfolding *remove-one-annot-true-clause-imp-wl-D-heur-inv-def*
prod-rel-iff
apply (*subst* (*8*) *surjective-pairing*, *subst prod.case*)
apply (*rule-tac x=y* **in** *exI*)
apply (*rule-tac x=* ‹*snd T′*› **in** *exI*)
by (*auto intro*: *twl-st-heur-restart-anaD simp*: *prod-rel-fst-snd-iff twl-st-heur-restart-anaD*)
subgoal by *auto*
subgoal by *auto*
subgoal for *k k′ start T T′*
apply (*subst* (*asm*)(*11*) *surjective-pairing*)
apply (*subst* (*asm*)(*15*) *surjective-pairing*)
unfolding *remove-one-annot-true-clause-imp-wl-D-heur-inv-def*
prod-rel-iff
apply (*subst* (*8*) *surjective-pairing*, *subst prod.case*)
apply (*rule-tac x=y* **in** *exI*)
apply (*rule-tac x=* ‹*snd T′*› **in** *exI*)
by (*auto intro*: *twl-st-heur-restart-anaD simp*: *prod-rel-fst-snd-iff twl-st-heur-restart-anaD*)
apply (*rule remove-one-annot-true-clause-trail-set-zeroed-until-state*)
apply *assumption+*
subgoal for *k ka start xa x′ x1 x2*
by (*auto intro*!: *remove-all-learned-subsumed-clauses-wl-id*)
done
done

lemmas *iterate-over-VMTF-def* =
iterate-over-VMTF-alt-def[*unfolded iterate-over-VMTFC-def simp-thms*]

definition *iterate-over-*$\mathcal{L}_{all}$*C* **where**
‹*iterate-over-*$\mathcal{L}_{all}$*C* = (λ*f* $\mathcal{A}_0$ *I P x*. *do* {
$\mathcal{A}$ ← *SPEC*(λ$\mathcal{A}$. *set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{A}_0$ ∧ *distinct-mset* $\mathcal{A}$);
(-, *x*) ← *WHILE*$_T$$^{\lambda(\mathcal{A},\, x).\ I\ \mathcal{A}\ x}$
(λ($\mathcal{B}$, *x*). $\mathcal{B}$ ≠ {#} ∧ *P x*)
(λ($\mathcal{B}$, *x*). *do* {
ASSERT($\mathcal{B}$ ≠ {#});
A ← *SPEC* (λ*A*. *A* ∈# $\mathcal{B}$);
x ← *f A x*;
RETURN (*remove1-mset A* $\mathcal{B}$, *x*)

```
    })
    (A, x);
  RETURN x
})›

definition iterate-over-L_all :: ‹-› where
  iterate-over-L_all-alt-def: ‹iterate-over-L_all f A I = iterate-over-L_all C f A I (λ-. True)›

lemmas iterate-over-L_all-def =
  iterate-over-L_all-alt-def[unfolded iterate-over-L_all C-def simp-thms]


lemma iterate-over-VMTFC-iterate-over-L_all C:
  fixes x :: 'a
  assumes vmtf: ‹(ns, m, fst-As, lst-As, next-search) ∈ vmtf A M› and
    nempty: ‹A ≠ {#}› ‹isasat-input-bounded A› and
    II': ‹⋀x B. set-mset B ⊆ set-mset A ⟹ I' B x ⟹ I x› and
    ‹⋀x. I x ⟹ P x = Q x›
  shows ‹iterate-over-VMTFC f I P (ns, Some fst-As) x ≤ ⇓ Id (iterate-over-L_all C f A I' Q x)›
proof −
  obtain xs' ys' where
    vmtf-ns: ‹vmtf-ns (ys' @ xs') m ns› and
    ‹fst-As = hd (ys' @ xs')› and
    ‹lst-As = last (ys' @ xs')› and
    vmtf-L: ‹vmtf-L_all A M (set xs', set ys')› and
    fst-As: ‹fst-As = hd (ys' @ xs')› and
    le: ‹∀ L∈atms-of (L_all A). L < length ns›
    using vmtf unfolding vmtf-def
    by blast
  define zs where ‹zs = ys' @ xs'›
  define is-lasts where
    ‹is-lasts B n m ⟷ set-mset B = set (drop m zs) ∧ set-mset B ⊆ set-mset A ∧
        distinct-mset B ∧
        card (set-mset B) ≤ length zs ∧
        card (set-mset B) + m = length zs ∧
        (n = option-hd (drop m zs)) ∧
        m ≤ length zs› for B and n :: ‹nat option› and m
  have card-A: ‹card (set-mset A) = length zs›
    ‹set-mset A = set zs› and
    nempty': ‹zs ≠ []› and
    dist-zs: ‹distinct zs›
    using vmtf-L vmtf-ns-distinct[OF vmtf-ns] nempty
    unfolding vmtf-L_all-def eq-commute[of - ‹atms-of -›] zs-def
    by (auto simp: atms-of-L_all-A_in card-Un-disjoint distinct-card)
  have hd-zs-le: ‹hd zs < length ns›
    using vmtf-ns-le-length[OF vmtf-ns, of ‹hd zs›] nempty'
    unfolding zs-def[symmetric]
    by auto
  have [refine0]: ‹
      (the x1a, A) ∈ nat-rel ⟹
      x = x2b ⟹
      f (the x1a) x2b ≤ ⇓ Id (f A x)› for x1a A x x2b
    by auto
  define iterate-over-VMTF2 where
    ‹iterate-over-VMTF2 ≡ (λf (I :: 'a ⇒ bool) (vm :: (nat, nat) vmtf-node list, n) x. do {
      let - = remdups-mset A;
```

```
    (-, -, x) ← WHILE_T^λ(n,m, x). I x
      (λ(n, -, x). n ≠ None ∧ P x)
      (λ(n, m, x). do {
        ASSERT(n ≠ None);
        let A = the n;
        ASSERT(A < length ns);
        ASSERT(A ≤ unat32-max div 2);
        x ← f A x;
        RETURN (get-next ((ns ! A)), Suc m, x)
      })
      (n, 0, x);
    RETURN x
  })›
have iterate-over-VMTF2-alt-def:
  ‹iterate-over-VMTF2 ≡ (λf (I :: 'a ⇒ bool) (vm :: (nat, nat) vmtf-node list, n) x. do {
    (-, -, x) ← WHILE_T^λ(n,m, x). I x
      (λ(n, -, x). n ≠ None ∧ P x)
      (λ(n, m, x). do {
        ASSERT(n ≠ None);
        let A = the n;
        ASSERT(A < length ns);
        ASSERT(A ≤ unat32-max div 2);
        x ← f A x;
        RETURN (get-next ((ns ! A)), Suc m, x)
      })
      (n, 0, x);
    RETURN x
  })›
  unfolding iterate-over-VMTF2-def by force
have nempty-iff: ‹(x1 ≠ None ∧ P x2a) = (x1b ≠ {#} ∧ Q xb)› (is ‹?A = ?B›)
if
  ‹(remdups-mset A, A') ∈ Id› and
  H: ‹(x, x') ∈ {((n, m, x), A', y). is-lasts A' n m ∧ x = y}› and
  ‹case x of (n, m, xa) ⇒ I xa› and
  ‹case x' of (A', x) ⇒ I' A' x› and
  st[simp]:
    ‹x2 = (x1a, x2a)›
    ‹x = (x1, x2)›
    ‹x' = (x1b, xb)›
  for A' x x' x1 x2 x1a x2a x1b xb
proof
  have KK: ‹P x2a = Q xb›
    by (subst assms) (use that in auto)
  show ?A if ?B
    using that H unfolding KK
    by (auto simp: is-lasts-def)
  show ?B if ?A
    using that H unfolding KK
    by (auto simp: is-lasts-def)
qed
have IH: ‹((get-next (ns ! the x1a), Suc x1b, xa), remove1-mset A x1, xb)
     ∈ {((n, m, x), A', y). is-lasts A' n m ∧ x = y}›
   if
    ‹(remdups-mset A, A') ∈ Id› and
    H: ‹(x, x') ∈ {((n, m, x), A', y). is-lasts A' n m ∧ x = y}› and
    ‹case x of (n, uu-, x) ⇒ n ≠ None ∧ P x› and
```

nempty: ‹*case x′ of* (*B*, *x*) ⇒ *B* ≠ {#} ∧ *Q x*› **and**
‹*case x of* (*n*, *m*, *xa*) ⇒ *I xa*› **and**
‹*case x′ of* (*A′*, *x*) ⇒ *I′ A′ x*› **and**
st:
‹*x′* = (*x1*, *x2*)›
‹*x2a* = (*x1b*, *x2b*)›
‹*x* = (*x1a*, *x2a*)›
‹(*xa*, *xb*) ∈ *Id*› **and**
‹*x1* ≠ {#}› **and**
‹*x1a* ≠ *None*› **and**
A: ‹(*the x1a*, *A*) ∈ *nat-rel*› **and**
‹*the x1a* < *length ns*›
for *A′ x x′ x1 x2 x1a x2a x1b x2b A xa xb*
proof −
have [*simp*]: ‹*distinct-mset x1*› ‹*x1b* < *length zs*›
using *H A nempty*
apply (*auto simp*: *st is-lasts-def simp flip*: *Cons-nth-drop-Suc*)
apply (*cases* ‹*x1b* = *length zs*›)
apply *auto*
done
then have [*simp*]: ‹*zs* ! *x1b* ∉ *set* (*drop* (*Suc x1b*) *zs*)›
by (*auto simp*: *in-set-drop-conv-nth nth-eq-iff-index-eq dist-zs*)
have [*simp*]: ‹*length zs* − *Suc x1b* + *x1b* = *length zs* ⟷ *False*›
using ‹*x1b* < *length zs*› **by** *presburger*
have ‹*vmtf-ns* (*take x1b zs* @ *zs* ! *x1b* # *drop* (*Suc x1b*) *zs*) *m ns*›
using *vmtf-ns*
by (*auto simp*: *Cons-nth-drop-Suc simp flip*: *zs-def*)
from *vmtf-ns-last-mid-get-next-option-hd*[*OF this*]
show *?thesis*
using *H A st*
by (*auto simp*: *st is-lasts-def dist-zs distinct-card distinct-mset-set-mset-remove1-mset*
simp flip: *Cons-nth-drop-Suc*)
qed
have *WTF*[*simp*]: ‹*length zs* − *Suc 0* = *length zs* ⟷ *zs* = []›
by (*cases zs*) *auto*
have *zs2*: ‹*set* (*xs′* @ *ys′*) = *set zs*›
by (*auto simp*: *zs-def*)
have *is-lasts-le*: ‹*is-lasts x1* (*Some A*) *x1b* ⟹ *A* < *length ns*› **for** *x2 xb x1b x1 A*
using *vmtf-L le nth-mem*[*of* ‹*x1b*› *zs*] **unfolding** *is-lasts-def prod.case vmtf-L*$_{all}$*-def*
set-append[*symmetric*]*zs-def*[*symmetric*] *zs2*
by (*auto simp*: *eq-commute*[*of* ‹*set zs*› ‹*atms-of* (*L*$_{all}$ *A*)›] *hd-drop-conv-nth*
simp del: *nth-mem*)
have *le-unat32-max*: ‹*the x1a* ≤ *unat32-max div 2*›
if
‹(*remdups-mset A*, *A′*) ∈ *Id*› **and**
‹(*x*, *x′*) ∈ {((*n*, *m*, *x*), *A′*, *y*). *is-lasts A′ n m* ∧ *x* = *y*}› **and**
‹*case x of* (*n*, *uu-*, *x*) ⇒ *n* ≠ *None* ∧ *P x*› **and**
‹*case x′ of* (*B*, *x*) ⇒ *B* ≠ {#} ∧ *Q x*› **and**
‹*case x of* (*n*, *m*, *xa*) ⇒ *I xa*› **and**
‹*case x′ of* (*A′*, *x*) ⇒ *I′ A′ x*› **and**
‹*x′* = (*x1*, *x2*)› **and**
‹*x2a* = (*x1b*, *xb*)› **and**
‹*x* = (*x1a*, *x2a*)› **and**
‹*x1* ≠ {#}› **and**
‹*x1a* ≠ *None*› **and**
‹(*the x1a*, *A*) ∈ *nat-rel*› **and**

```
    ‹the x1a < length ns›
  for 𝒜' x x' x1 x2 x1a x2a x1b xb A
 proof −
  have ‹the x1a ∈# 𝒜›
   using that unfolding is-lasts-def
   by clarsimp (auto simp: is-lasts-def)
  then show ?thesis
   using nempty by (auto dest!: multi-member-split simp: ℒall-add-mset)
 qed
 have ‹iterate-over-VMTF2 f I (ns, Some fst-As) x ≤ ⇓ Id (iterate-over-ℒall C f 𝒜 I' Q x)›
  unfolding iterate-over-VMTF2-def iterate-over-ℒall C-def prod.case
  apply (refine-vcg WHILEIT-refine[where R = ‹{((n :: nat option, m::nat, x::'a), (𝒜' :: nat multiset,
y)).
      is-lasts 𝒜' n m ∧ x = y}›])
  subgoal by simp
  subgoal by simp
  subgoal
   using card-𝒜 fst-As nempty nempty' hd-conv-nth[OF nempty'] hd-zs-le unfolding zs-def[symmetric]
     is-lasts-def
    by (simp-all add:  eq-commute[of ‹remdups-mset -›])
  subgoal for 𝒜' x x' x1 x2 x1a xaa using II'[of ‹-› xaa] unfolding is-lasts-def by auto
  subgoal for 𝒜' x x' x1 x2 x1a x2a x1b xb
   by (rule nempty-iff)
  subgoal by auto
  subgoal for 𝒜' x x' x1 x2 x1a x2a x1b xb
   by (simp add: is-lasts-def in-set-dropI)
  subgoal for 𝒜' x x' x1 x2 x1a x2a x1b xb
   by (auto simp: is-lasts-le)
  subgoal by (rule le-unat32-max)
  subgoal by auto
  subgoal for 𝒜' x x' x1 x2 x1a x2a x1b x2b A xa xb
   by (rule IH)
  subgoal by auto
  done
 moreover have ‹iterate-over-VMTFC f I P (ns, Some fst-As) x ≤ ⇓ Id (iterate-over-VMTF2 f I (ns,
Some fst-As) x)›
  unfolding iterate-over-VMTF2-alt-def iterate-over-VMTFC-def prod.case
   by (refine-vcg WHILEIT-refine[where R = ‹{((n :: nat option, x::'a), (n' :: nat option, m'::nat,
x'::'a)).
      n = n' ∧ x = x'}›]) auto
 ultimately show ?thesis
  by simp
qed


lemma iterate-over-VMTF-iterate-over-ℒall:
 fixes x :: 'a
 assumes vmtf: ‹(ns, m, fst-As, lst-As, next-search) ∈ vmtf 𝒜 M› and
  nempty: ‹𝒜 ≠ {#}› ‹isasat-input-bounded 𝒜› ‹⋀x ℬ. set-mset ℬ ⊆ set-mset 𝒜 ⟹ I' ℬ x ⟹ I x›
 shows ‹iterate-over-VMTF f I (ns, Some fst-As) x ≤ ⇓ Id (iterate-over-ℒall f 𝒜 I' x)›
 unfolding iterate-over-VMTF-alt-def iterate-over-ℒall-alt-def
 apply (rule iterate-over-VMTFC-iterate-over-ℒall C[OF assms])
 by auto


definition arena-is-packed :: ‹arena ⇒ nat clauses-l ⇒ bool› where
```

‹*arena-is-packed arena N* ⟷ *length arena* = (∑ *C* ∈# *dom-m N*. *length* (*N* ∝ *C*) + *header-size* (*N* ∝ *C*))›

lemma *arena-is-packed-empty*[*simp*]: ‹*arena-is-packed* [] *fmempty*›
by (*auto simp*: *arena-is-packed-def*)

lemma *arena-is-packed-append*:
assumes ‹*arena-is-packed* (*arena*) *N*› **and**
[*simp*]: ‹*length C* = *length* (*fst C'*) + *header-size* (*fst C'*)› **and**
[*simp*]: ‹*a* ∉# *dom-m N*›
shows ‹*arena-is-packed* (*arena* @ *C*) (*fmupd a C' N*)›
using *assms*(*1*) **by** (*auto simp*: *arena-is-packed-def*
intro!: *sum-mset-cong*)

lemma *arena-is-packed-append-valid*:
assumes
in-dom: ‹*fst C* ∈# *dom-m x1a*› **and**
valid0: ‹*valid-arena x1c x1a vdom0*› **and**
valid: ‹*valid-arena x1d x2a* (*set x2d*)› **and**
packed: ‹*arena-is-packed x1d x2a*› **and**
n: ‹*n* = *header-size* (*x1a* ∝ (*fst C*))›
shows ‹*arena-is-packed*
(*x1d* @
Misc.slice (*fst C* − *n*)
(*fst C* + *arena-length x1c* (*fst C*)) *x1c*)
(*fmupd* (*length x1d* + *n*) (*the* (*fmlookup x1a* (*fst C*))) *x2a*)›
proof −
have [*simp*]: ‹*length x1d* + *n* ∉# *dom-m x2a*›
using *valid* **by** (*auto dest*: *arena-lifting*(*2*) *valid-arena-in-vdom-le-arena*
simp: *arena-is-valid-clause-vdom-def header-size-def*)
have [*simp*]: ‹*arena-length x1c* (*fst C*) = *length* (*x1a* ∝ (*fst C*))› ‹*fst C* ≥ *n*›
‹*fst C* − *n* < *length x1c*› ‹*fst C* < *length x1c*›
using *valid0 valid in-dom* **by** (*auto simp*: *arena-lifting n less-imp-diff-less*)
have [*simp*]: ‹*length*
(*Misc.slice* (*fst C* − *n*)
(*fst C* + *length* (*x1a* ∝ (*fst C*))) *x1c*) =
length (*x1a* ∝ *fst C*) + *header-size* (*x1a* ∝ *fst C*)›
using *valid in-dom arena-lifting*(*10*)[*OF valid0*]
by (*fastforce simp*: *slice-len-min-If min-def arena-lifting*(*4*) *simp flip*: *n*)
show *?thesis*
by (*rule arena-is-packed-append*[*OF packed*]) *auto*
qed

definition *move-is-packed* :: ‹*arena* ⇒ - ⇒ *arena* ⇒ - ⇒ *bool*› **where**
‹*move-is-packed arena*$_o$ *N*$_o$ *arena N* ⟷
((∑ *C*∈#*dom-m N*$_o$. *length* (*N*$_o$ ∝ *C*) + *header-size* (*N*$_o$ ∝ *C*)) +
(∑ *C*∈#*dom-m N*. *length* (*N* ∝ *C*) + *header-size* (*N* ∝ *C*)) ≤ *length arena*$_o$)›

lemma *valid-arena-header-size*:
‹*valid-arena arena N vdom* ⟹ *C* ∈# *dom-m N* ⟹ *arena-header-size arena C* = *header-size* (*N* ∝ *C*)›
by (*auto simp*: *arena-header-size-def header-size-def arena-lifting*)

definition *rewatch-spec* :: ‹*nat twl-st-wl* ⇒ *nat twl-st-wl nres*› **where**
‹*rewatch-spec* = (λ(*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *WS*).
SPEC (λ(*M'*, *N'*, *D'*, *NE'*, *UE'*, *NEk'*, *UEk'*, *NS'*, *US'*, *N0'*, *U0'*, *Q'*, *WS'*).

$(M', N', D', NE', UE', NEk', UEk', NS', US', N0', U0', Q') = (M, N, D, NE, \{\#\}, NEk, UEk, NS, \{\#\}, N0, \{\#\}, Q) \wedge$
$correct\text{-}watching' (M, N', D, NE, UE, NEk', UEk', NS', US, N0, U0, Q', WS') \wedge$
$literals\text{-}are\text{-}\mathcal{L}_{in}' (M, N', D, NE, UE, NEk', UEk', NS', US, N0, U0, Q', WS'))\rangle$

lemma *blits-in-$\mathcal{L}_{in}$'-restart-wl-spec0'*:
$\langle literals\text{-}are\text{-}\mathcal{L}_{in}' (a, aq, ab, ac, ad, NEk, UEk, ae, af, N0, U0, Q, b) \Longrightarrow$
$literals\text{-}are\text{-}\mathcal{L}_{in}' (a, aq, ab, ac, ad, NEk, UEk, ae, af, N0, U0, \{\#\}, b)\rangle$
by (*auto simp*: *literals-are-$\mathcal{L}_{in}$'-empty blits-in-$\mathcal{L}_{in}$'-restart-wl-spec0*)

abbreviation *twl-st-heur-restart'''u* **where**
$\langle twl\text{-}st\text{-}heur\text{-}restart'''u\ r\ u \equiv$
$\{(S, T).\ (S, T) \in twl\text{-}st\text{-}heur\text{-}restart \wedge length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) = r \wedge$
$learned\text{-}clss\text{-}count\ S \leq u\}\rangle$

abbreviation *twl-st-heur-restart''''u* **where**
$\langle twl\text{-}st\text{-}heur\text{-}restart''''u\ r\ u \equiv$
$\{(S, T).\ (S, T) \in twl\text{-}st\text{-}heur\text{-}restart \wedge length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq r\ \wedge$
$learned\text{-}clss\text{-}count\ S \leq u\}\rangle$

fun *correct-watching'''* :: $\langle$- $\Rightarrow$ *'v twl-st-wl* $\Rightarrow$ *bool*$\rangle$ **where**
$\langle correct\text{-}watching'''\ \mathcal{A}\ (M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, Q, W) \longleftrightarrow$
$(\forall L \in\# all\text{-}lits\text{-}of\text{-}mm\ \mathcal{A}.$
$distinct\text{-}watched\ (W\ L) \wedge$
$(\forall (i, K, b) \in\# mset\ (W\ L).$
$i \in\# dom\text{-}m\ N \wedge K \in set\ (N \propto i) \wedge K \neq L \wedge$
$correctly\text{-}marked\text{-}as\text{-}binary\ N\ (i, K, b)) \wedge$
$fst\ `\#\ mset\ (W\ L) = clause\text{-}to\text{-}update\ L\ (M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, \{\#\}, \{\#\}))\rangle$

declare *correct-watching'''.simps*[*simp del*]

lemma *correct-watching'''-add-clause*:
assumes
corr: $\langle correct\text{-}watching'''\ \mathcal{A}\ ((a, aa, CD, ac, ad, NEk, UEk, NS, US, N0, U0, Q, b))\rangle$ **and**
leC: $\langle 2 \leq length\ C\rangle$ **and**
i-notin[*simp*]: $\langle i \notin\# dom\text{-}m\ aa\rangle$ **and**
dist[*iff*]: $\langle C\ !\ 0 \neq C\ !\ Suc\ 0\rangle$
shows $\langle correct\text{-}watching'''\ \mathcal{A}$
$((a, fmupd\ i\ (C, red)\ aa, CD, ac, ad, NEk, UEk, NS, US, N0, U0, Q, b$
$(C\ !\ 0 := b\ (C\ !\ 0)\ @\ [(i, C\ !\ Suc\ 0, length\ C = 2)],$
$C\ !\ Suc\ 0 := b\ (C\ !\ Suc\ 0)\ @\ [(i, C\ !\ 0, length\ C = 2)]))))\rangle$
proof −
have [*iff*]: $\langle C\ !\ Suc\ 0 \neq C\ !\ 0\rangle$
using $\langle C\ !\ 0 \neq C\ !\ Suc\ 0\rangle$ **by** *argo*
have [*iff*]: $\langle C\ !\ Suc\ 0 \in\# all\text{-}lits\text{-}of\text{-}m\ (mset\ C)\rangle$ $\langle C\ !\ 0 \in\# all\text{-}lits\text{-}of\text{-}m\ (mset\ C)\rangle$
$\langle C\ !\ Suc\ 0 \in set\ C\rangle$ $\langle C\ !\ 0 \in set\ C\rangle$ $\langle C\ !\ 0 \in set\ (watched\text{-}l\ C)\rangle$ $\langle C\ !\ Suc\ 0 \in set\ (watched\text{-}l\ C)\rangle$
using *leC* **by** (*force intro*!: *in-clause-in-all-lits-of-m nth-mem simp*: *in-set-conv-iff*
intro: *exI*[*of - 0*] *exI*[*of -* $\langle Suc\ 0\rangle$])+
have [*dest*!]: $\langle \bigwedge L.\ L \neq C\ !\ 0 \Longrightarrow L \neq C\ !\ Suc\ 0 \Longrightarrow L \in set\ (watched\text{-}l\ C) \Longrightarrow False\rangle$
by (*cases C*; *cases* $\langle tl\ C\rangle$; *auto*)+
have *i*: $\langle i \notin fst\ `\ set\ (b\ L)\rangle$ **if** $\langle L \in\# all\text{-}lits\text{-}of\text{-}mm\ \mathcal{A}\rangle$**for** *L*
using *corr i-notin that* **unfolding** *correct-watching'''.simps*
by *force*
have [*iff*]: $\langle (i, c, d) \notin set\ (b\ L)\rangle$ **if** $\langle L \in\# all\text{-}lits\text{-}of\text{-}mm\ \mathcal{A}\rangle$ **for** *L c d*

```
    using i[of L, OF that] by (auto simp: image-iff)
  then show ?thesis
    using corr
    by (force simp: correct-watching'''.simps ran-m-mapsto-upd-notin
      all-lits-of-mm-add-mset all-lits-of-mm-union clause-to-update-mapsto-upd-notin correctly-marked-as-binary.simps
        split: if-splits)
qed


lemma rewatch-correctness:
  assumes empty: ‹⋀L. L ∈# all-lits-of-mm 𝒜 ⟹ W L = []› and
    H[dest]: ‹⋀x. x ∈# dom-m N ⟹ distinct (N ∝ x) ∧ length (N ∝ x) ≥ 2› and
    incl: ‹set-mset (all-lits-of-mm (mset '# ran-mf N)) ⊆ set-mset (all-lits-of-mm 𝒜)›
  shows
    ‹rewatch N W ≤ SPEC(λW. correct-watching''' 𝒜 (M, N, C, NE, UE, NEk, UEk, NS, US, N₀,
U₀, Q, W))›
proof −
  define I where
    ‹I ≡ λ(a :: nat list) (b :: nat list) W.
       correct-watching''' 𝒜 ((M, fmrestrict-set (set a) N, C, NE, UE, NEk, UEk, NS, US, N₀, U₀, Q,
W))›
  have I0: ‹set-mset (dom-m N) ⊆ set x ∧ distinct x ⟹ I [] x W› for x
    using empty unfolding I-def by (auto simp: correct-watching'''.simps
      all-blits-are-in-problem-init.simps clause-to-update-def
      all-lits-of-mm-union)
  have le: ‹length (σ L) < size (dom-m N)›
     if ‹correct-watching''' 𝒜 (M, fmrestrict-set (set l1) N, C, NE, UE, NEk, UEk, NS, US, N₀, U₀,
Q, σ)› and
      ‹set-mset (dom-m N) ⊆ set x ∧ distinct x› and
     ‹x = l1 @ xa # l2› ‹xa ∈# dom-m N› ‹L ∈ set (N ∝ xa)›
     for L l1 σ xa l2 x
  proof −
    have 1: ‹card (set l1) ≤ length l1›
      by (auto simp: card-length)
    have ‹L ∈# all-lits-of-mm 𝒜›
      using that incl in-clause-in-all-lits-of-m[of L ‹mset (N ∝ xa)›]
      by (auto simp: correct-watching'''.simps dom-m-fmrestrict-set' ran-m-def
          all-lits-of-mm-add-mset all-lits-of-m-add-mset atm-of-all-lits-of-m
          in-all-lits-of-mm-ain-atms-of-iff
        dest!: multi-member-split)
    then have ‹distinct-watched (σ L)› and ‹fst ' set (σ L) ⊆ set l1 ∩ set-mset (dom-m N)›
      using that incl
      by (auto simp: correct-watching'''.simps dom-m-fmrestrict-set' dest!: multi-member-split)
    then have ‹length (map fst (σ L)) ≤ card (set l1 ∩ set-mset (dom-m N))›
      using 1 by (subst distinct-card[symmetric])
       (auto simp: distinct-watched-alt-def intro!: card-mono intro: order-trans)
    also have ‹... < card (set-mset (dom-m N))›
      using that by (auto intro!: psubset-card-mono)
    also have ‹... = size (dom-m N)›
      by (simp add: distinct-mset-dom distinct-mset-size-eq-card)
    finally show ?thesis by simp
  qed
  show ?thesis
    unfolding rewatch-def
    apply (refine-vcg
      nfoldli-rule[where I = ‹I›])
```

subgoal by (*rule I0*)
subgoal using *assms* **unfolding** *I-def* **by** *auto*
subgoal for *x xa l1 l2 σ* **using** *H*[*of xa*] **unfolding** *I-def* **apply** −
by (*rule*, *subst* (*asm*)*nth-eq-iff-index-eq*)
linarith+
subgoal for *x xa l1 l2 σ* **unfolding** *I-def* **by** (*rule le*) (*auto intro*!: *nth-mem*)
subgoal for *x xa l1 l2 σ* **unfolding** *I-def* **by** (*drule le*[**where** *L* = ‹*N* ∝ *xa* ! *1*›]) (*auto simp*: *I-def dest*!: *le*)
subgoal for *x xa l1 l2 σ*
unfolding *I-def*
by (*cases* ‹*the* (*fmlookup N xa*)›)
(*auto intro*!: *correct-watching'''-add-clause simp*: *dom-m-fmrestrict-set'*)
subgoal
unfolding *I-def*
by *auto*
subgoal by *auto*
subgoal unfolding *I-def*
by (*auto simp*: *fmlookup-restrict-set-id'*)
done
qed

lemma *heuristic-rel-incr-restartI*[*intro*!]:
‹*heuristic-rel A heur* ⟹ *heuristic-rel A* (*incr-restart-phase-end end-of-phase heur*)›
by (*auto simp*: *heuristic-rel-def heuristic-rel-stats-def incr-restart-phase-end-def*)

lemma *get-conflict-wl-is-None-heur-get-conflict-wl-is-None-ana*:
‹(*RETURN o get-conflict-wl-is-None-heur*, *RETURN o get-conflict-wl-is-None*) ∈
twl-st-heur-restart-ana' r (*u*) →*f* ⟨*Id*⟩*nres-rel*›
unfolding *get-conflict-wl-is-None-heur-def get-conflict-wl-is-None-def comp-def*
apply (*intro frefI nres-relI*) **apply** *refine-rcg*
by (*auto simp*: *twl-st-heur-restart-ana-def get-conflict-wl-is-None-heur-def get-conflict-wl-is-None-def option-lookup-clause-rel-def twl-st-heur-restart-def*
split: *option.splits*)
lemma *get-conflict-wl-is-None-heur-get-conflict-wl-is-None-restart*:
‹(*RETURN o get-conflict-wl-is-None-heur*, *RETURN o get-conflict-wl-is-None*) ∈
twl-st-heur-restart →*f* ⟨*Id*⟩*nres-rel*›
unfolding *get-conflict-wl-is-None-heur-def get-conflict-wl-is-None-def comp-def*
apply (*intro frefI nres-relI*) **apply** *refine-rcg*
by (*auto simp*: *twl-st-heur-restart-ana-def get-conflict-wl-is-None-heur-def get-conflict-wl-is-None-def option-lookup-clause-rel-def twl-st-heur-restart-def*
split: *option.splits*)

lemma *all-init-atms-alt-def*:
‹*all-init-atms* (*get-clauses-wl S'*)
(*IsaSAT-Setup.get-unkept-unit-init-clss-wl S'* + *IsaSAT-Setup.get-kept-unit-init-clss-wl S'* + *get-subsumed-init-clauses-wl S'* +
get-init-clauses0-wl S') = *all-init-atms-st S'*›
by (*auto simp*: *all-init-atms-st-def IsaSAT-Setup.get-unit-init-clss-wl-alt-def*)

lemma *twl-st-heur-restart-state-simp*:
assumes ‹(*S*, *S'*) ∈ *twl-st-heur-restart*›
shows
twl-st-heur-state-simp-watched: ‹*C* ∈# *L*$_{all}$ (*all-init-atms-st S'*) ⟹
watched-by-int S C = *watched-by S' C*›
‹*C* ∈# *L*$_{all}$ (*all-init-atms-st S'*) ⟹

get-watched-wl-heur S ! (nat-of-lit C) = get-watched-wl S′ C›**and**
‹*literals-to-update-wl S′ =*
uminus ‘# lit-of ‘# mset (drop (literals-to-update-wl-heur S) (rev (get-trail-wl S′)))› **and**
twl-st-heur-state-simp-watched2: ‹*C ∈# $\mathcal{L}_{all}$ (all-init-atms-st S′) ⟹*
nat-of-lit C < length(get-watched-wl-heur S)›
using *assms* **unfolding** *twl-st-heur-restart-def*
by (*solves* ‹*cases S*; *cases S′*; *auto simp add*: *Let-def map-fun-rel-def ac-simps all-init-atms-st-def*›)+

lemma *twl-st-heur-restart-ana-state-simp*:
assumes ‹(*S*, *S′*) ∈ *twl-st-heur-restart-ana u*›
shows
twl-st-heur-restart-ana-state-simp-watched: ‹*C ∈# $\mathcal{L}_{all}$ (all-init-atms-st S′) ⟹*
watched-by-int S C = watched-by S′ C›
‹*C ∈# $\mathcal{L}_{all}$ (all-init-atms-st S′) ⟹*
get-watched-wl-heur S ! (nat-of-lit C) = get-watched-wl S′ C›**and**
‹*literals-to-update-wl S′ =*
uminus ‘# lit-of ‘# mset (drop (literals-to-update-wl-heur S) (rev (get-trail-wl S′)))› **and**
twl-st-heur-restart-ana-state-simp-watched2: ‹*C ∈# $\mathcal{L}_{all}$ (all-init-atms-st S′) ⟹*
nat-of-lit C < length(get-watched-wl-heur S)›
using *assms twl-st-heur-restart-state-simp*[*of S S′*] **unfolding** *twl-st-heur-restart-ana-def*
by (*auto simp*:)

lemma *twl-st-heur-restart-ana-watchlist-in-vdom*:
‹*get-watched-wl-heur x2e ! nat-of-lit L ! x1d = (a, b) ⟹*
(x2e, x2f) ∈ twl-st-heur-restart-ana r ⟹ L ∈# $\mathcal{L}_{all}$ (all-init-atms-st x2f) ⟹
x1d < length (get-watched-wl-heur x2e ! nat-of-lit L) ⟹
a ∈ set (get-vdom x2e)›
apply (*drule nth-mem*)
by (*subst* (*asm*) *twl-st-heur-restart-ana-state-simp*, *assumption*, *assumption*)+
(*auto 5 3 simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def vdom-m-def*
all-init-atms-alt-def
dest!: *multi-member-split*)

lemma *twl-st-heur-restart-alt-def2*:
‹*twl-st-heur-restart =*
{(*S*,*T*).
let M′ = get-trail-wl-heur S; *N′ = get-clauses-wl-heur S*; *D′ = get-conflict-wl-heur S*;
W′ = get-watched-wl-heur S; *j = literals-to-update-wl-heur S*; *outl = get-outlearned-heur S*;
cach = get-conflict-cach S; *clvls = get-count-max-lvls-heur S*;
vm = get-vmtf-heur S;
vdom = get-aivdom S; *heur = get-heur S*; *old-arena = get-old-arena S*;
lcount = get-learned-count S; *occs = get-occs S in*
let M = get-trail-wl T; *N = get-clauses-wl T*; *D = get-conflict-wl T*;
Q = literals-to-update-wl T;
W = get-watched-wl T; *N0 = get-init-clauses0-wl T*; *U0 = get-learned-clauses0-wl T*;
NS = get-subsumed-init-clauses-wl T; *US = get-subsumed-learned-clauses-wl T*;
NEk = get-kept-unit-init-clss-wl T; *UEk = get-kept-unit-learned-clss-wl T*;
NE = get-unkept-unit-init-clss-wl T; *UE = get-unkept-unit-learned-clss-wl T in*
(M′, M) ∈ trail-pol (all-init-atms-st T) ∧
valid-arena N′ N (set (get-vdom-aivdom vdom)) ∧
(D′, D) ∈ option-lookup-clause-rel (all-init-atms-st T) ∧
(D = None ⟶ j ≤ length M) ∧
Q = uminus ‘# lit-of ‘# mset (drop j (rev M)) ∧
(W′, W) ∈ ⟨Id⟩map-fun-rel (D₀ (all-init-atms-st T)) ∧
vm ∈ bump-heur (all-init-atms-st T) M ∧
no-dup M ∧

```
    clvls ∈ counts-maximum-level M D ∧
    cach-refinement-empty (all-init-atms-st T) cach ∧
    out-learned M D outl ∧
    clss-size-corr-restart N NE {#} NEk UEk NS {#} N0 {#} lcount ∧
    vdom-m (all-init-atms-st T) W N ⊆ set (get-vdom-aivdom vdom) ∧
    aivdom-inv-dec vdom (dom-m N) ∧
    isasat-input-bounded (all-init-atms-st T) ∧
    isasat-input-nempty (all-init-atms-st T) ∧
    old-arena = [] ∧
      heuristic-rel (all-init-atms-st T) heur ∧
    (occs, empty-occs-list (all-init-atms-st T)) ∈ occurrence-list-ref
  }›
  unfolding twl-st-heur-restart-def Let-def
  by (auto simp: all-init-atms-st-def)

lemma length-watched-le-ana:
  assumes
    prop-inv: ‹correct-watching'-leaking-bin x1› and
    xb-x'a: ‹(x1a, x1) ∈ twl-st-heur-restart-ana r› and
    x2': ‹x2 ∈# \mathcal{L}_{all} (all-init-atms-st x1)›
  shows ‹length (watched-by x1 x2) ≤ r − MIN-HEADER-SIZE›
proof −
  have x2: ‹x2 ∈# all-init-lits-of-wl x1›
    using \mathcal{L}_{all}-all-init-atms(2) x2' by blast
  have dist: ‹distinct-watched (watched-by x1 x2)›
    using prop-inv x2 unfolding all-atms-def all-lits-def
    by (cases x1; auto simp: correct-watching'-leaking-bin.simps ac-simps all-lits-st-alt-def[symmetric])
  then have dist: ‹distinct-watched (watched-by x1 x2)›
    using xb-x'a
    by (cases x1; auto simp: \mathcal{L}_{all}-atm-of-all-lits-of-mm correct-watching.simps)
  have dist-vdom: ‹distinct (get-vdom x1a)›
    using xb-x'a
    by (cases x1)
     (auto simp: twl-st-heur-restart-ana-def twl-st-heur-restart-def twl-st-heur'-def aivdom-inv-dec-alt-def
Let-def)

  have
      valid: ‹valid-arena (get-clauses-wl-heur x1a) (get-clauses-wl x1) (set (get-vdom x1a))›
    using xb-x'a unfolding all-atms-def all-lits-def
    by (cases x1)
     (auto simp: twl-st-heur-restart-ana-def twl-st-heur-restart-def Let-def)

  have ‹vdom-m (all-init-atms-st x1) (get-watched-wl x1) (get-clauses-wl x1) ⊆ set (get-vdom x1a)›
    using xb-x'a
    by (cases x1)
      (auto simp: twl-st-heur-restart-ana-def twl-st-heur-restart-alt-def2 ac-simps Let-def)

  then have subset: ‹set (map fst (watched-by x1 x2)) ⊆ set (get-vdom x1a)›
    using x2' unfolding vdom-m-def \mathcal{L}_{all}-all-atms \mathcal{L}_{all}-all-init-atms
    by (cases x1)
      (force simp: twl-st-heur'-def twl-st-heur-def
        dest!: multi-member-split)
  have watched-incl: ‹mset (map fst (watched-by x1 x2)) ⊆# mset (get-vdom x1a)›
    by (rule distinct-subseteq-iff[THEN iffD1])
      (use dist[unfolded distinct-watched-alt-def] dist-vdom subset in
        ‹simp-all flip: distinct-mset-mset-distinct›)
```

have *vdom-incl*: ‹*set* (*get-vdom x1a*) ⊆ {*MIN-HEADER-SIZE*..< *length* (*get-clauses-wl-heur x1a*)}›
using *valid-arena-in-vdom-le-arena*[*OF valid*] *arena-dom-status-iff*[*OF valid*] **by** *auto*

have ‹*length* (*get-vdom x1a*) ≤ *length* (*get-clauses-wl-heur x1a*) − *MIN-HEADER-SIZE*›
by (*subst distinct-card*[*OF dist-vdom*, *symmetric*])
(*use card-mono*[*OF* - *vdom-incl*] **in** *auto*)
then show *?thesis*
using *size-mset-mono*[*OF watched-incl*] *xb-x'a*
by (*auto intro*!: *order-trans*[*of* ‹*length* (*watched-by x1 x2*)› ‹*length* (*get-vdom x1a*)›]
simp: *twl-st-heur-restart-ana-def*)
qed

lemma D_0-*cong'*: ‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *x* ∈ D_0 $\mathcal{A}$ ⟹ *x* ∈ D_0 $\mathcal{B}$›
by (*subst* (*asm*) D_0-*cong*, *assumption*)
lemma *map-fun-rel-*D_0*-cong*: ‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹*x* ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$) ⟹ *x* ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{B}$)›
by (*subst* (*asm*) D_0-*cong*, *assumption*)

lemma *vdom-m-cong'*: *set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *x* ∈ *vdom-m* $\mathcal{A}$ *a b* ⟹ *x* ∈ *vdom-m* $\mathcal{B}$ *a b*
by (*subst* (*asm*) *vdom-m-cong*, *assumption*)
lemma *vdom-m-cong''*: *set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *vdom-m* $\mathcal{A}$ *a b* ⊆ *A* ⟹ *vdom-m* $\mathcal{B}$ *a b* ⊆ *A*
by (*subst* (*asm*) *vdom-m-cong*, *assumption*)
lemma *cach-refinement-empty-cong'*: *set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *cach-refinement-empty* $\mathcal{A}$ *x* ⟹ *cach-refinement-empty* $\mathcal{B}$ *x*
by (*subst* (*asm*) *cach-refinement-empty-cong*, *assumption*)

end
theory *IsaSAT-Restart-Reduce*
imports *IsaSAT-Restart IsaSAT-Restart-Reduce-Defs*
begin

definition *find-local-restart-target-level* **where**
‹*find-local-restart-target-level M* - = *SPEC*(λ*i*. *i* ≤ *count-decided M*)›

lemma *find-local-restart-target-level-alt-def*:
‹*find-local-restart-target-level M vm* = *do* {
(*b*, *i*) ← *SPEC*(λ(*b*::*bool*, *i*). *i* ≤ *count-decided M*);
RETURN i
}›
unfolding *find-local-restart-target-level-def* **by** (*auto simp*: *RES-RETURN-RES2 uncurry-def*)

lemma *find-local-restart-target-level-int-find-local-restart-target-level*:
‹(*uncurry find-local-restart-target-level-int*, *uncurry find-local-restart-target-level*) ∈
[λ(*M*, *vm*). *vm* ∈ *bump-heur* $\mathcal{A}$ *M*]$_f$ *trail-pol* $\mathcal{A}$ ×$_r$ *Id* → ⟨*nat-rel*⟩*nres-rel*›
unfolding *find-local-restart-target-level-int-def find-local-restart-target-level-alt-def*
uncurry-def Let-def
apply (*intro frefI nres-relI*)
apply *clarify*
subgoal for *a aa ab ac ad b ba ae bb*
unfolding *access-focused-vmtf-array-def nres-monad3 bind-to-let-conv Let-def*
apply (*refine-rcg WHILEIT-rule*[**where** *R* = ‹*measure* (λ(*brk*, *i*). (*If brk 0 1*) + *length b* − *i*)›]
assert.ASSERT-leI)
subgoal by *auto*
subgoal
unfolding *find-local-restart-target-level-int-inv-def*

```
    by (auto simp: trail-pol-alt-def control-stack-length-count-dec)
  subgoal by auto
  subgoal by (auto simp: trail-pol-alt-def intro: control-stack-le-length-M)
  subgoal for s x1 x2
    by (subgoal-tac ‹a ! (b ! x2) ∈# \mathcal{L}_{all} \mathcal{A}›)
      (auto simp: trail-pol-alt-def rev-map lits-of-def rev-nth
          vmtf-def atms-of-def bump-heur-def bump-get-heuristics-def
        intro!: literals-are-in-\mathcal{L}_{in}-trail-in-lits-of-l)
  subgoal by (auto simp: find-local-restart-target-level-int-inv-def)
  subgoal by (auto simp: trail-pol-alt-def control-stack-length-count-dec
        find-local-restart-target-level-int-inv-def)
  subgoal by auto
  done
 done

lemma find-local-restart-target-level-st-alt-def:
 ‹find-local-restart-target-level-st = (λS. do {
    find-local-restart-target-level-int (get-trail-wl-heur S) (get-vmtf-heur S)})›
 apply (intro ext)
 by (auto simp: find-local-restart-target-level-st-def)

lemma cdcl-twl-local-restart-wl-D-spec-int:
 ‹cdcl-twl-local-restart-wl-spec (M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, Q, W) ≥ ( do {
    ASSERT(∃ last-GC last-Restart. restart-abs-wl-pre (M, N, D, NE, UE, NEk, UEk, NS, US, N0,
U0, Q, W) last-GC last-Restart False);
    i ← SPEC(λ-. True);
    if i
    then RETURN (M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, Q, W)
    else do {
     (M, Q') ← SPEC(λ(M', Q'). (∃ K M2. (Decided K # M', M2) ∈ set (get-all-ann-decomposition
M) ∧
          Q' = {#}) ∨ (M' = M ∧ Q' = Q));
     RETURN (M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, Q', W)
    }
  })›
proof −
 have If-Res: ‹(if i then (RETURN f) else (RES g)) = (RES (if i then {f} else g))› for i f g
  by auto
 show ?thesis
  unfolding cdcl-twl-local-restart-wl-spec-def prod.case RES-RETURN-RES2 If-Res
  by refine-vcg
    (auto simp: If-Res RES-RETURN-RES2 RES-RES-RETURN-RES uncurry-def
     image-iff split:if-splits)
qed

lemma cdcl-twl-local-restart-wl-D-heur-cdcl-twl-local-restart-wl-D-spec:
 ‹(cdcl-twl-local-restart-wl-D-heur, cdcl-twl-local-restart-wl-spec) ∈
  twl-st-heur''''u r u →_f ⟨twl-st-heur''''u r u⟩nres-rel›
proof −
 have K: ‹(do {
  j ← mop-isa-length-trail (get-trail-wl-heur S);
  RES {f j}
  }) = (do {
  ASSERT (isa-length-trail-pre (get-trail-wl-heur S));
  RES {f (isa-length-trail (get-trail-wl-heur S))}})› for S :: isasat and f
  by (cases S) (auto simp: mop-isa-length-trail-def)
```

have *K2*: ‹(*case S of*
(*a*, *b*) ⇒ *RES* (Φ *a b*)) =
(*RES* (*case S of* (*a*, *b*) ⇒ Φ *a b*))› **for** *S*
by (*cases S*) *auto*

have [*dest*]: ‹*av* = *None*› ‹*out-learned a av am* ⟹ *out-learned x1 av am*›
if ‹*restart-abs-wl-pre* (*a*, *au*, *av*, *aw*, *ax*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *ay*, *bd*) *last-GC last-Restart False*›
for *a au av aw ax ay bd x1 am NEk UEk NS US last-GC last-Restart N0 U0*
using *that*
unfolding *restart-abs-wl-pre-def restart-abs-l-pre-def*
restart-prog-pre-def
by (*auto simp*: *twl-st-l-def state-wl-l-def out-learned-def*)
have [*refine0*]:
‹*find-local-restart-target-level-int* (*get-trail-wl-heur S*) (*get-vmtf-heur S*) ≤
⇓ {(*i*, *b*). *b* = (*i* = *count-decided* (*get-trail-wl T*)) ∧
i ≤ *count-decided* (*get-trail-wl T*)} (*SPEC* (λ-. *True*))›
if ‹(*S*, *T*) ∈ *twl-st-heur*› **for** *S T*
apply (*rule find-local-restart-target-level-int-find-local-restart-target-level*[*THEN*
fref-to-Down-curry, *THEN order-trans*, *of* ‹*all-atms-st T*› ‹*get-trail-wl T*› ‹*get-vmtf-heur S*›])
subgoal using *that* **unfolding** *twl-st-heur-def* **by** *auto*
subgoal using *that* **unfolding** *twl-st-heur-def* **by** *auto*
subgoal by (*auto simp*: *find-local-restart-target-level-def conc-fun-RES*)
done
have *H*:
‹*set-mset* (*all-atms-st S*) =
set-mset (*all-init-atms-st S*)› (**is** *?A*)
‹*set-mset* (*all-atms-st S*) =
set-mset (*all-atms* (*get-clauses-wl S*) (*get-unit-clauses-wl S* + *get-subsumed-init-clauses-wl S* + *get-init-clauses0-wl S*))›
(**is** *?B*)
‹*get-conflict-wl S* = *None*› (**is** *?C*)
if *pre*: ‹*restart-abs-wl-pre S last-GC last-Restart False*›
for *S last-GC last-Restart*
proof −
obtain *T U* **where**
ST: ‹(*S*, *T*) ∈ *state-wl-l None*› **and**
‹*correct-watching S*› **and**
‹*blits-in-*$\mathcal{L}_{in}$ *S*› **and**
TU: ‹(*T*, *U*) ∈ *twl-st-l None*› **and**
struct: ‹*twl-struct-invs U*› **and**
‹*twl-list-invs T*› **and**
‹*clauses-to-update-l T* = {#}› **and**
‹*twl-stgy-invs U*› **and**
confl: ‹*get-conflict U* = *None*›
using *pre* **unfolding** *restart-abs-wl-pre-def restart-abs-l-pre-def restart-prog-pre-def* **apply** −
by *blast*

show *?C*
using *ST TU confl* **by** *auto*

have *alien*: ‹*cdcl*$_W$*-restart-mset.no-strange-atm* (*state*$_W$*-of U*)›
using *struct* **unfolding** *twl-struct-invs-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*
pcdcl-all-struct-invs-def state$_W$*-of-def*
by *fast*+
then show *?A* **and** *?B*

subgoal *A*
using *ST TU* **unfolding** *set-eq-iff in-set-all-atms-iff*
in-set-all-atms-iff in-set-all-init-atms-iff get-unit-clauses-wl-alt-def
using *literals-are-$\mathcal{L}_{in}$'-literals-are-$\mathcal{L}_{in}$-iff*(*3*) *struct* **by** *blast*
subgoal
using *ST TU alien* **unfolding** *set-eq-iff in-set-all-atms-iff all-atms-st-def*
in-set-all-atms-iff in-set-all-init-atms-iff get-unit-clauses-wl-alt-def
apply (*subst all-clss-lf-ran-m*[*symmetric*])
apply (*subst all-clss-lf-ran-m*[*symmetric*])
unfolding *image-mset-union*
by (*auto simp*: *cdcl$_W$-restart-mset.no-strange-atm-def twl-st twl-st-l in-set-all-atms-iff*
in-set-all-init-atms-iff)
done
qed
have *P*: ‹*P*›
if
ST: ‹(*S*, *bt*, *bu*, *bv*, *bw*, *bx*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *by*, *bz*)
∈ *twl-st-heur*› **and**
‹∃ *last-GC last-Restart*. *restart-abs-wl-pre* (*bt*, *bu*, *bv*, *bw*, *bx*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *by*, *bz*)
last-GC last-Restart False› **and**
‹*restart-abs-wl-heur-pre*
S
False› **and**
lvl: ‹(*lvl*, *i*)
∈ {(*i*, *b*).
b = (*i* = *count-decided* (*get-trail-wl* (*bt*, *bu*, *bv*, *bw*, *bx*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *by*, *bz*))) ∧
i ≤ *count-decided* (*get-trail-wl* (*bt*, *bu*, *bv*, *bw*, *bx*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *by*, *bz*))}› **and**
‹*i* ∈ {-. *True*}› **and**
‹*lvl* ≠ *count-decided-st-heur S*› **and**
i: ‹¬ *i*› **and**
H: ‹
isa-find-decomp-wl-imp (*get-trail-wl-heur S*) *lvl* (*get-vmtf-heur S*)
≤ ⇓ {(*a*, *b*). (*a*,*b*) ∈ *trail-pol* (*all-atms-st* (*bt*, *bu*, *bv*, *bw*, *bx*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *by*, *bz*)) ×$_f$
Id}
(*find-decomp-w-ns* (*all-atms-st* (*bt*, *bu*, *bv*, *bw*, *bx*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *by*, *bz*)) *bt lvl vm0*)
⟹ *P*›
for *a aa ab ac ad b ae af ag ba ah ai aj ak al am bb an bc ao aq bd ar as at'*
au av aw be ax ay az bf bg bh bi bj bk bl bm bn bo bp bq bt bu bv aqbd
bw bx by bz lvl i x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f S
x1g x2g x1h x2h x1i x2i P NS US last-GC last-Restart N0 U0 NEk UEk heur heur2 stats M' stats42
proof −
let *?$\mathcal{A}$* = ‹*all-atms-st* (*bt*, *bu*, *bv*, *bw*, *bx*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *by*, *bz*)›
let *?bo* = ‹*get-vdom-aivdom* (*get-aivdom S*)›
let *?ae* = ‹*get-clauses-wl-heur S*›
let *?heur* = ‹*get-heur S*›
let *?vm* = ‹*get-vmtf-heur S*›
have
tr: ‹(*get-trail-wl-heur S*, *bt*) ∈ *trail-pol ?$\mathcal{A}$*› **and**
‹*valid-arena ?ae bu* (*set ?bo*)› **and**
vm: ‹*?vm* ∈ *bump-heur ?$\mathcal{A}$ bt*› **and**
bounded: ‹*isasat-input-bounded ?$\mathcal{A}$*› **and**
heur: ‹*heuristic-rel ?$\mathcal{A}$?heur*›
using *ST* **unfolding** *twl-st-heur-def all-atms-def*
by (*auto*)

have *n-d*: ‹*no-dup bt*›

using *tr* **by** (*auto simp*: *trail-pol-def*)
show *?thesis*
apply (*rule H*)
apply (*rule isa-find-decomp-wl-imp-find-decomp-wl-imp*[*THEN fref-to-Down-curry2*, *THEN order-trans*,
of bt lvl ‹*get-vmtf-heur S*› - - - ‹*?A*›])
subgoal using *lvl i* **by** *auto*
subgoal using *vm tr* **by** *auto*
apply (*subst* (*3*) *Down-id-eq*[*symmetric*])
apply (*rule order-trans*)
apply (*rule ref-two-step'*)
apply (*rule find-decomp-wl-imp-find-decomp-wl'*[*THEN fref-to-Down-curry2*, *of* - *bt lvl* ‹*get-vmtf-heur S*›])
subgoal
using *that*(*1*−*8*) *vm bounded n-d tr*
by (*auto simp*: *find-decomp-w-ns-pre-def dest*: *trail-pol-literals-are-in-*$\mathcal{L}_{in}$*-trail*)
subgoal by *auto*
using *ST*
by (*auto simp*: *find-decomp-w-ns-def conc-fun-RES twl-st-heur-def*)
qed
note *cong* = *trail-pol-cong heuristic-rel-cong*
option-lookup-clause-rel-cong D_0*-cong isa-vmtf-cong*
cach-refinement-empty-cong vdom-m-cong isasat-input-nempty-cong
isasat-input-bounded-cong heuristic-rel-cong

show *?thesis*
supply [[*goals-limit=1*]]
unfolding *cdcl-twl-local-restart-wl-D-heur-def*
unfolding
find-decomp-wl-st-int-def find-local-restart-target-level-def incr-restart-stat-def
empty-Q-def find-local-restart-target-level-st-def nres-monad-laws
apply (*intro frefI nres-relI*)
apply *clarify*
apply (*rule ref-two-step*)
prefer *2*
apply (*rule cdcl-twl-local-restart-wl-D-spec-int*)
unfolding *bind-to-let-conv RES-RETURN-RES2 nres-monad-laws Let-def*
apply (*refine-vcg*)
subgoal unfolding *restart-abs-wl-heur-pre-def* **by** *blast*
apply *assumption*
subgoal by (*simp add*: *twl-st-heur-count-decided-st-alt-def*)
subgoal by (*auto simp*: *twl-st-heur-def count-decided-st-heur-def trail-pol-def*)

apply (*rule P*)
apply *assumption*+
apply (*rule refine-generalise1*)
apply *assumption*
subgoal for *a aa ab ac ad b ae af ag ba ah ai aj ak al az*
unfolding *RETURN-def RES-RES2-RETURN-RES RES-RES13-RETURN-RES find-decomp-w-ns-def conc-fun-RES*
RES-RES13-RETURN-RES K2 K
apply (*auto simp*: *intro-spec-iff intro*!: *ASSERT-leI isa-length-trail-pre*)
apply (*auto simp*: *isa-length-trail-length-u*[*THEN fref-to-Down-unRET-Id*]
intro: *trail-pol-no-dup*)
apply (*frule twl-st-heur-change-subsumed-clauses*[**where** *US'* = *ba* **and** *NS'* = *ag* **and**
lcount' = ‹*get-learned-count a*›])

```
    apply (solves ‹auto dest: H(2)›)[]
    apply (solves ‹auto simp: twl-st-heur-def›)[]
    apply (frule H(2))
    apply (frule H(3))
  apply (clarsimp simp: twl-st-heur-def)
  apply (rule-tac x=afa in exI)
  apply (auto simp: isa-length-trail-length-u[THEN fref-to-Down-unRET-Id] learned-clss-count-def
        all-atms-st-def
    intro: trail-pol-no-dup)
    done
  done
qed



lemma distinct-mset-union-iff:
  ‹distinct-mset (xs + ys) = (distinct-mset xs ∧ distinct-mset ys ∧ set-mset xs ∩ set-mset ys = {})›
  by (induction xs) (auto)

lemma aivdom-inv-dec-remove-deleted-clauses-from-avdom:
  ‹aivdom-inv-dec avdom0 (dom-m N) ⟹
  mset (take a (get-avdom-aivdom ba)) ⊆# mset (get-avdom-aivdom avdom0) ⟹
   mset (take a (get-avdom-aivdom ba)) ∩# dom-m N = mset (get-avdom-aivdom avdom0) ∩# dom-m
N ⟹
    get-vdom-aivdom ba = get-vdom-aivdom avdom0 ⟹
    get-ivdom-aivdom ba = get-ivdom-aivdom avdom0 ⟹
  mset (get-tvdom-aivdom ba) ⊆# mset (get-avdom-aivdom avdom0) ⟹
  mset (take a (get-avdom-aivdom ba)) ∩# dom-m N = mset (get-avdom-aivdom avdom0) ∩# dom-m
N ⟹
  aivdom-inv-dec (take-avdom-aivdom a ba) (dom-m N)›
 supply [simp del] = distinct-finite-set-mset-subseteq-iff
 using distinct-mset-mono[of ‹mset (take a (get-avdom-aivdom ba))› ‹mset (get-avdom-aivdom av-
dom0)›]
 apply (auto simp: aivdom-inv-dec-alt-def2 distinct-mset-mono intro: distinct-take
   simp flip: distinct-subseteq-iff)
 apply auto
 apply (metis UnE comp-apply in-mono inter-iff set-mset-comp-mset)
 apply (subst distinct-subseteq-iff[symmetric])
 apply (auto dest: distinct-mset-mono)
 by (metis mset-subset-eqD set-mset-mset subsetD)

lemma remove-deleted-clauses-from-avdom:
  assumes ‹aivdom-inv-dec avdom0 (dom-m N)›
   shows ‹remove-deleted-clauses-from-avdom N avdom0 ≤ SPEC(λaivdom. aivdom-inv-dec aivdom
(dom-m N) ∧
     get-vdom-aivdom aivdom = get-vdom-aivdom avdom0∧
     get-ivdom-aivdom aivdom = get-ivdom-aivdom avdom0∧
    mset (get-tvdom-aivdom aivdom) ⊆# mset (get-avdom-aivdom avdom0) ∧
   (∀ C ∈ set (get-tvdom-aivdom aivdom). C ∈# dom-m N ∧ ¬irred N C ∧ length (N ∝ C) ≠ 2))›
proof −
  have dist-avdom: ‹distinct (get-avdom-aivdom avdom0)›
    using assms by (auto simp: aivdom-inv-dec-alt-def2)
  have I0: ‹remove-deleted-clauses-from-avdom-inv N avdom0 (0, 0, empty-tvdom avdom0)›
    unfolding remove-deleted-clauses-from-avdom-inv-def by auto
  have ISuc-keep:
    ‹x ⟹ remove-deleted-clauses-from-avdom-inv N avdom0
```

(*a+1*, *aa* + *1*, *push-to-tvdom* (*get-avdom-aivdom ba* ! *aa*) (*swap-avdom-aivdom ba a aa*))› (**is** ‹- ⟹ *?A*›) **and**
ISuc-keep-no:
‹*remove-deleted-clauses-from-avdom-inv N avdom0*
(*a+1*, *aa* + *1*, (*swap-avdom-aivdom ba a aa*))› (**is** *?B*)
if
‹*remove-deleted-clauses-from-avdom-inv N avdom0 s*› **and**
‹*case s of* (*i*, *j*, *avdom*) ⇒ *j* < *length* (*get-avdom-aivdom avdom0*)› **and**
‹*s* = (*a*, *b*)› **and**
‹*b* = (*aa*, *ba*)› **and**
‹*get-avdom-aivdom ba* ! *aa* ∈# *dom-m N*› **and**
irred: ‹*x* ⟶ ¬ *irred N* (*get-avdom-aivdom ba* ! *aa*) ∧ *length* (*N* ∝ (*get-avdom-aivdom ba* ! *aa*)) ≠ *2*›
for *ba aa s a b x*
proof −
have [*simp*]: ‹*aa* < *length* (*get-avdom-aivdom ba*) ⟹
add-mset (*get-avdom-aivdom ba* ! *aa*) (*mset* (*take aa* (*get-avdom-aivdom ba*)) + *mset* (*drop* (*Suc aa*) (*get-avdom-aivdom ba*))) =
mset (*get-avdom-aivdom ba*)›
using *that* **apply** −
apply (*subst* (*4*)*append-take-drop-id*[*symmetric*, *of* ‹*get-avdom-aivdom ba*› *aa*])
apply (*subst mset-append*)
by (*auto simp flip*: *Cons-nth-drop-Suc*)
have *1*: ‹*distinct* (*take a* (*get-avdom-aivdom ba*) @ *drop aa* (*get-avdom-aivdom ba*))›
using *assms that*
distinct-mset-mono[*of* ‹*mset* (*take a* (*get-avdom-aivdom ba*) @ *drop aa* (*get-avdom-aivdom ba*))›
‹*mset* (*get-avdom-aivdom avdom0*)›]
by (*auto simp*: *aivdom-inv-dec-alt-def2 remove-deleted-clauses-from-avdom-inv-def distinct-mset-union-iff*)
have *2*: ‹*a* = *aa* ⟹*distinct* (*get-avdom-aivdom ba*)›
by (*metis* ‹*distinct* (*take a* (*get-avdom-aivdom ba*) @ *drop aa* (*get-avdom-aivdom ba*))› *append-take-drop-id*)

show ‹*x* ⟹ *?A*›
using *assms that dist-avdom 1 2* **apply** −
apply (*cases* ‹*Suc a* ≤ *aa*›)
unfolding *remove-deleted-clauses-from-avdom-inv-def prod.simps*
apply (*auto simp*: *drop-swap-irrelevant swap-only-first-relevant Suc-le-eq take-update-last remove-deleted-clauses-from-avdom-inv-def*
intro: *subset-mset.dual-order.trans*
simp flip: *take-Suc-conv-app-nth Cons-nth-drop-Suc*)
apply (*auto simp*: *take-Suc-conv-app-nth*)
done
show *?B*
using *1 2 assms that dist-avdom*
apply (*cases* ‹*Suc a* ≤ *aa*›)
unfolding *remove-deleted-clauses-from-avdom-inv-def prod.simps*
apply (*auto simp*: *drop-swap-irrelevant swap-only-first-relevant Suc-le-eq take-update-last remove-deleted-clauses-from-avdom-inv-def*
intro: *subset-mset.dual-order.trans*
simp flip: *take-Suc-conv-app-nth Cons-nth-drop-Suc*)
apply (*auto simp*: *take-Suc-conv-app-nth*)
done
qed
have *ISuc-nokeep*:
‹*remove-deleted-clauses-from-avdom-inv N avdom0*
(*a*, *aa* + *1*, *ba*)› (**is** *?A*)

```
    if
      ‹remove-deleted-clauses-from-avdom-inv N avdom0 s› and
      ‹case s of (i, j, avdom) ⇒ j < length (get-avdom-aivdom avdom0)› and
      ‹s = (a, b)› and
      ‹b = (aa, ba)› and
      ‹get-avdom-aivdom ba ! aa ∉# dom-m N›
    for ba aa s a b
  proof −
    have ‹aa < length (get-avdom-aivdom ba) ⟹
      add-mset (get-avdom-aivdom ba ! aa) (mset (take aa (get-avdom-aivdom ba)) + mset (drop (Suc
aa) (get-avdom-aivdom ba))) =
      mset (get-avdom-aivdom ba)›
      using that apply −
      apply (subst (4)append-take-drop-id[symmetric, of ‹get-avdom-aivdom ba› aa])
      apply (subst mset-append)
      by (auto simp flip: Cons-nth-drop-Suc)
    have ‹distinct (take a (get-avdom-aivdom ba) @ drop aa (get-avdom-aivdom ba))›
      using assms that
        distinct-mset-mono[of ‹mset (take a (get-avdom-aivdom ba) @ drop aa (get-avdom-aivdom ba))›
        ‹mset (get-avdom-aivdom avdom0)›]
    by (auto simp: aivdom-inv-dec-alt-def2 remove-deleted-clauses-from-avdom-inv-def distinct-mset-union-iff)
    moreover have ‹a = aa  ⟹distinct (get-avdom-aivdom ba)›
        by (metis ‹distinct (take a (get-avdom-aivdom ba) @ drop aa (get-avdom-aivdom ba))› ap-
pend-take-drop-id)

    ultimately show ?A
      using assms that dist-avdom apply −
      apply (cases ‹Suc a ≤ aa›)
      unfolding remove-deleted-clauses-from-avdom-inv-def prod.simps
    by (auto simp: drop-swap-irrelevant swap-only-first-relevant Suc-le-eq take-update-last remove-deleted-clauses-from-av
        intro: subset-mset.dual-order.trans subset-mset-trans-add-mset
        simp flip:  take-Suc-conv-app-nth Cons-nth-drop-Suc)
  qed

  show ?thesis
    using assms
    unfolding remove-deleted-clauses-from-avdom-def Let-def is-candidate-for-removal-def
    apply (refine-vcg WHILEIT-rule[where R = ‹measure (λ(i, j, avdom). length (get-avdom-aivdom
avdom) − j)›])
    subgoal by auto
    subgoal by (rule I0)
    subgoal by (auto simp: remove-deleted-clauses-from-avdom-inv-def)
    subgoal supply [[unify-trace-failure]] by (rule ISuc-keep)
    subgoal by auto
    subgoal by (rule ISuc-keep-no)
    subgoal by auto
    subgoal by (rule ISuc-nokeep)
    subgoal by (auto simp: aivdom-inv-dec-alt-def)
    subgoal by (auto simp: remove-deleted-clauses-from-avdom-inv-def)
    subgoal by (force intro!: aivdom-inv-dec-remove-deleted-clauses-from-avdom
      simp: remove-deleted-clauses-from-avdom-inv-def simp flip: distinct-subseteq-iff)
    subgoal by (auto simp: remove-deleted-clauses-from-avdom-inv-def)
    subgoal by (auto simp: remove-deleted-clauses-from-avdom-inv-def)
    subgoal by (force simp: remove-deleted-clauses-from-avdom-inv-def simp flip: distinct-subseteq-iff)
    subgoal by (auto simp: remove-deleted-clauses-from-avdom-inv-def)
    done
```

qed

lemma *arena-status-mark-unused*[*simp*]:
‹*arena-status* (*mark-unused arena C*) *D* = *arena-status arena D*›
by (*auto simp*: *arena-status-def mark-unused-def LBD-SHIFT-def*
nth-list-update')

lemma *isa-is-candidate-for-removal-is-candidate-for-removal*:
assumes
valid: ‹*valid-arena arena N vdom*› **and**
C: ‹(*C*, *C'*) ∈ *nat-rel*› **and**
MM': ‹(*M*, *M'*) ∈ *trail-pol* $\mathcal{A}$› **and**
$\mathcal{A}$: ‹*set-mset* (*all-atms N NUE*) = *set-mset* $\mathcal{A}$›
shows ‹*isa-is-candidate-for-removal M C arena* ≤ ⇓ *bool-rel* (*is-candidate-for-removal C' N*)›
proof −
have [*simp*]: ‹*C* ∈# *dom-m N* ⟹ *N* ∝ *C* ≠ []› **and**
C'[*simp*]: ‹*C'* = *C*›
using *valid C* **by** (*auto simp*: *valid-arena-nempty*)
have *1*: ‹*C* ∈# *dom-m N* ⟹ *atm-of* (*arena-lit arena C*) ∈# $\mathcal{A}$›
using *valid* $\mathcal{A}$
by (*cases* ‹*N* ∝ *C*›)
(*auto simp*: *arena-act-pre-def arena-is-valid-clause-idx-def*
arena-lifting arena-dom-status-iff(*1*)
arena-lit-pre-def all-atms-def all-lits-def
ran-m-def all-lits-of-mm-add-mset all-lits-of-m-add-mset
simp flip: *arena-lifting*
dest: *valid-arena-nempty*
dest!: *multi-member-split*)
have *2*: ‹*C* ∈# *dom-m N* ⟹ (*arena-lit arena C*) ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
using *1* **by** (*cases* ‹*arena-lit arena C*›) (*auto simp*: $\mathcal{L}_{all}$-*add-mset dest*!: *multi-member-split*)
have [*simp*]: ‹*get-clause-LBD-pre arena C*› ‹*arena-act-pre arena C*›
‹*arena-is-valid-clause-vdom arena C*› ‹*marked-as-used-pre arena C*›
if ‹*C* ∈# *dom-m N*›
using *valid that* **by** (*auto simp*: *get-clause-LBD-pre-def arena-is-valid-clause-idx-def*
arena-act-pre-def arena-is-valid-clause-vdom-def marked-as-used-pre-def)
show *?thesis*
using *1 2 valid MM'*
unfolding *isa-is-candidate-for-removal-def is-candidate-for-removal-def*
get-the-propagation-reason-pol-def mop-arena-lbd-def
mop-arena-status-def mop-marked-as-used-def Let-def
by (*refine-vcg mop-arena-lit*(*1*)
mop-arena-length[*THEN fref-to-Down-curry*, *THEN order-trans*,*of N C - C vdom*])
(*auto simp*:
arena-lifting arena-dom-status-iff(*1*) *trail-pol-alt-def*
ann-lits-split-reasons-def
intro: *exI*[*of - ‹C*::*nat*›] *exI*[*of - vdom*]
dest: *valid-arena-nempty*)
qed

lemma *isa-gather-candidates-for-reduction-remove-deleted-clauses-from-avdom*:
‹*valid-arena arena N* (*set vdom*) ⟹ *mset* (*get-avdom-aivdom avdom0*) ⊆# *mset vdom* ⟹
(*M*, *M'*) ∈ *trail-pol* $\mathcal{A}$ ⟹ *set-mset* (*all-atms N NUE*) = *set-mset* $\mathcal{A}$ ⟹
length (*get-avdom-aivdom avdom0*) ≤ *length* (*get-vdom-aivdom avdom0*) ⟹

distinct vdom ⟹
isa-gather-candidates-for-reduction M arena avdom0 ≤
⇓{((*arena'*, *st*), *st'*). (*st*, *st'*) ∈ *Id* ∧ *length arena'* = *length arena* ∧ *valid-arena arena' N* (*set vdom*)}
(*remove-deleted-clauses-from-avdom N avdom0*)›
unfolding *isa-gather-candidates-for-reduction-def remove-deleted-clauses-from-avdom-def Let-def*
apply (*refine-vcg WHILEIT-refine*[**where** *R*= ‹{((*arena'*, *st*), *st'*). (*st*, *st'*) ∈ *Id* ∧ *length arena'* = *length arena* ∧ *valid-arena arena' N* (*set vdom*)}›]
)
subgoal by (*auto dest*!: *valid-arena-vdom-le*(*2*) *size-mset-mono simp*: *distinct-card*)
subgoal by *auto*
subgoal by (*auto simp*: *remove-deleted-clauses-from-avdom-inv-def*)
subgoal by (*auto simp*: *remove-deleted-clauses-from-avdom-inv-def*)
subgoal by *auto*
subgoal by *auto*
subgoal for *x x' x1 x2 x1a x2a x1b x2b x1c x2c* **unfolding** *arena-is-valid-clause-vdom-def*
apply (*auto intro*!: *exI*[*of - N*] *exI*[*of - ‹set vdom›*] *dest*!: *mset-eq-setD dest*: *mset-le-add-mset*
simp: *Cons-nth-drop-Suc*[*symmetric*] *remove-deleted-clauses-from-avdom-inv-def*)
by (*meson in-multiset-in-set mset-subset-eqD union-single-eq-member*)
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *remove-deleted-clauses-from-avdom-inv-def*)
subgoal
apply (*auto simp*: *arena-lifting arena-dom-status-iff*(*1*) *Cons-nth-drop-Suc*[*symmetric*] *remove-deleted-clauses-from-av*
dest!: *mset-eq-setD dest*: *mset-le-add-mset*)
by (*metis arena-dom-status-iff*(*1*) *mset-subset-eqD set-mset-mset union-single-eq-member*)
subgoal by (*auto simp*: *arena-act-pre-def arena-is-valid-clause-idx-def*)
apply (*rule isa-is-candidate-for-removal-is-candidate-for-removal*; *assumption?*; *auto intro*!: *valid-arena-mark-unused*; *fail*)
subgoal by *auto*
subgoal by (*auto intro*!: *valid-arena-mark-unused*)
subgoal by (*auto intro*!: *valid-arena-mark-unused*)
subgoal by (*auto intro*!: *valid-arena-mark-unused*)
subgoal by (*auto intro*!: *valid-arena-mark-unused*)
subgoal by (*auto intro*!: *valid-arena-mark-unused*)
done

lemma *bind-result-subst-iff*:
‹*P* ≤ ⇓ {(*a*,*b*). (*f a*, *b*) ∈ *R*} *g* ⟷
do {
a ← *P*;
RETURN (*f a*)
} ≤ ⇓ *R g*›
by (*cases P*; *cases g*)
(*auto simp*: *RETURN-def RES-RES-RETURN-RES conc-fun-RES*)

lemma *isa-isa-gather-candidates-for-reduction-remove-deleted-clauses-from-avdom2*:
assumes ‹*valid-arena arena N* (*set* (*get-vdom-aivdom avdom*))›
‹*aivdom-inv-dec avdom* (*dom-m N*)› **and**
MM': ‹(*M*, *M'*) ∈ *trail-pol* 𝒜› **and**
𝒜: ‹*set-mset* (*all-atms N NUE*) = *set-mset* 𝒜›
shows
‹*isa-gather-candidates-for-reduction M arena avdom* ≤
(*SPEC* (λ(*arena'*, *aivdom*). *aivdom-inv-dec aivdom* (*dom-m N*) ∧
get-vdom-aivdom aivdom = *get-vdom-aivdom avdom*∧
get-ivdom-aivdom aivdom = *get-ivdom-aivdom avdom*∧

```
    mset (get-tvdom-aivdom aivdom) ⊆# mset (get-avdom-aivdom avdom) ∧
   valid-arena arena' N (set (get-vdom-aivdom avdom)) ∧
   length arena' = length arena ∧
   (∀ C ∈ set (get-tvdom-aivdom aivdom). C ∈# dom-m N ∧ ¬irred N C ∧ length (N ∝ C) ≠ 2))›
proof −
  have i: ‹mset (get-avdom-aivdom avdom) ⊆# mset (get-vdom-aivdom avdom)›
    ‹distinct (get-vdom-aivdom avdom)›
    ‹length (get-avdom-aivdom avdom) ≤ length (get-vdom-aivdom avdom)›
    using assms(2) by (auto simp: aivdom-inv-dec-alt-def dest: size-mset-mono)
  have H: ‹aivdom-inv (get-vdom-aivdom avdom, get-avdom-aivdom avdom, get-ivdom-aivdom avdom,
get-tvdom-aivdom avdom) (dom-m N)›
    using assms(2) by (cases avdom) (auto simp: aivdom-inv-dec-def simp del: aivdom-inv.simps)
  show ?thesis
    apply (rule order-trans)
    apply (rule isa-gather-candidates-for-reduction-remove-deleted-clauses-from-avdom)
    apply (rule assms i)+
    apply (rule order-trans)
    apply (rule ref-two-step'[OF remove-deleted-clauses-from-avdom])
    apply (rule assms)
    apply (auto simp: conc-fun-RES)
    done
qed

lemma mset-inter-eqD: ‹x1m ∩# af = xa ∩# af ⟹
    set-mset x1m ∩ set-mset (af) = set-mset xa ∩ set-mset af› for x1m af xa
    by auto
      (metis Int-iff comp-apply set-mset-comp-mset set-mset-inter)+

lemma aivdom-inv-cong2:
  ‹mset vdom = mset vdom' ⟹ mset avdom = mset avdom' ⟹ mset ivdom = mset ivdom' ⟹ mset
tvdom = mset tvdom' ⟹
    aivdom-inv (vdom, avdom, ivdom, tvdom) b ⟹ aivdom-inv (vdom', avdom', ivdom', tvdom') b›
  by (auto 3 3 simp: dest: same-mset-distinct-iff mset-eq-setD)

lemma aivdom-inv-dec-cong2:
  ‹aivdom-inv-dec aivdom b ⟹ mset (get-vdom-aivdom aivdom) = mset (get-vdom-aivdom aivdom')
⟹
  mset (get-avdom-aivdom aivdom) = mset (get-avdom-aivdom aivdom') ⟹
  mset (get-ivdom-aivdom aivdom) = mset (get-ivdom-aivdom aivdom') ⟹
  mset (get-tvdom-aivdom aivdom) = mset (get-tvdom-aivdom aivdom') ⟹ aivdom-inv-dec aivdom' b›
  using aivdom-inv-cong2[of ‹get-vdom-aivdom aivdom› ‹get-vdom-aivdom aivdom'›
    ‹get-avdom-aivdom aivdom› ‹get-avdom-aivdom aivdom'›
    ‹get-ivdom-aivdom aivdom› ‹get-ivdom-aivdom aivdom'›
    ‹get-tvdom-aivdom aivdom› ‹get-tvdom-aivdom aivdom'› b]
  by (cases aivdom; cases aivdom') (auto simp: aivdom-inv-dec-def simp del: aivdom-inv.simps)

lemma sort-clauses-by-score-reorder:
  assumes
    ‹valid-arena arena N (set (get-vdom-aivdom vdom))› and
    ‹aivdom-inv-dec vdom (dom-m N)›
  shows ‹sort-clauses-by-score arena vdom
      ≤ SPEC
      (λvdom'.
       mset (get-avdom-aivdom vdom) = mset (get-avdom-aivdom vdom') ∧
       mset (get-vdom-aivdom vdom) = mset (get-vdom-aivdom vdom') ∧
       mset (get-ivdom-aivdom vdom) = mset (get-ivdom-aivdom vdom') ∧
```

mset (*get-tvdom-aivdom vdom*) = *mset* (*get-tvdom-aivdom vdom'*) ∧
aivdom-inv-dec vdom' (*dom-m N*))›
proof −
have ‹*set* (*get-avdom-aivdom vdom*) ⊆ *set* (*get-vdom-aivdom vdom*)›
using *assms*(*2*)
by (*auto simp*: *aivdom-inv-dec-alt-def*)
then show *?thesis*
using *assms*
unfolding *sort-clauses-by-score-def*
apply *refine-vcg*
unfolding *valid-sort-clause-score-pre-def arena-is-valid-clause-vdom-def*
get-clause-LBD-pre-def arena-is-valid-clause-idx-def arena-act-pre-def
valid-sort-clause-score-pre-at-def
apply (*auto simp*: *valid-sort-clause-score-pre-def twl-st-heur-restart-ana-def arena-dom-status-iff*(*2*−)
arena-dom-status-iff(*1*)[*symmetric*] *in-set-conv-nth*
arena-act-pre-def get-clause-LBD-pre-def arena-is-valid-clause-idx-def twl-st-heur-restart-def
intro: *aivdom-inv-dec-cong2 dest*!: *set-mset-mono mset-subset-eqD*)
using *arena-dom-status-iff*(*1*) **apply** *force*
using *arena-dom-status-iff*(*1*) **by** (*smt* (*verit*, *best*) *aivdom-inv-dec-alt-def mset-subset-eqD nth-mem set-mset-mset*)
qed

lemma *specify-left-RES*:
assumes *m* ≤ *RES* Φ
assumes ⋀*x*. *x* ∈ Φ ⟹ *f x* ≤ *M*
shows *do* { *x* ← *m*; *f x* } ≤ *M*
using *assms* **by** (*auto simp*: *pw-le-iff refine-pw-simps*)

lemma *sort-vdom-heur-reorder-vdom-wl*:
‹(*sort-vdom-heur*, *reorder-vdom-wl*) ∈ *twl-st-heur-restart-ana r* →*f* ⟨{(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart-ana r* ∧
(∀ *C* ∈ *set* (*get-tvdom S*). *C* ∈# *dom-m* (*get-clauses-wl T*) ∧ ¬*irred* (*get-clauses-wl T*) *C* ∧
length (*get-clauses-wl T* ∝ *C*) ≠ *2*)}⟩*nres-rel*›
proof −
have *mark-to-delete-clauses-wl-pre-same-atms*: ‹*set-mset* (*all-atms-st T*) = *set-mset* (*all-init-atms-st T*)›
if ‹*mark-to-delete-clauses-wl-pre T*› **for** *T*
using *that* **unfolding** *mark-to-delete-clauses-wl-pre-def mark-to-delete-clauses-l-pre-def* **apply** −
apply *normalize-goal*+
by (*rule literals-are-*$\mathcal{L}_{in}$*'-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*symmetric*]) *assumption*+

show *?thesis*
unfolding *reorder-vdom-wl-def sort-vdom-heur-def Let-def*
apply (*intro frefI nres-relI*)
apply *refine-rcg*
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def aivdom-inv-dec-alt-def*
dest!: *valid-arena-vdom-subset size-mset-mono*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def aivdom-inv-dec-alt-def*
dest!: *valid-arena-vdom-subset size-mset-mono*)
apply (*rule specify-left-RES*)
apply (*rule-tac* *N* = ‹*get-clauses-wl y*› **and** *M'* = ‹*get-trail-wl y*› **and**
A = ‹*all-init-atms-st y*› **and**
NUE = ‹*get-unit-clauses-wl y* + *get-subsumed-clauses-wl y* + *get-clauses0-wl y*› **in**
isa-isa-gather-candidates-for-reduction-remove-deleted-clauses-from-avdom2[*unfolded conc-fun-RES*])
subgoal for *x y*
by (*case-tac y*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case*)

subgoal for *x y*
by (*case-tac y*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*
mem-Collect-eq prod.case)
subgoal for *x y*
by (*case-tac y*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case all-init-atms-st-def*)
subgoal for *x y*
unfolding *all-atms-st-def*[*symmetric*]
by (*rule mark-to-delete-clauses-wl-pre-same-atms*)
apply (*subst case-prod-beta*)
apply (*subst assert-bind-spec-conv*, *intro conjI*)
subgoal for *x y*
unfolding *valid-sort-clause-score-pre-def arena-is-valid-clause-vdom-def*
get-clause-LBD-pre-def arena-is-valid-clause-idx-def arena-act-pre-def
by (*intro impI ballI*)
(*auto intro*!: *exI*[*of - ‹get-clauses-wl y›*] *exI*[*of - ‹set (get-vdom x)›*]
simp: *valid-sort-clause-score-pre-def twl-st-heur-restart-ana-def arena-dom-status-iff*(*2*−)
arena-dom-status-iff(*1*)[*symmetric*]
arena-act-pre-def get-clause-LBD-pre-def arena-is-valid-clause-idx-def twl-st-heur-restart-def)
apply (*subst assert-bind-spec-conv*, *intro conjI*)
subgoal by *auto*
apply (*subst assert-bind-spec-conv*, *intro conjI*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def aivdom-inv-dec-alt-def*
dest!: *valid-arena-vdom-subset size-mset-mono*)
subgoal for *x y x1*
apply (*rewrite at ‹- ≤ ⌑›* *Down-id-eq*[*symmetric*])
apply (*rule bind-refine-spec*)
prefer *2*
apply (*rule sort-clauses-by-score-reorder*[*of - ‹get-clauses-wl y›*])
subgoal
by (*clarsimp simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*: *mset-eq-setD*)
subgoal
by (*case-tac ‹x1›*; *case-tac ‹get-content (snd x1)›*) *simp*
subgoal
by (*auto 5 3 simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def intro*: *aivdom-inv-dec-cong2*
dest: *mset-eq-setD*)
done
done
qed

lemma (**in** −) *insort-inner-clauses-by-score-invI*:
‹*valid-sort-clause-score-pre a ba* ⟹
mset ba = *mset a2'* ⟹
a1' < *length a2'* ⟹
valid-sort-clause-score-pre-at a (*a2'* ! *a1'*)›
unfolding *valid-sort-clause-score-pre-def all-set-conv-nth valid-sort-clause-score-pre-at-def*
by (*metis in-mset-conv-nth*)+

lemma *sort-clauses-by-score-invI*:
‹*valid-sort-clause-score-pre a b* ⟹
mset b = *mset a2'* ⟹ *valid-sort-clause-score-pre a a2'*›
using *mset-eq-setD* **unfolding** *valid-sort-clause-score-pre-def* **by** *blast*

lemma *valid-sort-clause-score-pre-swap*:

‹*valid-sort-clause-score-pre a b* ⟹ *x* < *length b* ⟹
ba < *length b* ⟹ *valid-sort-clause-score-pre a* (*swap b x ba*)›
by (*auto simp*: *valid-sort-clause-score-pre-def*)

lemma *mark-to-delete-clauses-wl-post-alt-def*:
‹*mark-to-delete-clauses-wl-post S0 S* ⟷
(∃ *T0 T*.
(*S0*, *T0*) ∈ *state-wl-l None* ∧
(*S*, *T*) ∈ *state-wl-l None* ∧
blits-in-$\mathcal{L}_{in}$ *S0* ∧
blits-in-$\mathcal{L}_{in}$ *S* ∧
get-unkept-learned-clss-wl S = {#} ∧
get-subsumed-learned-clauses-wl S = {#} ∧
get-learned-clauses0-wl S = {#} ∧
(∃ *U0 U*. (*T0*, *U0*) ∈ *twl-st-l None* ∧
(*T*, *U*) ∈ *twl-st-l None* ∧
remove-one-annot-true-clause** *T0 T* ∧
twl-list-invs T0 ∧
twl-struct-invs U0 ∧
twl-list-invs T ∧
twl-struct-invs U ∧
get-conflict-l T0 = *None* ∧
clauses-to-update-l T0 = {#}) ∧
correct-watching S0 ∧ *correct-watching S*)›
unfolding *mark-to-delete-clauses-wl-post-def mark-to-delete-clauses-l-post-def*
mark-to-delete-clauses-l-pre-def
apply (*rule iffI*; *normalize-goal*+)
subgoal for *T0 T U0*
apply (*rule exI*[*of* - *T0*])
apply (*rule exI*[*of* - *T*])
apply (*intro conjI*)
apply *auto*[*7*]
apply (*rule exI*[*of* - *U0*])
apply *auto*
using *rtranclp-remove-one-annot-true-clause-cdcl-twl-restart-l2*[*of T0 T U0*]
rtranclp-cdcl-twl-restart-l-list-invs[*of T0*]
apply (*auto dest*:)
using *rtranclp-cdcl-twl-restart-l-list-invs* **by** *blast*
subgoal for *T0 T U0 U*
apply (*rule exI*[*of* - *T0*])
apply (*rule exI*[*of* - *T*])
apply (*intro conjI*)
apply *auto*[*3*]
apply (*rule exI*[*of* - *U0*])
apply *auto*
done
done

lemma *mark-to-delete-clauses-wl-D-heur-pre-alt-def*:
‹(∃ *S'*. (*S*, *S'*) ∈ *twl-st-heur* ∧ *mark-to-delete-clauses-wl-pre S'*) ⟹
mark-to-delete-clauses-wl-D-heur-pre S› (**is** ‹*?pre* ⟹ *?A*›) **and**
mark-to-delete-clauses-wl-D-heur-pre-twl-st-heur:
‹*mark-to-delete-clauses-wl-pre T* ⟹
(*S*, *T*) ∈ *twl-st-heur* ⟹ (*S*, *T*) ∈ *twl-st-heur-restart*› (**is** ‹- ⟹ - ⟹ *?B*›) **and**
mark-to-delete-clauses-wl-post-twl-st-heur:

```
    ‹mark-to-delete-clauses-wl-post T0 T ⟹
      (S, T) ∈ twl-st-heur-restart ⟹ (clss-size-resetUS0-st S, T) ∈ twl-st-heur› (is ‹- ⟹ ?Cpre ⟹
?C›)
proof −
  note cong = trail-pol-cong heuristic-rel-cong
    option-lookup-clause-rel-cong D₀-cong isa-vmtf-cong phase-saving-cong
    cach-refinement-empty-cong vdom-m-cong isasat-input-nempty-cong
    isasat-input-bounded-cong empty-occs-list-cong

  show ?A if ?pre
    using that apply −
    supply [[goals-limit=1]]
    unfolding mark-to-delete-clauses-wl-D-heur-pre-def mark-to-delete-clauses-wl-pre-def
      mark-to-delete-clauses-l-pre-def
    apply normalize-goal+
    subgoal for T U V
      using literals-are-ℒ_in'-literals-are-ℒ_in-iff(3)[of T U V]
        cong[of ‹all-atms-st T› ‹all-init-atms-st T›]
 vdom-m-cong[of ‹all-atms-st T› ‹all-init-atms-st T› ‹get-watched-wl T› ‹get-clauses-wl T›]
      apply −
      apply (rule exI[of - T])
      apply (intro conjI) defer
      apply (rule exI[of - U])
      apply (intro conjI) defer
      apply (rule exI[of - V])
      apply (simp-all del: isasat-input-nempty-def isasat-input-bounded-def)
      apply (cases S; cases T)
      by (auto simp add: twl-st-heur-def twl-st-heur-restart-def all-init-atms-st-def
        intro: clss-size-corr-restart-clss-size-corr(1)
        simp del: isasat-input-nempty-def)
    done
  show ‹mark-to-delete-clauses-wl-pre T ⟹ (S, T) ∈ twl-st-heur ⟹ ?B›
    supply [[goals-limit=1]]
    unfolding mark-to-delete-clauses-wl-D-heur-pre-def mark-to-delete-clauses-wl-pre-def
      mark-to-delete-clauses-l-pre-def mark-to-delete-clauses-wl-pre-def
    apply normalize-goal+
    subgoal for U V
      using literals-are-ℒ_in'-literals-are-ℒ_in-iff(3)[of T U V]
        cong[of ‹all-atms-st T› ‹all-init-atms-st T›]
 vdom-m-cong[of ‹all-atms-st T› ‹all-init-atms-st T› ‹get-watched-wl T› ‹get-clauses-wl T›]
      apply −
      apply (simp-all del: isasat-input-nempty-def isasat-input-bounded-def)
      apply (cases S; cases T)
      by (auto simp add: twl-st-heur-def twl-st-heur-restart-def all-init-atms-st-def all-atms-st-def
        intro: clss-size-corr-restart-clss-size-corr(1)
        simp del: isasat-input-nempty-def)
    done
  show ‹mark-to-delete-clauses-wl-post T0 T ⟹ ?Cpre ⟹ ?C›
    supply [[goals-limit=1]]
    unfolding mark-to-delete-clauses-wl-post-alt-def
    apply normalize-goal+
    subgoal for U0 U V0 V
      using literals-are-ℒ_in'-literals-are-ℒ_in-iff(3)[of T U V]
        cong[of ‹all-init-atms-st T› ‹all-atms-st T›]
 vdom-m-cong[of ‹all-init-atms-st T› ‹all-atms-st T› ‹get-watched-wl T› ‹get-clauses-wl T›]
      apply −
```

apply (*simp-all del*: *isasat-input-nempty-def isasat-input-bounded-def*)
apply (*cases S*; *cases T*)
by (*auto simp add*: *twl-st-heur-def twl-st-heur-restart-def all-init-atms-st-def all-atms-st-def*
clss-size-resetUS0-st-def
simp del: *isasat-input-nempty-def*
intro: *clss-size-corr-restart-clss-size-corr*(*2*))
done
qed

lemma *find-largest-lbd-and-size*:
assumes
‹(*S*,*T*) ∈ *twl-st-heur-restart-ana*′ *r u*›
shows ‹*find-largest-lbd-and-size l S* ≤*SPEC* (λ-. *True*)›
proof −
have *arena*: ‹*valid-arena* (*get-clauses-wl-heur S*) (*get-clauses-wl T*) (*set* (*get-vdom S*))› **and**
avdom: ‹*aivdom-inv-dec* (*get-aivdom S*) (*dom-m* (*get-clauses-wl T*))›
using *assms* **unfolding** *twl-st-heur-restart-ana-def twl-st-heur-restart-def* **by** *auto*

have [*intro*!]: ‹*clause-not-marked-to-delete-heur-pre* (*S*, *get-tvdom S* ! *i*)›
‹¬ ¬ *clause-not-marked-to-delete-heur S* (*get-tvdom S* ! *i*) ⟹ *get-clause-LBD-pre* (*get-clauses-wl-heur S*) (*get-tvdom S* ! *i*)›
‹¬ ¬ *clause-not-marked-to-delete-heur S* (*get-tvdom S* ! *i*) ⟹ *arena-is-valid-clause-idx* (*get-clauses-wl-heur S*) (*get-tvdom S* ! *i*)›
if ‹*i* < *length* (*get-tvdom S*)›
for *i*
using *arena avdom multi-member-split*[*of* ‹*get-tvdom S* ! *i*› ‹*mset* (*get-tvdom S*)›] *that*
arena-dom-status-iff(*1*)[*OF arena*, *of* ‹*get-tvdom S* ! *i*›]
unfolding *clause-not-marked-to-delete-heur-pre-def arena-is-valid-clause-vdom-def*
aivdom-inv-dec-alt-def get-clause-LBD-pre-def arena-is-valid-clause-idx-def
clause-not-marked-to-delete-heur-def
by (*auto intro*!: *exI*[*of* - ‹*get-clauses-wl T*›] *exI*[*of* - ‹*set* (*get-vdom S*)›]
simp: *arena-lifting*)
have *le*: ‹*length* (*get-tvdom S*) ≤ *length* (*get-clauses-wl-heur S*)›
using *avdom valid-arena-vdom-le*[*OF arena*] **unfolding** *aivdom-inv-dec-alt-def* **by** (*auto simp*: *distinct-card dest*: *size-mset-mono*)

show *?thesis*
unfolding *find-largest-lbd-and-size-def mop-clause-not-marked-to-delete-heur-def nres-monad3*
mop-arena-lbd-st-def mop-arena-lbd-def mop-arena-length-st-def mop-arena-length-def
apply (*refine-vcg WHILEIT-rule*[**where** *R* = ‹*measure* (λ(*i*, -, -). *length* (*get-tvdom S*) − *i*)›])
subgoal by (*rule le*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *access-tvdom-at-pre-def*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

```
lemma mark-to-delete-clauses-wl-D-heur-alt-def:
  ‹mark-to-delete-clauses-wl-D-heur = (λS0. do {
      ASSERT (mark-to-delete-clauses-wl-D-heur-pre S0);
      S ← sort-vdom-heur S0;
      - ← RETURN (get-tvdom S);
      l ← number-clss-to-keep S;
      (lbd, sze) ← find-largest-lbd-and-size l S;
      ASSERT
          (length (get-tvdom S) ≤ length (get-clauses-wl-heur S0));
      (i, T) ←
        WHILE_T^λ-. True (λ(i, S). i < length (get-tvdom S))
        (λ(i, T). do {
            ASSERT (i < length (get-tvdom T));
            ASSERT (access-tvdom-at-pre T i);
            ASSERT
                (clause-not-marked-to-delete-heur-pre
                  (T, get-tvdom T ! i));
            b ← mop-clause-not-marked-to-delete-heur T
                (get-tvdom T ! i);
            if ¬b then RETURN (i, delete-index-vdom-heur i T)
            else do {
                ASSERT
                    (access-lit-in-clauses-heur-pre
                      ((T, get-tvdom T ! i), 0));
                ASSERT
                    (length (get-clauses-wl-heur T)
                     ≤ length (get-clauses-wl-heur S0));
                ASSERT
                    (length (get-tvdom T)
                     ≤ length (get-clauses-wl-heur T));
                L ← mop-access-lit-in-clauses-heur T
                    (get-tvdom T ! i) 0;
                D ← get-the-propagation-reason-pol
                    (get-trail-wl-heur T) L;
                ASSERT
                    (arena-is-valid-clause-idx
                      (get-clauses-wl-heur T) (get-tvdom T ! i));
                let can-del = (D ≠ Some (get-tvdom T ! i) ∧
                   arena-length (get-clauses-wl-heur T) (get-tvdom T ! i) ≠ 2);
                if can-del
                then do {
                     wasted ← mop-arena-length-st T (get-tvdom T ! i);
                     - ← log-del-clause-heur T (get-tvdom T ! i);
                      ASSERT(mark-garbage-pre
                        (get-clauses-wl-heur T, get-tvdom T ! i) ∧
                        1 ≤ clss-size-lcount (get-learned-count T) ∧ i < length (get-tvdom T));
                      RETURN
                    (i, mark-garbage-heur3 (get-tvdom T ! i) i (incr-wasted-st (of-nat wasted) T))
                    }
                else do {
                     RETURN (i + 1, T)
                    }
            }
        })
      (l, S);
```

```
        ASSERT
            (length (get-tvdom T) ≤ length (get-clauses-wl-heur S0));
        incr-reduction-stat (set-stats-size-limit-st lbd sze T)
      })›
  unfolding mark-to-delete-clauses-wl-D-heur-def
    mop-arena-lbd-def mop-arena-status-def mop-arena-length-def
    mop-marked-as-used-def bind-to-let-conv Let-def
    nres-monad3 mop-mark-garbage-heur3-def mop-mark-unused-st-heur-def
    incr-wasted-st-twl-st
  by (auto intro!: ext bind-cong[OF refl])


lemma mark-to-delete-clauses-GC-wl-D-heur-alt-def:
  ‹mark-to-delete-clauses-GC-wl-D-heur = (λS0. do {
      ASSERT (mark-to-delete-clauses-GC-wl-D-heur-pre S0);
      S ← sort-vdom-heur S0;
      - ← RETURN (get-tvdom S);
      l ← number-clss-to-keep S;
      (lbd, sze) ← find-largest-lbd-and-size l S;
      ASSERT
          (length (get-tvdom S) ≤ length (get-clauses-wl-heur S0));
      (i, T) ←
        WHILE_T^λ-. True (λ(i, S). i < length (get-tvdom S))
        (λ(i, T). do {
            ASSERT (i < length (get-tvdom T));
            ASSERT (access-tvdom-at-pre T i);
            ASSERT
                (clause-not-marked-to-delete-heur-pre
                  (T, get-tvdom T ! i));
            b ← mop-clause-not-marked-to-delete-heur T
                (get-tvdom T ! i);
            if ¬b then RETURN (i, delete-index-vdom-heur i T)
            else do {
                ASSERT
                    (access-lit-in-clauses-heur-pre
                      ((T, get-tvdom T ! i), 0));
                ASSERT
                    (length (get-clauses-wl-heur T)
                     ≤ length (get-clauses-wl-heur S0));
                ASSERT
                    (length (get-tvdom T)
                     ≤ length (get-clauses-wl-heur T));
                ASSERT
                    (arena-is-valid-clause-idx
                      (get-clauses-wl-heur T) (get-tvdom T ! i));
                let can-del = (arena-length (get-clauses-wl-heur T) (get-tvdom T ! i) ≠ 2);
                if can-del
                then do {
                    wasted ← mop-arena-length-st T (get-tvdom T ! i);
                    - ← log-del-clause-heur T (get-tvdom T ! i);
                    ASSERT(mark-garbage-pre
                      (get-clauses-wl-heur T, get-tvdom T ! i) ∧
                      1 ≤ clss-size-lcount (get-learned-count T)∧ i < length (get-tvdom T));
                    RETURN
                   (i, mark-garbage-heur3 (get-tvdom T ! i) i (incr-wasted-st (of-nat wasted) T))
                   }
```

```
                    else do {
                          RETURN
                           (i + 1, T)
                        }
                    }
                })
              (l, S);
          ASSERT
              (length (get-tvdom T) ≤ length (get-clauses-wl-heur S0));
          incr-restart-stat (set-stats-size-limit-st lbd sze T)
        })›
    unfolding mark-to-delete-clauses-GC-wl-D-heur-def
      mop-arena-lbd-def mop-arena-status-def mop-arena-length-def
      mop-marked-as-used-def bind-to-let-conv Let-def
      nres-monad3 mop-mark-garbage-heur3-def mop-mark-unused-st-heur-def
      incr-wasted-st-twl-st
    by (auto intro!: ext intro!: bind-cong[OF refl])

lemma learned-clss-count-mark-garbage-heur3:
  ‹clss-size-lcount (get-learned-count xs) ≥ Suc 0 ⟹ learned-clss-count (mark-garbage-heur3 C i xs) =
(learned-clss-count xs) − 1› and
  learned-clss-count-incr-st[simp]:
  ‹learned-clss-count (incr-wasted-st b xs) = learned-clss-count xs›
  by (cases xs; auto simp: mark-garbage-heur3-def incr-wasted-st-def learned-clss-count-def; fail)+

lemma mark-to-delete-clauses-wl-D-heur-mark-to-delete-clauses-wl-D:
  ‹(mark-to-delete-clauses-wl-D-heur, mark-to-delete-clauses-wl) ∈
     twl-st-heur-restart-ana' r u →f ⟨twl-st-heur-restart-ana' r u⟩nres-rel›
proof −
  have mark-to-delete-clauses-wl-D-alt-def:
    ‹mark-to-delete-clauses-wl = (λS0. do {
      ASSERT(mark-to-delete-clauses-wl-pre S0);
      S ← reorder-vdom-wl S0;
      xs ← collect-valid-indices-wl S;
      l ← SPEC(λ-::nat. True);
      - ← SPEC(λ-::nat×nat. True);
      (-, S, -) ← WHILE_T^mark-to-delete-clauses-wl-inv S xs
        (λ(i, T, xs). i < length xs)
        (λ(i, T, xs). do {
          b ← RETURN (xs!i ∈# dom-m (get-clauses-wl T));
          if ¬b then RETURN (i, T, delete-index-and-swap xs i)
          else do {
            ASSERT(0 < length (get-clauses-wl T∝(xs!i)));
    ASSERT (get-clauses-wl T ∝ (xs ! i) ! 0 ∈# L_all (all-init-atms-st T));
          K ← RETURN (get-clauses-wl T ∝ (xs ! i) ! 0);
          b ← RETURN (); — propagation reason
          can-del ← SPEC(λb. b ⟶
            (Propagated (get-clauses-wl T∝(xs!i)!0) (xs!i) ∉ set (get-trail-wl T)) ∧
             ¬irred (get-clauses-wl T) (xs!i) ∧ length (get-clauses-wl T ∝ (xs!i)) ≠ 2);
          ASSERT(i < length xs);
          if can-del
          then do{
            - ← RETURN (length (get-clauses-wl T ∝ (xs!i)));
            - ← RETURN (log-clause T (xs!i));
            RETURN (i, mark-garbage-wl (xs!i) T, delete-index-and-swap xs i)}
          else
```

```
            RETURN (i+1, T, xs)
          }
        })
        (l, S, xs);
      remove-all-learned-subsumed-clauses-wl S
    })›
    unfolding mark-to-delete-clauses-wl-def reorder-vdom-wl-def bind-to-let-conv Let-def nres-monad3
      summarize-ASSERT-conv
    by (force intro!: ext)

  have mono: ‹g = g' ⟹ do {f; g} = do {f; g'}›
    ‹(⋀x. h x = h' x) ⟹ do {x ← f; h x} = do {x ← f; h' x}› for f f' :: ‹- nres› and g g' and h h'
    by auto
  have mark-to-delete-clauses-wl-pre-same-atms: ‹set-mset (all-atms-st T) = set-mset (all-init-atms-st
T)›
    if ‹mark-to-delete-clauses-wl-pre T› for T
    using that unfolding mark-to-delete-clauses-wl-pre-def mark-to-delete-clauses-l-pre-def apply −
    apply normalize-goal+
    by (rule literals-are-ℒ_in'-literals-are-ℒ_in-iff(3)[symmetric]) assumption+
  have mark-to-delete-clauses-wl-D-heur-pre-cong:
    ‹aivdom-inv-dec vdom' (dom-m (get-clauses-wl S')) ⟹
    mset (get-vdom-aivdom (get-aivdom T)) = mset (get-vdom-aivdom vdom') ⟹
    (T, S') ∈ twl-st-heur-restart ⟹
    mark-to-delete-clauses-wl-pre S' ⟹
    mark-to-delete-clauses-wl-D-heur-pre T ⟹
    valid-arena N'' (get-clauses-wl S') (set (get-vdom-aivdom (get-aivdom T))) ⟹
    mark-to-delete-clauses-wl-D-heur-pre (set-clauses-wl-heur N'' (set-aivdom-wl-heur vdom' T))›
    for M' N' D' j W' vm clvls cach lbd outl stats fast-ema slow-ema avdom avdom'
      ccount lcount heur old-arena ivdom opts S' vdom' N'' T
     using mset-eq-setD[of ‹get-vdom-aivdom (get-aivdom T)› ‹get-vdom-aivdom vdom'›, symmetric]
apply −
    unfolding mark-to-delete-clauses-wl-D-heur-pre-def apply normalize-goal+
    by (rule-tac x=S' in exI)
     (clarsimp simp: twl-st-heur-restart-def dest: mset-eq-setD intro: )

  have [refine0]:
    ‹sort-vdom-heur S ≤ ⇓ {(U, V). (U, V) ∈ twl-st-heur-restart-ana' r u ∧ V = T ∧
        (mark-to-delete-clauses-wl-pre T ⟶ mark-to-delete-clauses-wl-pre V) ∧
         (mark-to-delete-clauses-wl-D-heur-pre S ⟶ mark-to-delete-clauses-wl-D-heur-pre U) ∧
        (∀ C∈set (get-tvdom U). ¬irred (get-clauses-wl V) C ∧ length (get-clauses-wl V ∝ C) ≠ 2)}
        (reorder-vdom-wl T)› (is ‹- ≤ ⇓?sort -›)
    if ‹(S, T) ∈ twl-st-heur-restart-ana' r u› and ‹mark-to-delete-clauses-wl-pre T› for S T
    supply [simp del] = EQ-def
    using that unfolding reorder-vdom-wl-def sort-vdom-heur-def Let-def
    apply (refine-rcg ASSERT-leI)
    subgoal by (auto simp: twl-st-heur-restart-ana-def twl-st-heur-restart-def aivdom-inv-dec-alt-def
      dest!: valid-arena-vdom-subset size-mset-mono)
    subgoal by (auto simp: twl-st-heur-restart-ana-def twl-st-heur-restart-def aivdom-inv-dec-alt-def
      dest!: valid-arena-vdom-subset size-mset-mono)
      apply (rule specify-left-RES)
      apply (rule-tac  N = ‹get-clauses-wl T› and M' = ‹get-trail-wl T› and
        𝒜 = ‹all-init-atms-st T› and
        NUE = ‹get-unit-clauses-wl T + get-subsumed-clauses-wl T + get-clauses0-wl T› in
    isa-isa-gather-candidates-for-reduction-remove-deleted-clauses-from-avdom2[unfolded conc-fun-RES])
    subgoal
     by (case-tac T; auto simp: twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case)
```

subgoal
by (*case-tac T*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case*)
subgoal
by (*case-tac T*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case all-init-atms-st-def*)
subgoal
unfolding *all-atms-st-def*[*symmetric*]
by (*rule mark-to-delete-clauses-wl-pre-same-atms*)
apply (*subst case-prod-beta*)
apply (*intro ASSERT-leI*)
subgoal
unfolding *valid-sort-clause-score-pre-def arena-is-valid-clause-vdom-def get-clause-LBD-pre-def arena-is-valid-clause-idx-def arena-act-pre-def*
by (*auto simp*: *valid-sort-clause-score-pre-def twl-st-heur-restart-ana-def arena-dom-status-iff*(*1*)[*symmetric*] *arena-act-pre-def get-clause-LBD-pre-def arena-is-valid-clause-idx-def twl-st-heur-restart-def aivdom-inv-dec-alt-def*
intro!: *exI*[*of - ‹get-clauses-wl T›*] *exI*[*of - ‹set* (*get-vdom S*)›]
dest: *set-mset-mono mset-subset-eqD*)
subgoal by (*auto simp*: *EQ-def*)
subgoal
by (*cases T*)
(*clarsimp simp add*: *twl-st-heur-restart-ana-def valid-arena-vdom-subset twl-st-heur-restart-def aivdom-inv-dec-alt-def case-prod-beta split*:
dest!: *size-mset-mono valid-arena-vdom-subset*)
subgoal for *arena-aivdom*
apply (*rewrite at ‹- ≤ ⌑›* *Down-id-eq*[*symmetric*])
apply (*rule bind-refine-spec*)
prefer *2*
apply (*rule sort-clauses-by-score-reorder*[*of - ‹get-clauses-wl T›*])
subgoal
by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*: *mset-eq-setD*)
subgoal
by (*cases ‹arena-aivdom›*; *cases ‹get-content* (*snd arena-aivdom*)›)
(*simp add*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*)
subgoal
apply *auto*
apply (*auto simp*: *learned-clss-count-def*
intro: *mark-to-delete-clauses-wl-D-heur-pre-cong*
intro: *aivdom-inv-cong2*
dest: *mset-eq-setD*)
apply (*auto 4 3 simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*
intro: *aivdom-inv-cong2 dest*: *mset-eq-setD*; *fail*)[]
apply (*rule mark-to-delete-clauses-wl-D-heur-pre-cong*)
apply *assumption+*
apply (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*)
done
done
done

have [*refine0*]: ‹*RETURN* (*get-tvdom x*) ≤ ⇓ {(*xs*, *xs'*). *xs* = *xs'* ∧ *xs* = *get-tvdom x* ∧
(∀ *C*∈*set* (*get-tvdom x*). ¬*irred* (*get-clauses-wl y*) *C* ∧ *length* (*get-clauses-wl y* ∝ *C*) ≠ *2*)}
(*collect-valid-indices-wl y*)›
(**is** ‹- ≤ ⇓ *?indices* -›)
if
‹(*x*, *y*) ∈ *?sort S T*› **and**
‹*mark-to-delete-clauses-wl-D-heur-pre x*›

for *x y S T*
proof −
show *?thesis* **using** *that* **by** (*auto simp*: *collect-valid-indices-wl-def simp*: *RETURN-RES-refine-iff*)
qed

have *init*:
‹(*u′*, *xs*) ∈ *?indices S Sa* ⟹
(*l*, *la*) ∈ *nat-rel* ⟹
(*S*, *Sa*) ∈ *twl-st-heur-restart-ana′ r u* ⟹
((*l*, *S*), *la*, *Sa*, *xs*) ∈ *nat-rel* ×*f*
{(*Sa′*, (*T*, *xs*)). (*Sa′*, *T*) ∈ *twl-st-heur-restart-ana′ r u* ∧ *xs* = *get-tvdom Sa′* ∧
set (*get-tvdom Sa′*) ⊆ *set* (*get-tvdom S*) ∧
(∀ *C*∈*set* (*get-tvdom Sa′*). ¬*irred* (*get-clauses-wl T*) *C* ∧ *length* (*get-clauses-wl T* ∝ *C*) ≠ *2*)}›
(**is** ‹- ⟹ - ⟹ - ⟹ - ∈ *?R*›)
for *x y S Sa xs l la u′*
by *auto*

define *reason-rel* **where**
‹*reason-rel K x1a* ≡ {(*C*, - :: *unit*).
(*C* ≠ *None*) = (*Propagated K* (*the C*) ∈ *set* (*get-trail-wl x1a*)) ∧
(*C* = *None*) = (*Decided K* ∈ *set* (*get-trail-wl x1a*) ∨
K ∉ *lits-of-l* (*get-trail-wl x1a*)) ∧
(∀ *C1*. (*Propagated K C1* ∈ *set* (*get-trail-wl x1a*) ⟶ *C1* = *the C*))}› **for** *K* :: ‹*nat literal*› **and** *x1a*
have *get-the-propagation-reason*:
‹*get-the-propagation-reason-pol* (*get-trail-wl-heur x2b*) *L*
≤ *SPEC* (λ*D*. (*D*, ()) ∈ *reason-rel K x1a*)›
if
‹(*x*, *y*) ∈ *twl-st-heur-restart-ana′ r u*› **and**
‹*mark-to-delete-clauses-wl-pre y*› **and**
‹*mark-to-delete-clauses-wl-D-heur-pre x*› **and**
‹(*S*, *Sa*) ∈ *?sort x y*› **and**
‹(*ys*, *xs*) ∈ *?indices S Sa*› **and**
‹(*l*, *la*) ∈ *nat-rel*› **and**
‹*la* ∈ {-. *True*}› **and**
xa-x′: ‹(*xa*, *x′*) ∈ *?R S*› **and**
‹*case xa of* (*i*, *S*) ⇒ *i* < *length* (*get-tvdom S*)› **and**
‹*case x′ of* (*i*, *T*, *xs*) ⇒ *i* < *length xs*› **and**
‹*x1b* < *length* (*get-tvdom x2b*)› **and**
‹*access-tvdom-at-pre x2b x1b*› **and**
dom: ‹(*b*, *ba*)
∈ {(*b*, *b′*).
(*b*, *b′*) ∈ *bool-rel* ∧
b = (*x2a* ! *x1* ∈# *dom-m* (*get-clauses-wl x1a*))}›
‹¬ ¬ *b*›
‹¬ ¬ *ba*› **and**
‹*0* < *length* (*get-clauses-wl x1a* ∝ (*x2a* ! *x1*))› **and**
‹*access-lit-in-clauses-heur-pre* ((*x2b*, *get-tvdom x2b* ! *x1b*), *0*)› **and**
st:
‹*x2* = (*x1a*, *x2a*)›
‹*x′* = (*x1*, *x2*)›
‹*xa* = (*x1b*, *x2b*)› **and**
L: ‹*get-clauses-wl x1a* ∝ (*x2a* ! *x1*) ! *0* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1a*)› **and**
L′: ‹(*L*, *K*)
∈ {(*L*, *L′*).
(*L*, *L′*) ∈ *nat-lit-lit-rel* ∧

```
      L' = get-clauses-wl x1a ∝ (x2a ! x1) ! 0}›
    for x y S Sa xs' xs l la xa x' x1 x2 x1a x2a x1' x2' x3 x1b ys x2b L K b ba
  proof −
    have L: ‹arena-lit (get-clauses-wl-heur x2b) (x2a ! x1) ∈# \mathcal{L}_{all} (all-init-atms-st x1a)›
    using L that by (auto dest!: twl-st-heur-restart(2) simp: st arena-lifting dest: twl-st-heur-restart-anaD)

    show ?thesis
      apply (rule order.trans)
      apply (rule get-the-propagation-reason-pol[THEN fref-to-Down-curry,
        of ‹all-init-atms-st x1a› ‹get-trail-wl x1a›
   ‹arena-lit (get-clauses-wl-heur x2b) (get-tvdom x2b ! x1b + 0)›])
      subgoal
        using xa-x' L L' by (auto simp add: twl-st-heur-restart-def st)
      subgoal
          using xa-x' L' dom by (auto simp add: twl-st-heur-restart-ana-def twl-st-heur-restart-def st
arena-lifting
          all-init-atms-st-def get-unit-init-clss-wl-alt-def)
      using that unfolding get-the-propagation-reason-def reason-rel-def apply −
      by (auto simp: twl-st-heur-restart lits-of-def get-the-propagation-reason-def
          conc-fun-RES
        dest!: twl-st-heur-restart-anaD dest: twl-st-heur-restart(2)  twl-st-heur-restart-same-annotD im-
ageI[of - - lit-of])
  qed

  have already-deleted:
    ‹((x1b,  delete-index-vdom-heur x1b x2b), x1, x1a, delete-index-and-swap x2a x1) ∈ ?R S›
    if
      ‹(x, y) ∈ twl-st-heur-restart-ana' r u› and
      ‹mark-to-delete-clauses-wl-D-heur-pre x› and
      ‹(S, Sa) ∈ ?sort x y› and
      ‹(l, la) ∈ nat-rel› and
      ‹la ∈ {-. True}› and
      xx: ‹(xa, x') ∈ ?R S› and
      nempty: ‹case xa of (i, S) ⇒ i < length (get-tvdom S)› and
      ‹case x' of (i, T, xs) ⇒ i < length xs› and
      st:
      ‹x2 = (x1a, x2a)›
      ‹x' = (x1, x2)›
      ‹xa = (x1b, x2b)› and
      le: ‹x1b < length (get-tvdom x2b)› and
      ‹access-tvdom-at-pre x2b x1b› and
      ba: ‹(b, ba) ∈ {(b, b'). (b, b') ∈ bool-rel ∧ b = (x2a ! x1 ∈# dom-m (get-clauses-wl x1a))}›
        ‹¬ba›
    for x y S xs l la xa x' xz x1 x2 x1a x2a x2b x2c x2d ys x1b Sa ba b
  proof −
    show ?thesis
      using xx nempty le ba unfolding st
      by (cases ‹get-tvdom x2b› rule: rev-cases)
       (auto 4 3 simp: twl-st-heur-restart-ana-def delete-index-vdom-heur-def
          twl-st-heur-restart-def mark-garbage-heur-def mark-garbage-wl-def
          learned-clss-l-l-fmdrop size-remove1-mset-If learned-clss-count-def
          aivdom-inv-removed-inactive
        intro: valid-arena-extra-information-mark-to-delete'
        intro!: aivdom-inv-dec-removed-inactive-tvdom
        dest!: in-set-butlastD in-vdom-m-fmdropD
        elim!: in-set-upd-cases)
```

qed

have *mop-clause-not-marked-to-delete-heur*:
‹*mop-clause-not-marked-to-delete-heur x2b* (*get-tvdom x2b* ! *x1b*)
≤ *SPEC*
(λ*c*. (*c*, *x2a* ! *x1* ∈# *dom-m* (*get-clauses-wl x1a*))
∈ {(*b*, *b'*). (*b*,*b'*) ∈ *bool-rel* ∧ (*b* ⟷ *x2a* ! *x1* ∈# *dom-m* (*get-clauses-wl x1a*))})›
if
‹(*xa*, *x'*) ∈ *?R S*› **and**
‹*case xa of* (*i*, *S*) ⇒ *i* < *length* (*get-tvdom S*)› **and**
‹*case x' of* (*i*, *T*, *xs*) ⇒ *i* < *length xs*› **and**
‹*mark-to-delete-clauses-wl-inv Sa xs x'*› **and**
‹*x2* = (*x1a*, *x2a*)› **and**
‹*x'* = (*x1*, *x2*)› **and**
‹*xa* = (*x1b*, *x2b*)› **and**
‹*clause-not-marked-to-delete-heur-pre* (*x2b*, *get-tvdom x2b* ! *x1b*)›
for *x y S Sa uu xs l la xa x' x1 x2 x1a x2a x1b x2b*
unfolding *mop-clause-not-marked-to-delete-heur-def*
apply *refine-vcg*
subgoal
using *that* **by** *blast*
subgoal
using *that* **by** (*auto simp*: *twl-st-heur-restart arena-lifting dest*: *twl-st-heur-restart*(*2*) *dest*!: *twl-st-heur-restart-anaD*)
done

have *mop-access-lit-in-clauses-heur*:
‹*mop-access-lit-in-clauses-heur x2b* (*get-tvdom x2b* ! *x1b*) *0*
≤ *SPEC*
(λ*c*. (*c*, *get-clauses-wl x1a* ∝ (*x2a* ! *x1*) ! *0*)
∈ {(*L*, *L'*). (*L*, *L'*) ∈ *nat-lit-lit-rel* ∧ *L'* = *get-clauses-wl x1a* ∝ (*x2a* ! *x1*) ! *0*})›
if
‹(*x*, *y*) ∈ *twl-st-heur-restart-ana' r u*› **and**
‹*mark-to-delete-clauses-wl-pre y*› **and**
‹*mark-to-delete-clauses-wl-D-heur-pre x*› **and**
‹(*S*, *Sa*) ∈ *?sort x y*› **and**
‹(*uu*, *xs*) ∈ *?indices S Sa*› **and**
‹(*l*, *la*) ∈ *nat-rel*› **and**
‹*la* ∈ {*uu*. *True*}› **and**
‹*length* (*get-tvdom S*) ≤ *length* (*get-clauses-wl-heur x*)› **and**
‹(*xa*, *x'*) ∈ *?R S*› **and**
‹*case xa of* (*i*, *S*) ⇒ *i* < *length* (*get-tvdom S*)› **and**
‹*case x' of* (*i*, *T*, *xs*) ⇒ *i* < *length xs*› **and**
‹*mark-to-delete-clauses-wl-inv Sa xs x'*› **and**
‹*x2* = (*x1a*, *x2a*)› **and**
‹*x'* = (*x1*, *x2*)› **and**
‹*xa* = (*x1b*, *x2b*)› **and**
‹*x1b* < *length* (*get-tvdom x2b*)› **and**
‹*access-tvdom-at-pre x2b x1b*› **and**
‹*clause-not-marked-to-delete-heur-pre* (*x2b*, *get-tvdom x2b* ! *x1b*)› **and**
‹(*b*, *ba*)
∈ {(*b*, *b'*).
(*b*, *b'*) ∈ *bool-rel* ∧
b = (*x2a* ! *x1* ∈# *dom-m* (*get-clauses-wl x1a*))}› **and**
‹¬ ¬ *b*› **and**
‹¬ ¬ *ba*› **and**

‹0 < length (get-clauses-wl x1a ∝ (x2a ! x1))› **and**
‹get-clauses-wl x1a ∝ (x2a ! x1) ! 0
∈# $\mathcal{L}_{all}$ (all-init-atms-st x1a)› **and**
pre: ‹access-lit-in-clauses-heur-pre ((x2b, get-tvdom x2b ! x1b), 0)›
for x y S Sa uu xs l la xa x′ x1 x2 x1a x2a x1b x2b b ba
unfolding mop-access-lit-in-clauses-heur-def mop-arena-lit2-def
apply refine-vcg
subgoal using pre **unfolding** access-lit-in-clauses-heur-pre-def **by** simp
subgoal using that **by** (auto dest!: twl-st-heur-restart-anaD twl-st-heur-restart-valid-arena simp: arena-lifting)
done

have incr-restart-stat: ‹incr-reduction-stat (set-stats-size-limit-st lbd sze T)
≤ ⇓ (twl-st-heur-restart-ana′ r u) (remove-all-learned-subsumed-clauses-wl S)›
if ‹(T, S) ∈ twl-st-heur-restart-ana′ r u› **for** S T i u lbd sze
using that
by (cases S; cases T)
(auto simp: conc-fun-RES incr-restart-stat-def learned-clss-count-def set-stats-size-limit-st-def
twl-st-heur-restart-ana-def twl-st-heur-restart-def
remove-all-learned-subsumed-clauses-wl-def clss-size-corr-def incr-reduction-stat-def
clss-size-lcountUE-def clss-size-lcountUS-def clss-size-def
clss-size-resetUS-def clss-size-lcount-def clss-size-lcountU0-def
RES-RETURN-RES)

have only-irred: ‹¬ irred (get-clauses-wl x1a) (x2a ! x1)› (**is** ?A) **and**
get-learned-count-ge: ‹Suc 0 ≤ clss-size-lcount (get-learned-count x2b)› (**is** ?B)
if
‹(x, y) ∈ twl-st-heur-restart-ana′ r u› **and**
‹mark-to-delete-clauses-wl-pre y› **and**
‹mark-to-delete-clauses-wl-D-heur-pre x› **and**
‹(S, Sa) ∈ ?sort x y› **and**
indices: ‹(uu, xs) ∈ ?indices S Sa› **and**
‹(l, la) ∈ nat-rel› **and**
‹la ∈ {-. True}› **and**
‹length (get-tvdom S) ≤ length (get-clauses-wl-heur x)› **and**
xx: ‹(xa, x′) ∈ ?R S› **and**
‹case xa of (i, S) ⇒ i < length (get-tvdom S)› **and**
‹case x′ of (i, T, xs) ⇒ i < length xs› **and**
‹mark-to-delete-clauses-wl-inv Sa xs x′› **and**
st: ‹x2 = (x1a, x2a)› ‹x′ = (x1, x2)› ‹xa = (x1b, x2b)› **and**
‹x1b < length (get-tvdom x2b)› **and**
‹access-tvdom-at-pre x2b x1b› **and**
‹clause-not-marked-to-delete-heur-pre (x2b, get-tvdom x2b ! x1b)› **and**
dom: ‹(b, ba) ∈ {(b, b′). (b, b′) ∈ bool-rel ∧ b = (x2a ! x1 ∈# dom-m (get-clauses-wl x1a))}›
‹¬ ¬ b›
‹¬ ¬ ba› **and**
‹0 < length (get-clauses-wl x1a ∝ (x2a ! x1))› **and**
‹get-clauses-wl x1a ∝ (x2a ! x1) ! 0 ∈# $\mathcal{L}_{all}$ (all-init-atms-st x1a)› **and**
‹access-lit-in-clauses-heur-pre ((x2b, get-tvdom x2b ! x1b), 0)› **and**
‹length (get-clauses-wl-heur x2b) ≤ length (get-clauses-wl-heur x)› **and**
‹length (get-tvdom x2b) ≤ length (get-clauses-wl-heur x2b)› **and**
‹(L, K) ∈ {(L, L′). (L, L′) ∈ nat-lit-lit-rel ∧ L′ = get-clauses-wl x1a ∝ (x2a ! x1) ! 0}› **and**
‹(D, bb) ∈ reason-rel K x1a› **and**
‹arena-is-valid-clause-idx (get-clauses-wl-heur x2b) (get-tvdom x2b ! x1b)› **and**
‹D ≠ Some (get-tvdom x2b ! x1b) ∧ arena-length (get-clauses-wl-heur x2b) (get-tvdom x2b ! x1b) ≠ 2›

for *x y S Sa uu xs l la xa x′ x1 x2 x1a x2a x1b x2b b ba L K D bb*
proof −
have ‹*get-tvdom x2b* ! *x1* ∈ *set* (*get-tvdom x2b*)› **and**
x: ‹(*x2b*, *x1a*) ∈ *twl-st-heur-restart-ana r*›
using *that* **by** (*auto dest*: *simp*: *arena-lifting twl-st-heur-restart*)
then show *?A*
using *indices xx*
by (*auto dest*: *twl-st-heur-restart-anaD twl-st-heur-restart-valid-arena*
simp: *arena-lifting twl-st-heur-restart st*)
then show *?B*
using *dom x* **by** (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def ran-m-def*
dest!: *multi-member-split*
dest!: *clss-size-corr-restart-rew*)
qed
have *length-filter-le*: ‹((*x1b*, *mark-garbage-heur3* (*get-tvdom x2b* ! *x1b*) *x1b* (*incr-wasted-st* (*word-of-nat wasted*) *x2b*)), *x1*,
mark-garbage-wl (*x2a* ! *x1*) *x1a*, *delete-index-and-swap x2a x1*)
∈ *?R S*›
if *H*:
‹(*x*, *y*) ∈ *twl-st-heur-restart-ana′ r u*›
‹*mark-to-delete-clauses-wl-pre y*›
‹*mark-to-delete-clauses-wl-D-heur-pre x*›
‹(*S*, *Sa*) ∈ *?sort x y*›
‹(*uu*, *xs*) ∈ *?indices S Sa*›
‹(*l*, *la*) ∈ *nat-rel*›
‹*la* ∈ {-. *True*}›
‹*length* (*get-tvdom S*) ≤ *length* (*get-clauses-wl-heur x*)›
‹(*xa*, *x′*) ∈ *?R S*›
‹*case xa of* (*i*, *S*) ⇒ *i* < *length* (*get-tvdom S*)›
‹*case x′ of* (*i*, *T*, *xs*) ⇒ *i* < *length xs*›
‹*mark-to-delete-clauses-wl-inv Sa xs x′*›
‹*x2* = (*x1a*, *x2a*)›
‹*x′* = (*x1*, *x2*)›
‹*xa* = (*x1b*, *x2b*)›
‹*x1b* < *length* (*get-tvdom x2b*)›
‹*access-tvdom-at-pre x2b x1b*›
‹*clause-not-marked-to-delete-heur-pre* (*x2b*, *get-tvdom x2b* ! *x1b*)›
‹(*b*, *ba*) ∈ {(*b*, *b′*). (*b*, *b′*) ∈ *bool-rel* ∧ *b* = (*x2a* ! *x1* ∈# *dom-m* (*get-clauses-wl x1a*))}›
‹¬ ¬ *b*›
‹¬ ¬ *ba*›
‹*0* < *length* (*get-clauses-wl x1a* ∝ (*x2a* ! *x1*))›
‹*get-clauses-wl x1a* ∝ (*x2a* ! *x1*) ! *0* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1a*)›
‹*access-lit-in-clauses-heur-pre* ((*x2b*, *get-tvdom x2b* ! *x1b*), *0*)›
‹*length* (*get-clauses-wl-heur x2b*) ≤ *length* (*get-clauses-wl-heur x*)›
‹*length* (*get-tvdom x2b*) ≤ *length* (*get-clauses-wl-heur x2b*)›
‹(*L*, *K*) ∈ {(*L*, *L′*). (*L*, *L′*) ∈ *nat-lit-lit-rel* ∧ *L′* = *get-clauses-wl x1a* ∝ (*x2a* ! *x1*) ! *0*}›
‹(*D*, *bb*) ∈ *reason-rel K x1a*›
‹*arena-is-valid-clause-idx* (*get-clauses-wl-heur x2b*) (*get-tvdom x2b* ! *x1b*)›
‹(*D* ≠ *Some* (*get-tvdom x2b* ! *x1b*) ∧ *arena-length* (*get-clauses-wl-heur x2b*) (*get-tvdom x2b* ! *x1b*) ≠ *2*, *can-del*)
∈ *bool-rel*›
‹*x1* < *length x2a*›
‹*D* ≠ *Some* (*get-tvdom x2b* ! *x1b*) ∧ *arena-length* (*get-clauses-wl-heur x2b*) (*get-tvdom x2b* ! *x1b*) ≠ *2*›
can-del
for *x y S Sa xs l la xa x′ x1 x2 x1a x2a x1b x2b b ba can-del D L K bb uu wasted*

proof −
have [*simp*]: ‹*mark-garbage-heur3 i C* (*incr-wasted-st b S*) = *incr-wasted-st b* (*mark-garbage-heur3 i C S*)› **for** *i C S b*
by (*cases S*; *auto simp*: *mark-garbage-heur3-def incr-wasted-st-def*)
have ‹*mark-garbage-pre* (*get-clauses-wl-heur x2b*, *get-tvdom x2b* ! *x1b*)›
‹*x1b* < *length* (*get-tvdom x2b*)›
using *that*
unfolding *prod.simps mark-garbage-pre-def*
arena-is-valid-clause-vdom-def arena-is-valid-clause-idx-def **apply** −
by (*rule exI*[*of* - ‹*get-clauses-wl x1a*›], *rule exI*[*of* - ‹*set* (*get-vdom x2b*)›])
(*auto simp*: *twl-st-heur-restart twl-st-heur-restart-ana-def dest*: *twl-st-heur-restart-valid-arena*)
moreover have *0*: ‹*Suc 0* ≤ *clss-size-lcount* (*get-learned-count x2b*)›
‹¬ *irred* (*get-clauses-wl x1a*) (*x2a* ! *x1*)›
using *get-learned-count-ge*[*OF that*(*1*−*29*,*32*)] *only-irred*[*OF that*(*1*−*29*,*32*)] **by** *auto*
moreover have ‹(*mark-garbage-heur3* (*get-tvdom x2b* ! *x1*) *x1*
(*incr-wasted-st* (*word-of-nat wasted*) *x2b*),
mark-garbage-wl (*get-tvdom x2b* ! *x1*) *x1a*)
∈ *twl-st-heur-restart-ana r*›
by (*use that 0* **in**
‹*auto intro*!: *incr-wasted-st mark-garbage-heur-wl-ana simp*: *twl-st-heur-restart*
learned-clss-count-mark-garbage-heur3
dest: *twl-st-heur-restart-valid-arena twl-st-heur-restart-anaD*›)
moreover have ‹*learned-clss-count*
(*mark-garbage-heur3* (*get-tvdom x2b* ! *x1*) *x1*
(*incr-wasted-st* (*word-of-nat wasted*) *x2b*))
≤ *u*›
by (*use that 0* **in**
‹*auto intro*!: *incr-wasted-st mark-garbage-heur-wl-ana simp*: *twl-st-heur-restart*
learned-clss-count-mark-garbage-heur3
dest: *twl-st-heur-restart-valid-arena twl-st-heur-restart-anaD*›)
moreover have ‹*xb* ∈ *set* (*get-tvdom S*)›
if ‹*xb* ∈ *set* (*butlast* ((*get-tvdom x2b*)[*x1* := *last* (*get-tvdom x2b*)]))› **for** *xb*
proof −
have ‹*xb* ∈ *set* (*get-tvdom x2b*)›
using *that H* **by** (*auto dest*!: *in-set-butlastD*)
(*metis Misc.last-in-set in-set-upd-eq len-greater-imp-nonempty*)
then show *?thesis*
using *H* **by** *auto*
qed
moreover have *K*: ‹¬*irred* (*get-clauses-wl* (*mark-garbage-wl* (*get-tvdom x2b* ! *x1*) *x1a*)) *C*› ‹*C* ≠ *get-tvdom x2b* ! *x1*›
if ‹*C* ∈ *set* (*butlast* ((*get-tvdom x2b*)[*x1* := *last* (*get-tvdom x2b*)]))› **for** *C*
proof −
have *a*: ‹*distinct* (*get-tvdom x2b*)› ‹*x1* < *length* (*get-tvdom x2b*)›
using *H*(*9*−*15*) **by** (*auto simp*: *twl-st-heur-restart-ana-def*
twl-st-heur-restart-def aivdom-inv-dec-alt-def)
then have *1*: ‹*get-tvdom x2b* = *take x1* (*get-tvdom x2b*) @ *get-tvdom x2b* ! *x1* # *drop* (*Suc x1*) (*get-tvdom x2b*)›
by (*subst append-take-drop-id*[*of x1*, *symmetric*], *subst Cons-nth-drop-Suc*[*symmetric*])
auto
have ‹*set* (*delete-index-and-swap* (*get-tvdom x2b*) *x1*) =
set (*get-tvdom x2b*) − {*get-tvdom x2b*!*x1*}›
using *a*
apply (*subst* (*asm*) (*2*)*1*, *subst* (*asm*) (*1*)*1*)
apply (*subst* (*2*)*1*, *subst* (*1*)*1*)
apply (*cases* ‹*drop* (*Suc x1*) (*get-tvdom x2b*)› *rule*: *rev-cases*)

```
      by (auto simp: nth-append list-update-append1 list-update-append2 butlast-append
        dest: in-set-butlastD)
    then show [simp]: ‹C ≠ get-tvdom x2b ! x1›
      using that by auto
    show ‹¬irred (get-clauses-wl (mark-garbage-wl (get-tvdom x2b ! x1) x1a)) C›
      using that H
      apply (cases x1a)
      apply (auto simp: mark-garbage-wl-def)
      by (metis Misc.last-in-set in-set-butlastD in-set-upd-cases len-greater-imp-nonempty)
  qed
  moreover have ‹length (get-clauses-wl (mark-garbage-wl (get-tvdom x2b ! x1) x1a) ∝ C) ≠ 2›
    if ‹C ∈ set (butlast ((get-tvdom x2b)[x1 := last (get-tvdom x2b)]))› for C
  proof −
    have ‹C ∈ set (get-tvdom x2b)› ‹C ≠ get-tvdom x2b ! x1›
      using K(2)[OF that] that
      apply (auto simp: mark-garbage-wl-def)
      using in-set-delete-index-and-swapD by fastforce
    then show ?thesis
      using H
      by (cases x1a)
       (auto simp: mark-garbage-wl-def)
  qed
  ultimately show ?thesis
    using that supply [[goals-limit=1]]
    by (auto intro!: )
qed
have mop-arena-length-st: ‹mop-arena-length-st x2b (get-tvdom x2b ! x1b)
 ≤ SPEC
 (λc. (c, length (get-clauses-wl x1a ∝ (x2a ! x1)))) ∈ nat-rel)›
 if H:
   ‹(x, y) ∈ twl-st-heur-restart-ana′ r u›
   ‹mark-to-delete-clauses-wl-pre y›
   ‹mark-to-delete-clauses-wl-D-heur-pre x›
   ‹(S, Sa) ∈ ?sort x y›
   ‹(uu, xs) ∈ ?indices S Sa›
   ‹(l, la) ∈ nat-rel›
   ‹la ∈ {-. True}›
   ‹length (get-tvdom S) ≤ length (get-clauses-wl-heur x)›
   ‹(xa, x′) ∈ ?R S›
   ‹case xa of (i, S) ⇒ i < length (get-tvdom S)›
   ‹case x′ of (i, T, xs) ⇒ i < length xs›
   ‹mark-to-delete-clauses-wl-inv Sa xs x′›
   ‹x2 = (x1a, x2a)›
   ‹x′ = (x1, x2)›
   ‹xa = (x1b, x2b)›
   ‹x1b < length (get-tvdom x2b)›
   ‹access-tvdom-at-pre x2b x1b›
   ‹clause-not-marked-to-delete-heur-pre (x2b, get-tvdom x2b ! x1b)›
   ‹(b, ba) ∈ {(b, b′). (b, b′) ∈ bool-rel ∧ b = (x2a ! x1 ∈# dom-m (get-clauses-wl x1a))}›
   ‹¬ ¬ b›
   ‹¬ ¬ ba›
   ‹0 < length (get-clauses-wl x1a ∝ (x2a ! x1))›
   ‹get-clauses-wl x1a ∝ (x2a ! x1) ! 0 ∈# \mathcal{L}_{all} (all-init-atms-st x1a)›
   ‹access-lit-in-clauses-heur-pre ((x2b, get-tvdom x2b ! x1b), 0)›
   ‹length (get-clauses-wl-heur x2b) ≤ length (get-clauses-wl-heur x)›
   ‹length (get-tvdom x2b) ≤ length (get-clauses-wl-heur x2b)›
```

‹(L, K) ∈ {(L, L'). (L, L') ∈ nat-lit-lit-rel ∧ L' = get-clauses-wl x1a ∝ (x2a ! x1) ! 0}›
‹(D, bb) ∈ reason-rel K x1a›
‹arena-is-valid-clause-idx (get-clauses-wl-heur x2b) (get-tvdom x2b ! x1b)›
‹(D ≠ Some (get-tvdom x2b ! x1b) ∧ arena-length (get-clauses-wl-heur x2b) (get-tvdom x2b ! x1b) ≠ 2, can-del)
∈ bool-rel›
‹x1 < length x2a›
‹D ≠ Some (get-tvdom x2b ! x1b) ∧ arena-length (get-clauses-wl-heur x2b) (get-tvdom x2b ! x1b) ≠ 2›
can-del
for x y S Sa xs l la xa x' x1 x2 x1a x2a x1b x2b b ba can-del D L K bb uu
unfolding mop-arena-length-st-def
apply (rule mop-arena-length[THEN fref-to-Down-curry, THEN order-trans,
of ‹get-clauses-wl x1a› ‹get-tvdom x2b ! x1b› - - ‹set (get-vdom x2b)›])
using that **by** (auto simp: twl-st-heur-restart-ana-def twl-st-heur-restart-def)
show ?thesis
supply sort-vdom-heur-def[simp] twl-st-heur-restart-anaD[dest] [[goals-limit=1]]
unfolding mark-to-delete-clauses-wl-D-heur-alt-def mark-to-delete-clauses-wl-D-alt-def
access-lit-in-clauses-heur-def
apply (intro frefI nres-relI)
apply (refine-vcg sort-vdom-heur-reorder-vdom-wl[THEN fref-to-Down] incr-restart-stat find-largest-lbd-and-size)
subgoal
unfolding mark-to-delete-clauses-wl-D-heur-pre-def **by** fast
apply assumption
subgoal by auto
subgoal for x y S T **unfolding** number-clss-to-keep-def **by** (cases S) (auto)
apply (solves auto)
subgoal by (auto simp: twl-st-heur-restart-ana-def twl-st-heur-restart-def aivdom-inv-dec-alt-def
dest!: valid-arena-vdom-subset size-mset-mono)
apply (rule init; solves auto)
subgoal by auto
subgoal by auto
subgoal by (auto simp: access-tvdom-at-pre-def)
subgoal for x y S Sa u xs l la - - x1 x2 xb x' x1c x2c x1a x2a x1b x2b
unfolding clause-not-marked-to-delete-heur-pre-def arena-is-valid-clause-vdom-def
prod.simps
by (rule exI[of - ‹get-clauses-wl (fst x2c)›], rule exI[of - ‹set (get-vdom x2b)›])
(auto simp: twl-st-heur-restart
intro!: twl-st-heur-restart-valid-arena[simplified]
dest: twl-st-heur-restart-get-tvdom-nth-get-vdom[simplified])
apply (rule mop-clause-not-marked-to-delete-heur; assumption)
subgoal for x y S Sa uu xs l la xa x' x1 x2 x1a x2a x1b x2b
by (auto simp: twl-st-heur-restart)
subgoal
by (rule already-deleted)
subgoal for x y S Sa u xs l la - - x1 x2 xb x' x1c x2c x1a x2a x1b x2b
unfolding access-lit-in-clauses-heur-pre-def prod.simps arena-lit-pre-def
arena-is-valid-clause-idx-and-access-def
by (rule bex-leI[of - ‹get-tvdom x2b ! x1b›], simp add: aivdom-inv-dec-alt-def,
rule exI[of - ‹get-clauses-wl (fst x2c)›])
(auto simp: twl-st-heur-restart-ana-def twl-st-heur-restart-def aivdom-inv-dec-alt-def)
subgoal by (auto simp: twl-st-heur-restart-ana-def twl-st-heur-restart-def dest!: valid-arena-vdom-subset size-mset-mono)
subgoal premises p **using** p(7−)
by (auto simp: twl-st-heur-restart-ana-def twl-st-heur-restart-def aivdom-inv-dec-alt-def
dest!: valid-arena-vdom-subset size-mset-mono)

apply (*rule mop-access-lit-in-clauses-heur*; *assumption*)
apply (*rule get-the-propagation-reason*; *assumption*)
subgoal for *x y S Sa u xs l la - - x1 x2 xb x' x1c x2c x1a x2a x1b x2b*
unfolding *prod.simps*
arena-is-valid-clause-vdom-def arena-is-valid-clause-idx-def
by (*rule exI*[*of - ‹get-clauses-wl (fst x2c)›*], *rule exI*[*of - ‹set (get-vdom x2b)›*])
(*auto simp*: *twl-st-heur-restart dest*:*twl-st-heur-restart-get-avdom-nth-get-vdom*
intro!: *twl-st-heur-restart-valid-arena*[*simplified*])
subgoal
unfolding *marked-as-used-pre-def*
by (*auto simp*: *twl-st-heur-restart reason-rel-def*)
subgoal for *x y S Sa uu xs l la xa x' x1 x2 x1a x2a x1b x2b b ba L K D bb*
by (*rule only-irred*)
subgoal
by (*auto dest*!: *twl-st-heur-restart-anaD twl-st-heur-restart-valid-arena simp*: *arena-lifting*)
subgoal by *fast*
apply (*rule mop-arena-length-st*; *assumption*)
apply (*rule log-del-clause-heur-log-clause*[**where** *r*=*r* **and** *u*=*u*])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal for *x y S Sa u xs l la - - x1 x2 xb x' x1c x2c x1a x2a x1b x2b*
unfolding *prod.simps mark-garbage-pre-def*
arena-is-valid-clause-vdom-def arena-is-valid-clause-idx-def **apply** −
by (*rule exI*[*of - ‹get-clauses-wl (fst x2c)›*], *rule exI*[*of - ‹set (get-vdom x2b)›*])
(*auto simp*: *twl-st-heur-restart twl-st-heur-restart-ana-def dest*: *twl-st-heur-restart-valid-arena*)
subgoal premises *that*
using *get-learned-count-ge*[*OF that*(*2*−*8*,*12*−*15*,*17*−*33*)] *that*(*32*−)
using *only-irred*[*OF that*(*2*−*8*,*12*−*15*,*17*−*33*)]
by *auto*
subgoal for *x y S Sa - xs l la xa x' x1 x2 x1a x2a x1b x2b*
by (*rule length-filter-le*)
subgoal for *x y*
unfolding *valid-sort-clause-score-pre-def arena-is-valid-clause-vdom-def*
get-clause-LBD-pre-def arena-is-valid-clause-idx-def arena-act-pre-def
by (*force simp*: *valid-sort-clause-score-pre-def twl-st-heur-restart-ana-def arena-dom-status-iff*
arena-act-pre-def get-clause-LBD-pre-def arena-is-valid-clause-idx-def twl-st-heur-restart-def
intro!: *exI*[*of - ‹get-clauses-wl T›*] *dest*!: *set-mset-mono mset-subset-eqD*)
subgoal for *x y S Sa uu xs l la xa x' x1 x2 x1a x2a x1c x2c b ba L x2b*
using *size-mset-mono*[*of ‹mset (get-tvdom x2b)› ‹mset (get-vdom x2b)›*]
by (*clarsimp simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def aivdom-inv-dec-alt-def*
dest!: *valid-arena-vdom-subset*)
subgoal
by *auto*
done
qed

lemma *cdcl-twl-mark-clauses-to-delete-alt-def*:
‹*cdcl-twl-mark-clauses-to-delete S* = *do* {
- ← *ASSERT* (*mark-to-delete-clauses-wl-D-heur-pre S*);
T ← *mark-to-delete-clauses-wl-D-heur S*;
RETURN (*schedule-next-reduction-st* (*clss-size-resetUS0-st T*))
}›
unfolding *cdcl-twl-mark-clauses-to-delete-def IsaSAT-Profile.start-def IsaSAT-Profile.stop-def*
by *auto*

lemma *learned-clss-count-clss-size-resetUS0-st-le*:
  ‹*learned-clss-count* (*clss-size-resetUS0-st T*) ≤ *learned-clss-count T*› **and**
  clss-size-resetUS0-st-simp[*simp*]:
  ‹*get-clauses-wl-heur* (*clss-size-resetUS0-st T*) = *get-clauses-wl-heur T*›
  by (*cases T*;*clarsimp simp*: *clss-size-resetUS0-st-def learned-clss-count-def*
    clss-size-lcountUS-def clss-size-lcountU0-def clss-size-lcountUE-def
    clss-size-resetUS-def clss-size-resetU0-def clss-size-resetUE-def clss-size-lcountUEk-def
    clss-size-lcount-def
    split: *prod.splits*)+

lemma *learned-clss-count-schedule-next-reduction-st-le*:
  ‹*learned-clss-count* (*schedule-next-reduction-st T*) = *learned-clss-count T*› **and**
  schedule-next-reduction-st-simp[*simp*]:
  ‹*get-clauses-wl-heur* (*schedule-next-reduction-st T*) = *get-clauses-wl-heur T*›
  by (*solves* ‹*cases T*;*clarsimp simp*: *schedule-next-reduction-st-def learned-clss-count-def Let-def*
     schedule-next-reduce-st-def›)+

lemma *schedule-next-reduction-sttwl-st-heur*:
  ‹(*S*,*T*)∈*twl-st-heur* ⟹ (*schedule-next-reduction-st S*, *T*) ∈ *twl-st-heur*›
  by (*auto simp*: *twl-st-heur-def schedule-next-reduction-st-def Let-def*
    schedule-next-reduce-st-def)

lemma *cdcl-twl-mark-clauses-to-delete-cdcl-twl-full-restart-wl-prog-D*:
  ‹(*cdcl-twl-mark-clauses-to-delete*, *cdcl-twl-full-restart-wl-prog*) ∈
    twl-st-heur''''u r u →*f* ⟨*twl-st-heur''''u r u*⟩*nres-rel*›
  unfolding *cdcl-twl-mark-clauses-to-delete-alt-def cdcl-twl-full-restart-wl-prog-def*
  apply (*intro frefI nres-relI*)
  apply (*refine-vcg*
    mark-to-delete-clauses-wl-D-heur-mark-to-delete-clauses-wl-D[*THEN fref-to-Down*])
  subgoal
    by (*rule mark-to-delete-clauses-wl-D-heur-pre-alt-def*) *fast*
  apply (*rule twl-st-heur-restartD2*)
  subgoal
    by (*rule mark-to-delete-clauses-wl-D-heur-pre-twl-st-heur*) *auto*
  subgoal for *x y T Ta*
    using *learned-clss-count-clss-size-resetUS0-st-le*[*of T*]
      learned-clss-count-schedule-next-reduction-st-le[*of* ‹*clss-size-resetUS0-st T*›]
    by (*auto simp*: *mark-to-delete-clauses-wl-post-twl-st-heur twl-st-heur-restart-anaD*
        schedule-next-reduction-sttwl-st-heur)
     (*auto simp*: *twl-st-heur-restart-ana-def*)
  done

lemma *cdcl-twl-restart-wl-heur-cdcl-twl-restart-wl-D-prog*:
  ‹(*cdcl-twl-restart-wl-heur*, *cdcl-twl-restart-wl-prog*) ∈
    twl-st-heur''''u r u →*f* ⟨*twl-st-heur''''u r u*⟩*nres-rel*›
  unfolding *cdcl-twl-restart-wl-prog-def cdcl-twl-restart-wl-heur-def*
  by (*intro frefI nres-relI*)
    (*refine-rcg lhs-step-If*
    cdcl-twl-local-restart-wl-D-heur-cdcl-twl-local-restart-wl-D-spec[*THEN fref-to-Down*]
    cdcl-twl-mark-clauses-to-delete-cdcl-twl-full-restart-wl-prog-D[*THEN fref-to-Down*])

lemma *mark-to-delete-clauses-wl-D-heur-mark-to-delete-clauses-GC-wl-D*:
  ‹(*mark-to-delete-clauses-GC-wl-D-heur*, *mark-to-delete-clauses-GC-wl*) ∈
    twl-st-heur-restart-ana' r u →*f*
     ⟨*twl-st-heur-restart-ana' r u*⟩*nres-rel*›
proof −

```
  have H: ‹mark-to-delete-clauses-GC-wl-pre S ∧ mark-to-delete-clauses-wl-pre S ⟷
    mark-to-delete-clauses-GC-wl-pre S› for S
   unfolding mark-to-delete-clauses-GC-wl-pre-def mark-to-delete-clauses-wl-pre-def
   mark-to-delete-clauses-l-GC-pre-def mark-to-delete-clauses-l-pre-def
   by blast
  have mark-to-delete-clauses-GC-wl-D-alt-def:
    ‹mark-to-delete-clauses-GC-wl  = (λS0. do {
      ASSERT(mark-to-delete-clauses-GC-wl-pre S0);
      S ← reorder-vdom-wl S0;
      xs ← collect-valid-indices-wl S;
      l ← SPEC (λ-::nat. True);
      - ← SPEC (λ-::nat×nat. True);
      (-, S, -) ← WHILE_T^mark-to-delete-clauses-GC-wl-inv S xs
        (λ(i, T, xs). i < length xs)
        (λ(i, T, xs). do {
          b ← RETURN (xs!i ∈# dom-m (get-clauses-wl T));
          if ¬b then RETURN (i, T, delete-index-and-swap xs i)
          else do {
            ASSERT(0 < length (get-clauses-wl T∝(xs!i)));
     ASSERT (get-clauses-wl T ∝ (xs ! i) ! 0 ∈# L_all (all-init-atms-st T));
            can-del ← SPEC(λb. b ⟶
              (¬irred (get-clauses-wl T) (xs!i) ∧ length (get-clauses-wl T ∝ (xs!i)) ≠ 2));
            ASSERT(i < length xs);
            if can-del
            then
              RETURN (i, mark-garbage-wl (xs!i) T, delete-index-and-swap xs i)
            else
              RETURN (i+1, T, xs)
          }
        })
        (l, S, xs);
      remove-all-learned-subsumed-clauses-wl S
    })›
   unfolding mark-to-delete-clauses-GC-wl-def reorder-vdom-wl-def bind-to-let-conv Let-def
     nres-monad3 summarize-ASSERT-conv H
   by (auto intro!: ext bind-cong[OF refl])


  have mono: ‹g = g' ⟹ do {f; g} = do {f; g'}›
     ‹(⋀x. h x = h' x) ⟹ do {x ← f; h x} = do {x ← f; h' x}› for f f' :: ‹- nres› and g g' and h h'
    by auto
  have mark-to-delete-clauses-wl-pre-same-atms: ‹set-mset (all-atms-st T) = set-mset (all-init-atms-st
T)›
    if ‹mark-to-delete-clauses-wl-pre T› for T
    using that unfolding mark-to-delete-clauses-wl-pre-def mark-to-delete-clauses-l-pre-def apply -
    apply normalize-goal+
    by (rule literals-are-L_in'-literals-are-L_in-iff(3)[symmetric]) assumption+

  have mark-to-delete-clauses-wl-D-heur-pre-cong:
    ‹aivdom-inv-dec vdom' (dom-m (get-clauses-wl S')) ⟹
    mset (get-vdom-aivdom (get-aivdom T)) = mset (get-vdom-aivdom vdom') ⟹
    (T, S') ∈ twl-st-heur-restart ⟹
    mark-to-delete-clauses-GC-wl-pre S' ⟹
    mark-to-delete-clauses-GC-wl-D-heur-pre T ⟹
    valid-arena N'' (get-clauses-wl S') (set (get-vdom-aivdom (get-aivdom T))) ⟹
    mark-to-delete-clauses-GC-wl-D-heur-pre (set-clauses-wl-heur N'' (set-aivdom-wl-heur vdom' T))›
```

for M' N' D' j W' vm $clvls$ $cach$ lbd $outl$ $stats$ $fast\text{-}ema$ $slow\text{-}ema$ $avdom$ $avdom'$
$ccount$ $lcount$ $heur$ $old\text{-}arena$ $ivdom$ $opts$ S' $vdom'$ N'' T
using *mset-eq-setD*[*of* ‹*get-vdom-aivdom* (*get-aivdom T*)› ‹*get-vdom-aivdom vdom'*›, *symmetric*]
apply −
unfolding *mark-to-delete-clauses-GC-wl-D-heur-pre-def*
by (*rule-tac x*=S' **in** *exI*)
(*clarsimp simp*: *twl-st-heur-restart-def dest*: *mset-eq-setD intro*:)

have *mark-to-delete-clauses-wl-pre-same-atms*: ‹*set-mset* (*all-atms-st T*) = *set-mset* (*all-init-atms-st T*)›
if ‹*mark-to-delete-clauses-GC-wl-pre T*› **for** T
using *that* **unfolding** *mark-to-delete-clauses-GC-wl-pre-def mark-to-delete-clauses-l-pre-def*
mark-to-delete-clauses-l-GC-pre-def **apply** −
apply *normalize-goal+*
by (*rule literals-are-*$\mathcal{L}_{in}$*'-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*symmetric*]) *assumption+*

have [*refine0*]:
‹*sort-vdom-heur S* ≤ ⇓ {(*U*, *V*). (*U*, *V*) ∈ *twl-st-heur-restart-ana' r u* ∧ *V* = *T* ∧
(*mark-to-delete-clauses-GC-wl-pre T* ⟶ *mark-to-delete-clauses-GC-wl-pre V*) ∧
(*mark-to-delete-clauses-GC-wl-D-heur-pre S* ⟶ *mark-to-delete-clauses-GC-wl-D-heur-pre U*) ∧
(∀ *C*∈*set* (*get-tvdom U*). ¬*irred* (*get-clauses-wl V*) *C*)}
(*reorder-vdom-wl T*)› (**is** ‹- ≤ ⇓*?sort* -›)
if ‹(*S*, *T*) ∈ *twl-st-heur-restart-ana' r u*› **and** ‹*mark-to-delete-clauses-GC-wl-pre T*› **for** S T
supply [*simp del*] = *EQ-def*
using *that* **unfolding** *reorder-vdom-wl-def sort-vdom-heur-def Let-def*
apply (*refine-rcg ASSERT-leI*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def aivdom-inv-dec-alt-def*
dest!: *valid-arena-vdom-subset size-mset-mono*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def aivdom-inv-dec-alt-def*
dest!: *valid-arena-vdom-subset size-mset-mono*)
apply (*rule specify-left-RES*)
apply (*rule-tac* N = ‹*get-clauses-wl T*› **and** M' = ‹*get-trail-wl T*› **and**
$\mathcal{A}$ = ‹*all-init-atms-st T*› **and**
NUE = ‹*get-unit-clauses-wl T* + *get-subsumed-clauses-wl T* + *get-clauses0-wl T*› **in**
isa-isa-gather-candidates-for-reduction-remove-deleted-clauses-from-avdom2[*unfolded conc-fun-RES*])
subgoal
by (*case-tac T*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case*)
subgoal
by (*case-tac T*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case*)
subgoal
by (*case-tac T*; *auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def mem-Collect-eq prod.case*
all-init-atms-st-def)
subgoal
unfolding *all-atms-st-def*[*symmetric*]
by (*rule mark-to-delete-clauses-wl-pre-same-atms*)
apply (*subst case-prod-beta*)
apply (*intro ASSERT-leI*)
subgoal for *arena-aivdom*
unfolding *valid-sort-clause-score-pre-def arena-is-valid-clause-vdom-def*
get-clause-LBD-pre-def arena-is-valid-clause-idx-def arena-act-pre-def
by (*auto simp*: *valid-sort-clause-score-pre-def twl-st-heur-restart-ana-def arena-dom-status-iff*(*1*)[*symmetric*]
arena-act-pre-def get-clause-LBD-pre-def arena-is-valid-clause-idx-def twl-st-heur-restart-def aivdom-inv-dec-alt-def
intro!: *exI*[*of* - ‹*get-clauses-wl T*›] *exI*[*of* - ‹*set* (*get-vdom S*)›]
dest: *set-mset-mono mset-subset-eqD*)
subgoal by (*auto simp*: *EQ-def*)

subgoal
by (*cases T*)
(*clarsimp simp add*: *twl-st-heur-restart-ana-def valid-arena-vdom-subset twl-st-heur-restart-def aivdom-inv-dec-alt-def case-prod-beta split*:
dest!: *size-mset-mono valid-arena-vdom-subset*)
subgoal for *arena-aivdom*
apply (*rewrite at* ‹- ≤ ⌑› *Down-id-eq*[*symmetric*])
apply (*rule bind-refine-spec*)
prefer *2*
apply (*rule sort-clauses-by-score-reorder*[*of* - ‹*get-clauses-wl T*›])
subgoal
by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*: *mset-eq-setD*)
subgoal
by (*cases* ‹*arena-aivdom*›; *cases* ‹*get-content* (*snd arena-aivdom*)›)
(*simp add*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*)
subgoal
apply *auto*
apply (*auto simp*: *learned-clss-count-def*
intro: *mark-to-delete-clauses-wl-D-heur-pre-cong*
intro: *aivdom-inv-cong2*
dest: *mset-eq-setD*)
apply (*auto 4 3 simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*
intro: *aivdom-inv-cong2 dest*: *mset-eq-setD*; *fail*)[]
apply (*rule mark-to-delete-clauses-wl-D-heur-pre-cong*)
apply *assumption*+
apply (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*)
done
done
done

have [*refine0*]: ‹*RETURN* (*get-tvdom x*) ≤ ⇓ {(*xs*, *xs'*). *xs* = *xs'* ∧ *xs* = *get-tvdom x* ∧
(∀ *C*∈*set* (*get-tvdom x*). ¬*irred* (*get-clauses-wl y*) *C*)} (*collect-valid-indices-wl y*)›
(**is** ‹- ≤ ⇓ *?indices* -›)
if
‹(*x*, *y*) ∈ *?sort S T*› **and**
‹*mark-to-delete-clauses-GC-wl-D-heur-pre x*›
for *x y S T*
proof −
show *?thesis* **using** *that* **by** (*auto simp*: *collect-valid-indices-wl-def simp*: *RETURN-RES-refine-iff*)
qed

have *init*:
‹(*u'*, *xs*) ∈ *?indices S Sa* ⟹
(*l*, *la*) ∈ *nat-rel* ⟹
(*S*, *Sa*) ∈ *twl-st-heur-restart-ana' r u* ⟹
((*l*, *S*), *la*, *Sa*, *xs*) ∈ *nat-rel* ×$_f$
{(*Sa'*, (*T*, *xs*)). (*Sa'*, *T*) ∈ *twl-st-heur-restart-ana' r u* ∧ *xs* = *get-tvdom Sa'* ∧
set (*get-tvdom Sa'*) ⊆ *set* (*get-tvdom S*) ∧ (∀ *C*∈*set* (*get-tvdom Sa'*). ¬*irred* (*get-clauses-wl T*) *C*)}›
(**is** ‹- ⟹ - ⟹ - ⟹ - ∈ *?R*›)
for *x y S Sa xs l la u'*
by *auto*

define *reason-rel* **where**
‹*reason-rel K x1a* ≡ {(*C*, - :: *unit*).
(*C* ≠ *None*) = (*Propagated K* (*the C*) ∈ *set* (*get-trail-wl x1a*)) ∧
(*C* = *None*) = (*Decided K* ∈ *set* (*get-trail-wl x1a*) ∨

K ∉ *lits-of-l* (*get-trail-wl x1a*)) ∧
(∀ *C1*. (*Propagated K C1* ∈ *set* (*get-trail-wl x1a*) ⟶ *C1* = *the C*))}› **for** *K* :: ‹*nat literal*› **and** *x1a*

have *already-deleted*:
‹((*x1b*, *delete-index-vdom-heur x1b x2b*), *x1*, *x1a*, *delete-index-and-swap x2a x1*) ∈ *?R S*›
if
‹(*x*, *y*) ∈ *twl-st-heur-restart-ana′ r u*› **and**
‹*mark-to-delete-clauses-GC-wl-D-heur-pre x*› **and**
‹(*S*, *Sa*) ∈ *?sort x y*› **and**
‹(*l*, *la*) ∈ *nat-rel*› **and**
‹*la* ∈ {-. *True*}› **and**
xx: ‹(*xa*, *x′*) ∈ *?R S*› **and**
nempty: ‹*case xa of* (*i*, *S*) ⇒ *i* < *length* (*get-tvdom S*)› **and**
‹*case x′ of* (*i*, *T*, *xs*) ⇒ *i* < *length xs*› **and**
st:
‹*x2* = (*x1a*, *x2a*)›
‹*x′* = (*x1*, *x2*)›
‹*xa* = (*x1b*, *x2b*)› **and**
le: ‹*x1b* < *length* (*get-tvdom x2b*)› **and**
‹*access-tvdom-at-pre x2b x1b*› **and**
ba: ‹(*b*, *ba*) ∈ {(*b*, *b′*). (*b*, *b′*) ∈ *bool-rel* ∧ *b* = (*x2a* ! *x1* ∈# *dom-m* (*get-clauses-wl x1a*))}›
‹¬*ba*›
for *x y S xs l la xa x′ xz x1 x2 x1a x2a x2b x2c x2d ys x1b Sa ba b*
proof −
show *?thesis*
using *xx nempty le ba* **unfolding** *st*
by (*cases* ‹*get-tvdom x2b*› *rule*: *rev-cases*)
(*auto 4 3 simp*: *twl-st-heur-restart-ana-def delete-index-vdom-heur-def*
twl-st-heur-restart-def mark-garbage-heur-def mark-garbage-wl-def
learned-clss-l-l-fmdrop size-remove1-mset-If learned-clss-count-def
aivdom-inv-removed-inactive
intro: *valid-arena-extra-information-mark-to-delete′*
intro!: *aivdom-inv-dec-removed-inactive-tvdom*
dest!: *in-set-butlastD in-vdom-m-fmdropD*
elim!: *in-set-upd-cases*)
qed

have *mop-clause-not-marked-to-delete-heur*:
‹*mop-clause-not-marked-to-delete-heur x2b* (*get-tvdom x2b* ! *x1b*)
≤ *SPEC*
(λ*c*. (*c*, *x2a* ! *x1* ∈# *dom-m* (*get-clauses-wl x1a*))
∈ {(*b*, *b′*). (*b*,*b′*) ∈ *bool-rel* ∧ (*b* ⟷ *x2a* ! *x1* ∈# *dom-m* (*get-clauses-wl x1a*))})›
if
‹(*xa*, *x′*) ∈ *?R S*› **and**
‹*case xa of* (*i*, *S*) ⇒ *i* < *length* (*get-tvdom S*)› **and**
‹*case x′ of* (*i*, *T*, *xs*) ⇒ *i* < *length xs*› **and**
‹*mark-to-delete-clauses-GC-wl-inv Sa xs x′*› **and**
‹*x2* = (*x1a*, *x2a*)› **and**
‹*x′* = (*x1*, *x2*)› **and**
‹*xa* = (*x1b*, *x2b*)› **and**
‹*clause-not-marked-to-delete-heur-pre* (*x2b*, *get-tvdom x2b* ! *x1b*)›
for *x y S Sa uu xs l la xa x′ x1 x2 x1a x2a x1b x2b*
unfolding *mop-clause-not-marked-to-delete-heur-def*
apply *refine-vcg*

subgoal
using *that* **by** *blast*
subgoal
using *that* **by** (*auto simp*: *twl-st-heur-restart arena-lifting dest*: *twl-st-heur-restart*(*2*) *dest*!: *twl-st-heur-restart-anaD*)
done
have *incr-reduction-stat*: ‹*incr-restart-stat* (*set-stats-size-limit-st lbd sze T*) ≤ ⇓ (*twl-st-heur-restart-ana′ r u*) (*remove-all-learned-subsumed-clauses-wl S*)›
if ‹(*T*, *S*) ∈ *twl-st-heur-restart-ana′ r u*› **for** *S T i u lbd sze*
using *that*
by (*cases S*; *cases T*)
(*auto simp*: *conc-fun-RES incr-restart-stat-def learned-clss-count-def set-stats-size-limit-st-def twl-st-heur-restart-ana-def twl-st-heur-restart-def remove-all-learned-subsumed-clauses-wl-def clss-size-corr-def clss-size-lcountUE-def clss-size-lcountUS-def clss-size-def clss-size-resetUS-def clss-size-lcount-def clss-size-lcountU0-def incr-reduction-stat-def RES-RETURN-RES*)

have *only-irred*: ‹¬ *irred* (*get-clauses-wl x1a*) (*x2a* ! *x1*)› (**is** *?A*) **and**
get-learned-count-ge: ‹*Suc 0* ≤ *clss-size-lcount* (*get-learned-count x2b*)› (**is** *?B*)
if
‹(*x*, *y*) ∈ *twl-st-heur-restart-ana′ r u*› **and**
‹*mark-to-delete-clauses-GC-wl-pre y*› **and**
‹*mark-to-delete-clauses-GC-wl-D-heur-pre x*› **and**
‹(*S*, *Sa*) ∈ *?sort x y*› **and**
indices: ‹(*uu*, *xs*) ∈ *?indices S Sa*› **and**
‹(*l*, *la*) ∈ *nat-rel*› **and**
‹*la* ∈ {-. *True*}› **and**
‹*length* (*get-tvdom S*) ≤ *length* (*get-clauses-wl-heur x*)› **and**
xx: ‹(*xa*, *x′*) ∈ *?R S*› **and**
‹*case xa of* (*i*, *S*) ⇒ *i* < *length* (*get-tvdom S*)› **and**
‹*case x′ of* (*i*, *T*, *xs*) ⇒ *i* < *length xs*› **and**
‹*mark-to-delete-clauses-GC-wl-inv Sa xs x′*› **and**
st: ‹*x2* = (*x1a*, *x2a*)› ‹*x′* = (*x1*, *x2*)› ‹*xa* = (*x1b*, *x2b*)› **and**
‹*x1b* < *length* (*get-tvdom x2b*)› **and**
‹*access-tvdom-at-pre x2b x1b*› **and**
‹*clause-not-marked-to-delete-heur-pre* (*x2b*, *get-tvdom x2b* ! *x1b*)› **and**
dom: ‹(*b*, *ba*) ∈ {(*b*, *b′*). (*b*, *b′*) ∈ *bool-rel* ∧ *b* = (*x2a* ! *x1* ∈# *dom-m* (*get-clauses-wl x1a*))}›
‹¬ ¬ *b*›
‹¬ ¬ *ba*› **and**
‹*0* < *length* (*get-clauses-wl x1a* ∝ (*x2a* ! *x1*))› **and**
‹*get-clauses-wl x1a* ∝ (*x2a* ! *x1*) ! *0* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1a*)› **and**
‹*access-lit-in-clauses-heur-pre* ((*x2b*, *get-tvdom x2b* ! *x1b*), *0*)› **and**
‹*length* (*get-clauses-wl-heur x2b*) ≤ *length* (*get-clauses-wl-heur x*)› **and**
‹*length* (*get-tvdom x2b*) ≤ *length* (*get-clauses-wl-heur x2b*)› **and**
‹*arena-is-valid-clause-idx* (*get-clauses-wl-heur x2b*) (*get-tvdom x2b* ! *x1b*)›
for *x y S Sa uu xs l la xa x′ x1 x2 x1a x2a x1b x2b b ba L K D bb*
proof −
have ‹*get-tvdom x2b* ! *x1* ∈ *set* (*get-tvdom x2b*)› **and**
x: ‹(*x2b*, *x1a*) ∈ *twl-st-heur-restart-ana r*›
using *that* **by** (*auto dest*: *simp*: *arena-lifting twl-st-heur-restart*)
then show *?A*
using *indices xx*
by (*auto dest*: *twl-st-heur-restart-anaD twl-st-heur-restart-valid-arena simp*: *arena-lifting twl-st-heur-restart st*)
then show *?B*

using *dom x* **by** (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def ran-m-def*
dest!: *multi-member-split*
dest!: *clss-size-corr-restart-rew*)
qed
have *length-filter-le*: ‹⇓ *nat-rel* ((*RETURN* ∘ (λc. *length* (*get-clauses-wl x1a* ∝ *c*))) (*get-tvdom x2b* ! *x1b*))
≤ *SPEC*
(λ*wasted*.
do {
- ← *log-del-clause-heur x2b* (*get-tvdom x2b* ! *x1b*);
- ← *ASSERT*
(*mark-garbage-pre* (*get-clauses-wl-heur x2b*, *get-tvdom x2b* ! *x1b*) ∧
1 ≤ *clss-size-lcount* (*get-learned-count x2b*) ∧ *x1b* < *length* (*get-tvdom x2b*));
RETURN (*x1b*, *mark-garbage-heur3* (*get-tvdom x2b* ! *x1b*) *x1b* (*incr-wasted-st* (*word-of-nat wasted*) *x2b*))
} ≤ *SPEC*
(λc. (*c*, *x1*, *mark-garbage-wl* (*x2a* ! *x1*) *x1a*, *delete-index-and-swap x2a x1*) ∈ *?R S*))›
if *H*:
‹(*x*, *y*) ∈ *twl-st-heur-restart-ana′ r u*›
‹*mark-to-delete-clauses-GC-wl-pre y*›
‹*mark-to-delete-clauses-GC-wl-D-heur-pre x*›
‹(*S*, *Sa*) ∈ *?sort x y*›
‹(*uu*, *xs*) ∈ *?indices S Sa*›
‹(*l*, *la*) ∈ *nat-rel*›
‹*la* ∈ {-. *True*}›
‹*length* (*get-tvdom S*) ≤ *length* (*get-clauses-wl-heur x*)›
‹(*xa*, *x′*) ∈ *?R S*›
‹*case xa of* (*i*, *S*) ⇒ *i* < *length* (*get-tvdom S*)›
‹*case x′ of* (*i*, *T*, *xs*) ⇒ *i* < *length xs*›
‹*mark-to-delete-clauses-GC-wl-inv Sa xs x′*›
‹*x2* = (*x1a*, *x2a*)›
‹*x′* = (*x1*, *x2*)›
‹*xa* = (*x1b*, *x2b*)›
‹*x1b* < *length* (*get-tvdom x2b*)›
‹*access-tvdom-at-pre x2b x1b*›
‹*clause-not-marked-to-delete-heur-pre* (*x2b*, *get-tvdom x2b* ! *x1b*)›
‹(*b*, *ba*) ∈ {(*b*, *b′*). (*b*, *b′*) ∈ *bool-rel* ∧ *b* = (*x2a* ! *x1* ∈# *dom-m* (*get-clauses-wl x1a*))}›
‹¬ ¬ *b*›
‹¬ ¬ *ba*›
‹*0* < *length* (*get-clauses-wl x1a* ∝ (*x2a* ! *x1*))›
‹*get-clauses-wl x1a* ∝ (*x2a* ! *x1*) ! *0* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1a*)›
‹*access-lit-in-clauses-heur-pre* ((*x2b*, *get-tvdom x2b* ! *x1b*), *0*)›
‹*length* (*get-clauses-wl-heur x2b*) ≤ *length* (*get-clauses-wl-heur x*)›
‹*length* (*get-tvdom x2b*) ≤ *length* (*get-clauses-wl-heur x2b*)›
‹*arena-is-valid-clause-idx* (*get-clauses-wl-heur x2b*) (*get-tvdom x2b* ! *x1b*)›
‹(*arena-length* (*get-clauses-wl-heur x2b*) (*get-tvdom x2b* ! *x1b*) ≠ *2*, *can-del*)
∈ *bool-rel*›
‹*x1* < *length x2a*›
‹*arena-length* (*get-clauses-wl-heur x2b*) (*get-tvdom x2b* ! *x1b*) ≠ *2*›
can-del
for *x y S Sa xs l la xa x′ x1 x2 x1a x2a x1b x2b b ba can-del D L K bb uu*
proof −
have [*simp*]: ‹*mark-garbage-heur3 i C* (*incr-wasted-st b S*) = *incr-wasted-st b* (*mark-garbage-heur3 i C S*)› **for** *i C S b*
by (*cases S*; *auto simp*: *mark-garbage-heur3-def incr-wasted-st-def*)
have ‹*mark-garbage-pre* (*get-clauses-wl-heur x2b*, *get-tvdom x2b* ! *x1b*)›

```
    ‹x1b < length (get-tvdom x2b)›
    using that
    unfolding prod.simps mark-garbage-pre-def
      arena-is-valid-clause-vdom-def arena-is-valid-clause-idx-def apply −
    by (rule exI[of - ‹get-clauses-wl x1a›], rule exI[of - ‹set (get-vdom x2b)›])
     (auto simp: twl-st-heur-restart twl-st-heur-restart-ana-def dest: twl-st-heur-restart-valid-arena)
  moreover have 0: ‹Suc 0 ≤ clss-size-lcount (get-learned-count x2b)›
    ‹¬ irred (get-clauses-wl x1a) (x2a ! x1)›
    using get-learned-count-ge[OF that(1−27)] only-irred[OF that(1−27)] by auto
  moreover have ‹(mark-garbage-heur3 (get-tvdom x2b ! x1) x1
    (incr-wasted-st (word-of-nat (length (get-clauses-wl x1a ∝ (get-tvdom x2b ! x1)))) x2b),
    mark-garbage-wl (get-tvdom x2b ! x1) x1a)
    ∈ twl-st-heur-restart-ana r›
    by (use that 0 in
      ‹auto intro!: incr-wasted-st mark-garbage-heur-wl-ana simp: twl-st-heur-restart
      learned-clss-count-mark-garbage-heur3
      dest: twl-st-heur-restart-valid-arena twl-st-heur-restart-anaD›)
  moreover have ‹learned-clss-count
    (mark-garbage-heur3 (get-tvdom x2b ! x1) x1
    (incr-wasted-st (word-of-nat (length (get-clauses-wl x1a ∝ (get-tvdom x2b ! x1)))) x2b))
    ≤ u›
    by (use that 0 in
      ‹auto intro!: incr-wasted-st mark-garbage-heur-wl-ana simp: twl-st-heur-restart
      learned-clss-count-mark-garbage-heur3
      dest: twl-st-heur-restart-valid-arena twl-st-heur-restart-anaD›)
  moreover have ‹xb ∈ set (get-tvdom S)›
    if ‹xb ∈ set (butlast ((get-tvdom x2b)[x1 := last (get-tvdom x2b)]))› for xb
  proof −
    have ‹xb ∈ set (get-tvdom x2b)›
      using that H by (auto dest!: in-set-butlastD)
       (metis Misc.last-in-set in-set-upd-eq len-greater-imp-nonempty)
    then show ?thesis
      using H by auto
  qed
  moreover have ‹¬irred (get-clauses-wl (mark-garbage-wl (get-tvdom x2b ! x1) x1a)) C›
    if ‹C ∈ set (butlast ((get-tvdom x2b)[x1 := last (get-tvdom x2b)]))› for C
  proof −
    have a: ‹distinct (get-tvdom x2b)› ‹x1 < length (get-tvdom x2b)›
      using H(9−15) by (auto simp: twl-st-heur-restart-ana-def
        twl-st-heur-restart-def aivdom-inv-dec-alt-def)
    then have 1: ‹get-tvdom x2b = take x1 (get-tvdom x2b) @ get-tvdom x2b ! x1 # drop (Suc x1)
(get-tvdom x2b)›
      by (subst append-take-drop-id[of x1, symmetric], subst Cons-nth-drop-Suc[symmetric])
       auto
    have ‹set (delete-index-and-swap (get-tvdom x2b) x1) =
      set (get-tvdom x2b) − {get-tvdom x2b!x1}›
      using a
      apply (subst (asm) (2)1, subst (asm) (1)1)
      apply (subst (2)1, subst (1)1)
      apply (cases ‹drop (Suc x1) (get-tvdom x2b)› rule: rev-cases)
      by (auto simp: nth-append list-update-append1 list-update-append2 butlast-append
        dest: in-set-butlastD)
    then have [simp]: ‹C ≠ get-tvdom x2b ! x1›
      using that by auto
    show ?thesis
      using that H
```

apply (*cases x1a*)
apply (*auto simp*: *mark-garbage-wl-def*)
by (*metis Misc.last-in-set in-set-butlastD in-set-upd-cases len-greater-imp-nonempty*)
qed
ultimately show *?thesis* **apply** −
using *that*
apply (*auto intro*!: *ASSERT-leI*)
apply (*subst bind-rule-complete*)
apply (*rule order-trans*)
apply (*rule log-del-clause-heur-log-clause*[**where** *r*=*r* **and** *u*=*u*])
by *auto*
qed
show *?thesis*
supply *sort-vdom-heur-def*[*simp*] *twl-st-heur-restart-anaD*[*dest*] [[*goals-limit*=*1*]]
unfolding *mark-to-delete-clauses-GC-wl-D-heur-alt-def mark-to-delete-clauses-GC-wl-D-alt-def*
access-lit-in-clauses-heur-def
apply (*intro frefI nres-relI*)
apply (*refine-vcg sort-vdom-heur-reorder-vdom-wl*[*THEN fref-to-Down*] *incr-reduction-stat find-largest-lbd-and-size*)
subgoal
unfolding *mark-to-delete-clauses-GC-wl-D-heur-pre-def* **by** *fast*
apply *assumption*
subgoal by *auto*
subgoal for *x y S T* **unfolding** *number-clss-to-keep-def* **by** (*cases S*) (*auto*)
apply (*solves auto*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def aivdom-inv-dec-alt-def*
dest!: *valid-arena-vdom-subset size-mset-mono*)
apply (*rule init*; *solves auto*)
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *access-tvdom-at-pre-def*)
subgoal for *x y S Sa u xs l la - - x1 x2 xb x′ x1c x2c x1a x2a x1b x2b*
unfolding *clause-not-marked-to-delete-heur-pre-def arena-is-valid-clause-vdom-def*
prod.simps
by (*rule exI*[*of - ‹get-clauses-wl* (*fst x2c*)›], *rule exI*[*of - ‹set* (*get-vdom x2b*)›])
(*auto simp*: *twl-st-heur-restart*
intro!: *twl-st-heur-restart-valid-arena*[*simplified*]
dest: *twl-st-heur-restart-get-tvdom-nth-get-vdom*[*simplified*])
apply (*rule mop-clause-not-marked-to-delete-heur*; *assumption*)
subgoal for *x y S Sa uu xs l la xa x′ x1 x2 x1a x2a x1b x2b*
by (*auto simp*: *twl-st-heur-restart*)
subgoal
by (*rule already-deleted*)
subgoal for *x y S Sa u xs l la - - x1 x2 xb x′ x1c x2c x1a x2a x1b x2b*
unfolding *access-lit-in-clauses-heur-pre-def prod.simps arena-lit-pre-def*
arena-is-valid-clause-idx-and-access-def
by (*rule bex-leI*[*of - ‹get-tvdom x2b* ! *x1b*›], *simp add*: *aivdom-inv-dec-alt-def*,
rule exI[*of - ‹get-clauses-wl* (*fst x2c*)›])
(*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def aivdom-inv-dec-alt-def*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def dest*!: *valid-arena-vdom-subset*
size-mset-mono)
subgoal premises *p* **using** *p*(*7*−)
by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def aivdom-inv-dec-alt-def*
dest!: *valid-arena-vdom-subset size-mset-mono*)
subgoal for *x y S Sa u xs l la - - x1 x2 xb x′ x1c x2c x1a x2a x1b x2b*
unfolding *prod.simps*
arena-is-valid-clause-vdom-def arena-is-valid-clause-idx-def

by (*rule exI*[*of* - ‹*get-clauses-wl* (*fst x2c*)›], *rule exI*[*of* - ‹*set* (*get-vdom x2b*)›])
(*auto simp*: *twl-st-heur-restart intro*!: *twl-st-heur-restart-valid-arena*[*simplified*]
dest: *twl-st-heur-restart-get-avdom-nth-get-vdom*)
subgoal
unfolding *marked-as-used-pre-def*
by (*auto simp*: *twl-st-heur-restart reason-rel-def*)
subgoal
by (*auto dest*!: *twl-st-heur-restart-anaD twl-st-heur-restart-valid-arena*[*simplified*]
simp: *arena-lifting*)
subgoal by *fast*
subgoal for *x y S Sa u xs l la - - x1 x2 xb x' x1c x2c x1a x2a x1b x2b*
unfolding *mop-arena-length-st-def*
apply (*rule mop-arena-length*[*THEN fref-to-Down-curry*, *THEN order-trans*,
of ‹*get-clauses-wl x1a*› ‹*get-tvdom x2b* ! *x1b*› - - ‹*set* (*get-vdom x2b*)›])
subgoal
by *auto*
subgoal
by (*auto simp*: *twl-st-heur-restart-valid-arena*[*simplified*])
subgoal
by (*rule length-filter-le*)
done
subgoal for *x y*
unfolding *valid-sort-clause-score-pre-def arena-is-valid-clause-vdom-def*
get-clause-LBD-pre-def arena-is-valid-clause-idx-def arena-act-pre-def
by (*force simp*: *valid-sort-clause-score-pre-def twl-st-heur-restart-ana-def arena-dom-status-iff*
arena-act-pre-def get-clause-LBD-pre-def arena-is-valid-clause-idx-def twl-st-heur-restart-def
intro!: *exI*[*of* - ‹*get-clauses-wl T*›] *dest*!: *set-mset-mono mset-subset-eqD*)
subgoal for *x y S Sa uu xs l la xa x' x1 x2 x1a x2a x1c x2c b ba L x2b*
using *size-mset-mono*[*of* ‹*mset* (*get-tvdom x2b*)› ‹*mset* (*get-vdom x2b*)›]
by (*clarsimp simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def aivdom-inv-dec-alt-def*
dest!: *valid-arena-vdom-subset*)
subgoal
by *auto*
done
qed

definition *isasat-GC-entry* :: ‹-› **where**
‹*isasat-GC-entry* $\mathcal{A}$ *vdom0 arena-old* W' = {(($arena_o$, (*arena*, *aivdom*)), (N_o, *N*)). *valid-arena* $arena_o$ N_o *vdom0* ∧ *valid-arena arena N* (*set* (*get-vdom-aivdom aivdom*)) ∧ *vdom-m* $\mathcal{A}$ W' N_o ⊆ *vdom0* ∧ *dom-m N* = *mset* (*get-vdom-aivdom aivdom*) ∧
arena-is-packed arena N ∧
aivdom-inv-strong-dec aivdom (*dom-m N*) ∧
length $arena_o$ = *length arena-old* ∧
move-is-packed $arena_o$ N_o *arena N*}›

definition *isasat-GC-refl* :: ‹-› **where**
‹*isasat-GC-refl* $\mathcal{A}$ *vdom0 arena-old* = {(($arena_o$, (*arena*, *aivdom*), *W*), (N_o, *N*, *W'*)). *valid-arena* $arena_o$ N_o *vdom0* ∧ *valid-arena arena N* (*set* (*get-vdom-aivdom aivdom*)) ∧
(*W*, *W'*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$) ∧ *vdom-m* $\mathcal{A}$ W' N_o ⊆ *vdom0* ∧ *dom-m N* = *mset* (*get-vdom-aivdom aivdom*) ∧
arena-is-packed arena N ∧ *aivdom-inv-strong-dec aivdom* (*dom-m N*) ∧
length $arena_o$ = *length arena-old* ∧
(∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *length* (*W' L*) ≤ *length* $arena_o$) ∧*move-is-packed* $arena_o$ N_o *arena N*}›

lemma *move-is-packed-empty*[*simp*]: ‹*valid-arena arena N vdom* ⟹ *move-is-packed arena N* [] *fmempty*›
by (*auto simp*: *move-is-packed-def valid-arena-ge-length-clauses*)

lemma *move-is-packed-append*:
assumes
dom: ‹*C* ∈# *dom-m x1a*› **and**
E: ‹*length E* = *length* (*x1a* ∝ *C*) + *header-size* (*x1a* ∝ *C*)› ‹(*fst E'*) = (*x1a* ∝ *C*)›
‹*n* = *header-size* (*x1a* ∝ *C*)› **and**
valid: ‹*valid-arena x1d x2a D'*› **and**
packed: ‹*move-is-packed x1c x1a x1d x2a*›
shows ‹*move-is-packed* (*extra-information-mark-to-delete x1c C*)
(*fmdrop C x1a*)
(*x1d* @ *E*)
(*fmupd* (*length x1d* + *n*) *E'* *x2a*)›
proof −
have [*simp*]: ‹(∑ *x*∈#*remove1-mset C*
(*dom-m*
x1a). *length*
(*fst* (*the* (*if x* = *C then None*
else fmlookup x1a x))) +
header-size
(*fst* (*the* (*if x* = *C then None*
else fmlookup x1a x)))) =
(∑ *x*∈#*remove1-mset C*
(*dom-m*
x1a). *length*
(*x1a* ∝ *x*) +
header-size
(*x1a* ∝ *x*))›
by (*rule sum-mset-cong*)
(*use distinct-mset-dom*[*of x1a*] **in** ‹*auto dest*!: *simp*: *distinct-mset-remove1-All*›)
have [*simp*]: ‹(*length x1d* + *header-size* (*x1a* ∝ *C*)) ∉# (*dom-m x2a*)›
using *valid arena-lifting*(2) **by** *blast*
have [*simp*]: ‹(∑ *x*∈#(*dom-m x2a*). *length*
(*fst* (*the* (*if length x1d* + *header-size* (*x1a* ∝ *C*) = *x*
then Some E'
else fmlookup x2a x))) +
header-size
(*fst* (*the* (*if length x1d* + *header-size* (*x1a* ∝ *C*) = *x*
then Some E'
else fmlookup x2a x)))) =
(∑ *x*∈#*dom-m x2a*. *length*
(*x2a* ∝ *x*) +
header-size
(*x2a* ∝ *x*))›
by (*rule sum-mset-cong*)
(*use distinct-mset-dom*[*of x2a*] **in** ‹*auto dest*!: *simp*: *distinct-mset-remove1-All*›)
show *?thesis*
using *packed dom E*
by (*auto simp*: *move-is-packed-def split*: *if-splits dest*!: *multi-member-split*)
qed

lemma *isasat-GC-clauses-prog-copy-wl-entry*:
assumes ‹*valid-arena arena N vdom0*› **and**
‹*valid-arena arena' N'* (*set* (*get-vdom-aivdom aivdom*))› **and**
vdom: ‹*vdom-m* 𝒜 *W N* ⊆ *vdom0*› **and**

L: ‹*atm-of A* ∈# $\mathcal{A}$› **and**
L′-L: ‹(*A′*, *A*) ∈ *nat-lit-lit-rel*› **and**
W: ‹(*W′* , *W*) ∈ ⟨*Id*⟩*map-fun-rel* (*D*$_0$ $\mathcal{A}$)› **and**
‹*dom-m N′* = *mset* (*get-vdom-aivdom aivdom*)› **and**
‹*arena-is-packed arena′ N′*› **and**
ivdom: ‹*aivdom-inv-strong-dec aivdom* (*dom-m N′*)› **and**
r: ‹*length arena* = *r*› **and**
le: ‹∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *length* (*W L*) ≤ *length arena*› **and**
packed: ‹*move-is-packed arena N arena′ N′*›
shows ‹*isasat-GC-clauses-prog-copy-wl-entry arena W′ A′* (*arena′*, *aivdom*)
≤ ⇓ (*isasat-GC-entry* $\mathcal{A}$ *vdom0 arena W*)
(*cdcl-GC-clauses-prog-copy-wl-entry N* (*W A*) *A N′*)›
(**is** ‹- ≤ ⇓ (*?R*) -›)
proof −
have *A*: ‹*A′* = *A*› **and** *K*[*simp*]: ‹*W′* ! *nat-of-lit A* = *W A*›
using *L′-L L W* **apply** *auto*
by (*cases A*) (*auto simp*: *map-fun-rel-def* $\mathcal{L}_{all}$*-add-mset dest*!: *multi-member-split*)
have *A-le*: ‹*nat-of-lit A* < *length W′*›
using *W L* **by** (*cases A*; *auto simp*: *map-fun-rel-def* $\mathcal{L}_{all}$*-add-mset dest*!: *multi-member-split*)
have *length-slice*: ‹*C* ∈# *dom-m x1a* ⟹ *valid-arena x1c x1a vdom′* ⟹
length
(*Misc.slice* (*C* − *header-size* (*x1a* ∝ *C*))
(*C* + *arena-length x1c C*) *x1c*) =
arena-length x1c C + *header-size* (*x1a* ∝ *C*)› **for** *x1c x1a C vdom′*
using *arena-lifting*(*1*−*4*,*10*)[*of x1c x1a vdom′ C*]
by (*auto simp*: *header-size-def slice-len-min-If min-def split*: *if-splits*)
show *?thesis*
unfolding *isasat-GC-clauses-prog-copy-wl-entry-def cdcl-GC-clauses-prog-copy-wl-entry-def prod.case A*
arena-header-size-def[*symmetric*]
apply (*refine-vcg ASSERT-leI WHILET-refine*[**where** *R* = ‹*nat-rel* ×$_r$ *?R*›])
subgoal using *A-le* **by** (*auto simp*: *isasat-GC-entry-def*)
subgoal using *le L K* **by** (*cases A*) (*auto dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$*-add-mset*)
subgoal using *assms* **by** (*auto simp*: *isasat-GC-entry-def*)
subgoal using *W L* **by** *auto*
subgoal by *auto*
subgoal for *x x′ x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d*
using *vdom L*
unfolding *arena-is-valid-clause-vdom-def K isasat-GC-entry-def*
by (*cases A*)
(*force dest*!: *multi-member-split simp*: *vdom-m-def* $\mathcal{L}_{all}$*-add-mset*)+
subgoal
using *vdom L*
unfolding *arena-is-valid-clause-vdom-def K isasat-GC-entry-def*
by (*subst arena-dom-status-iff*)
(*cases A* ; *auto dest*!: *multi-member-split simp*: *arena-lifting arena-dom-status-iff*
vdom-m-def $\mathcal{L}_{all}$*-add-mset*; *fail*)+
subgoal
unfolding *arena-is-valid-clause-idx-def isasat-GC-entry-def*
by *auto*
subgoal unfolding *isasat-GC-entry-def move-is-packed-def arena-is-packed-def*
by (*auto simp*: *valid-arena-header-size arena-lifting dest*!: *multi-member-split*)
subgoal using *r* **by** (*auto simp*: *isasat-GC-entry-def*)
subgoal by (*auto dest*: *valid-arena-header-size simp*: *arena-lifting dest*!: *valid-arena-vdom-subset multi-member-split simp*: *arena-header-size-def isasat-GC-entry-def aivdom-inv-strong-dec-alt-def*
split: *if-splits*)

subgoal by (*auto simp*: *isasat-GC-entry-def aivdom-inv-strong-dec-alt-def dest*!: *size-mset-mono*)
subgoal by (*auto simp*: *isasat-GC-entry-def aivdom-inv-strong-dec-alt-def dest*!: *size-mset-mono*)
subgoal by (*auto simp*: *isasat-GC-entry-def aivdom-inv-strong-dec-alt-def dest*!: *size-mset-mono*)
subgoal
by (*force simp*: *isasat-GC-entry-def dest*: *arena-lifting*(*2*))
subgoal by (*auto simp*: *arena-header-size-def*)
subgoal for *x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d D*
using *valid-arena-in-vdom-le-arena*(*1*)[*of x1d x2a ‹set* (*get-vdom-aivdom x2d*)*› D*] **apply** −

by (*rule order-trans*[*OF fm-mv-clause-to-new-arena*])
(*force intro*: *valid-arena-extra-information-mark-to-delete'*
simp: *arena-lifting remove-1-mset-id-iff-notin*
mark-garbage-pre-def isasat-GC-entry-def min-def
valid-arena-header-size
simp del: *aivdom-inv.simps*
dest: *in-vdom-m-fmdropD arena-lifting*(*2*)
intro!: *arena-is-packed-append-valid subset-mset-trans-add-mset*
aivdom-inv-dec-intro-init-strong-add-mset aivdom-inv-dec-intro-learned-strong-add-mset
move-is-packed-append length-slice aivdom-inv-intro-add-mset)+
subgoal
by *auto*
subgoal
by *auto*
done
qed

lemma *isasat-GC-clauses-prog-single-wl*:
assumes
‹(*X*, *X'*) ∈ *isasat-GC-refl* $\mathcal{A}$ *vdom0 arena0*› **and**
X: ‹*X* = (*arena*, (*arena'*, *aivdom*), *W*)› ‹*X'* = (*N*, *N'*, *W'*)› **and**
L: ‹*A* ∈# $\mathcal{A}$› **and**
st: ‹(*A*, *A'*) ∈ *Id*› **and**
st': ‹*narena* = (*arena'*, *aivdom*)› **and**
ae: ‹*length arena0* = *length arena*› **and**
le-all: ‹∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *length* (*W' L*) ≤ *length arena*›
shows ‹*isasat-GC-clauses-prog-single-wl arena narena W A*
≤ ⇓ (*isasat-GC-refl* $\mathcal{A}$ *vdom0 arena0*)
(*cdcl-GC-clauses-prog-single-wl N W' A' N'*)›
(**is** ‹- ≤ ⇓ *?R* -›)
proof −
let *?vdom* = ‹*get-vdom-aivdom aivdom*›
have *H*:
‹*valid-arena arena N vdom0*›
‹*valid-arena arena' N'* (*set ?vdom*)› **and**
vdom: ‹*vdom-m* $\mathcal{A}$ *W' N* ⊆ *vdom0*› **and**
L: ‹*A* ∈# $\mathcal{A}$› **and**
eq: ‹*A'* = *A*› **and**
WW': ‹(*W*, *W'*) ∈ ⟨*Id*⟩*map-fun-rel* (*D*$_0$ $\mathcal{A}$)› **and**
vdom-dom: ‹*dom-m N'* = *mset ?vdom*› **and**
packed: ‹*arena-is-packed arena' N'*› **and**
aivdom: ‹*aivdom-inv-strong-dec aivdom* (*dom-m N'*)› **and**
packed2: ‹*move-is-packed arena N arena' N'*› **and**
incl: ‹*vdom-m* $\mathcal{A}$ *W' N* ⊆ *vdom0*›
using *assms X st* **by** (*auto simp*: *isasat-GC-refl-def*)

have *vdom2*: ‹*vdom-m* $\mathcal{A}$ *W' x1* ⊆ *vdom0* ⟹ *vdom-m* $\mathcal{A}$ (*W'*(*L* := [])) *x1* ⊆ *vdom0*› **for** *x1 L*

by (*force simp*: *vdom-m-def dest*!: *multi-member-split*)
have *vdom-m-upd*: ‹$x \in$ *vdom-m* $\mathcal{A}$ (*W*(*Pos A* := [], *Neg A* := [])) *N* $\Longrightarrow x \in$ *vdom-m* $\mathcal{A}$ *W N*› **for** *x W A N*
by (*auto simp*: *image-iff vdom-m-def dest*: *multi-member-split*)
have *vdom-m3*: ‹$x \in$ *vdom-m* $\mathcal{A}$ *W a* $\Longrightarrow$ *dom-m a* $\subseteq\#$ *dom-m b* $\Longrightarrow$ *dom-m b* $\subseteq\#$ *dom-m c* $\Longrightarrow x \in$ *vdom-m* $\mathcal{A}$ *W c*› **for** *a b c W x*
unfolding *vdom-m-def* **by** *auto*
have *W*: ‹(*W*[*2* $*$ *A* := [], *Suc* (*2* $*$ *A*) := []], *W'*(*Pos A* := [], *Neg A* := []))
$\in$ ⟨*Id*⟩*map-fun-rel* (D_0 $\mathcal{A}$)› **for** *A*
using *WW'* **unfolding** *map-fun-rel-def*
apply *clarify*
apply (*intro conjI*)
apply *auto*[]
apply (*drule multi-member-split*)
apply (*case-tac L*)
apply (*auto dest*!: *multi-member-split*)
done
have *le*: ‹*nat-of-lit* (*Pos A*) < *length W*› ‹*nat-of-lit* (*Neg A*) < *length W*›
using *WW' L* **by** (*auto dest*!: *multi-member-split simp*: *map-fun-rel-def* $\mathcal{L}_{all}$*-add-mset*)
have [*refine0*]: ‹*RETURN* (*Pos A*) $\le \Downarrow$ *Id* (*RES* {*Pos A*, *Neg A*})› **by** *auto*
have *vdom-upD*:‹ $x \in$ *vdom-m* $\mathcal{A}$ (*W'*(*Pos A* := [], *Neg A* := [])) *xd* $\Longrightarrow x \in$ *vdom-m* $\mathcal{A}$ (λ*a*. *if a* = *Pos A then* [] *else W' a*) *xd*›
for *W' a A x xd*
by (*auto simp*: *vdom-m-def*)
show *?thesis*
unfolding *isasat-GC-clauses-prog-single-wl-def*
cdcl-GC-clauses-prog-single-wl-def eq st' isasat-GC-refl-def
apply (*refine-vcg*
isasat-GC-clauses-prog-copy-wl-entry[**where** *r*= ‹*length arena*› **and** $\mathcal{A}$ = $\mathcal{A}$])
subgoal using *le* **by** *auto*
subgoal using *le* **by** *auto*
apply (*rule H*(*1*); *fail*)
apply (*rule H*(*2*); *fail*)
subgoal using *incl* **by** *auto*
subgoal using *L* **by** *auto*
subgoal using *WW'* **by** *auto*
subgoal using *vdom-dom* **by** *blast*
subgoal using *packed* **by** *blast*
subgoal using *aivdom* **by** *blast*
subgoal by *blast*
subgoal using *le-all* **by** *auto*
subgoal using *packed2* **by** *auto*
subgoal using *ae* **by** (*auto simp*: *isasat-GC-entry-def*)
apply (*solves* ‹*auto simp*: *isasat-GC-entry-def*›)
apply (*solves* ‹*auto simp*: *isasat-GC-entry-def*›)
apply (*rule vdom2*; *auto*)
supply *isasat-GC-entry-def*[*simp*]
subgoal using *WW'* **by** (*auto simp*: *map-fun-rel-def dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$*-add-mset*)
subgoal using *L* **by** *auto*
subgoal using *L* **by** *auto*
subgoal using *WW'* **by** (*auto simp*: *map-fun-rel-def dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$*-add-mset*)
subgoal using *WW'* **by** (*auto simp*: *map-fun-rel-def dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$*-add-mset*)
subgoal using *WW' le-all* **by** (*auto simp*: *map-fun-rel-def dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$*-add-mset*)
subgoal using *WW' le-all* **by** (*auto simp*: *map-fun-rel-def dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$*-add-mset*)
subgoal using *WW' le-all* **by** (*auto simp*: *map-fun-rel-def dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$*-add-mset*)
subgoal using *WW' le-all* **by** (*auto simp*: *map-fun-rel-def dest*!: *multi-member-split simp*: $\mathcal{L}_{all}$*-add-mset*)

subgoal using *W ae le-all aivdom* **by** (*auto simp*: *dest*!: *vdom-upD*)
done
qed

definition *cdcl-GC-clauses-prog-wl2* **where**
‹*cdcl-GC-clauses-prog-wl2* = (λ*N0* $\mathcal{A}0$ *WS*. *do* {
$\mathcal{A} \leftarrow$ *SPEC*(λ$\mathcal{A}$. *set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{A}0$);
(-, (*N*, *N'*, *WS*)) ← $WHILE_T^{cdcl\text{-}GC\text{-}clauses\text{-}prog\text{-}wl\text{-}inv\ \mathcal{A}\ N0}$
(λ($\mathcal{B}$, -). $\mathcal{B} \neq \{\#\}$)
(λ($\mathcal{B}$, (*N*, *N'*, *WS*)). *do* {
ASSERT($\mathcal{B} \neq \{\#\}$);
A ← *SPEC* (λ*A*. *A* ∈# $\mathcal{B}$);
(*N*, *N'*, *WS*) ← *cdcl-GC-clauses-prog-single-wl N WS A N'*;
RETURN (*remove1-mset A* $\mathcal{B}$, (*N*, *N'*, *WS*))
})
($\mathcal{A}$, (*N0*, *fmempty*, *WS*));
RETURN (*N*, *N'*, *WS*)
})›

lemma *cdcl-GC-clauses-prog-wl-inv-cong-empty*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹
cdcl-GC-clauses-prog-wl-inv $\mathcal{A}$ *N* ({#}, *x*) ⟹ *cdcl-GC-clauses-prog-wl-inv* $\mathcal{B}$ *N* ({#}, *x*)›
by (*auto simp*: *cdcl-GC-clauses-prog-wl-inv-def*)

lemma *isasat-GC-clauses-prog-wl2*:
assumes ‹*valid-arena arena*$_o$ *N*$_o$ *vdom0*› **and**
‹*valid-arena arena N* (*set vdom*)› **and**
vdom: ‹*vdom-m* $\mathcal{A}$ *W'* *N*$_o$ ⊆ *vdom0*› **and**
vmtf: ‹*ns* ∈ *bump-heur* $\mathcal{A}$ *M*› **and**
nempty: ‹$\mathcal{A} \neq \{\#\}$› **and**
W-W': ‹(*W*, *W'*) ∈ ⟨*Id*⟩*map-fun-rel* (*D*$_0$ $\mathcal{A}$)› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$› **and** *old*: ‹*old-arena* = []› **and**
le-all: ‹∀ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$. *length* (*W' L*) ≤ *length arena*$_o$›
shows
‹*isasat-GC-clauses-prog-wl2 ns* (*arena*$_o$, (*old-arena*, *empty-aivdom aivdom*), *W*)
≤ ⇓ ({((*arena*$_o$', (*arena*, *aivdom*), *W*), (*N*$_o$', *N*, *W'*)). *valid-arena arena*$_o$' *N*$_o$' *vdom0* ∧
valid-arena arena N (*set* (*get-vdom-aivdom aivdom*)) ∧
(*W*, *W'*) ∈ ⟨*Id*⟩*map-fun-rel* (*D*$_0$ $\mathcal{A}$) ∧ *vdom-m* $\mathcal{A}$ *W'* *N*$_o$' ⊆ *vdom0* ∧
cdcl-GC-clauses-prog-wl-inv $\mathcal{A}$ *N*$_o$ ({#}, *N*$_o$', *N*, *W'*) ∧ *dom-m N* = *mset* (*get-vdom-aivdom aivdom*) ∧
arena-is-packed arena N ∧ *aivdom-inv-strong-dec aivdom* (*dom-m N*) ∧
length arena$_o$' = *length arena*$_o$})
(*cdcl-GC-clauses-prog-wl2* *N*$_o$ $\mathcal{A}$ *W'*)›
proof −
define *f* **where**
‹*f A* ≡ (λ(*arena*$_o$, *arena*, *W*). *isasat-GC-clauses-prog-single-wl arena*$_o$ *arena W A*)› **for** *A* :: *nat*
let *?R* = ‹{((*A'*, *arena*$_o$', (*arena*, *vdom*), *W*), ($\mathcal{A}''$, *N*$_o$', *N*, *W'*)). *A'* = $\mathcal{A}''$ ∧
((*arena*$_o$', (*arena*, *vdom*), *W*), (*N*$_o$', *N*, *W'*)) ∈ *isasat-GC-refl* $\mathcal{A}$ *vdom0 arena*$_o$ ∧
length arena$_o$' = *length arena*$_o$}›
have *H*: ‹(*X*, *X'*) ∈ *?R* ⟹ *X* = (*x1*, *x2*) ⟹ *x2* = (*x3*, *x4*) ⟹ *x4* = (*x5*, *x6*) ⟹
X' = (*x1'*, *x2'*) ⟹ *x2'* = (*x3'*, *x4'*) ⟹ *x4'* = (*x5'*, *x6'*) ⟹
((*x3*, (*fst x5*, (*snd x5*)), *x6*), (*x3'*, *x5'*, *x6'*)) ∈ *isasat-GC-refl* $\mathcal{A}$ *vdom0 arena*$_o$›
for *X X' A B x1 x1' x2 x2' x3 x3' x4 x4' x5 x5' x6 x6' x0 x0' x x'*
supply [[*show-types*]]
by *auto*

have *isasat-GC-clauses-prog-wl-alt-def*:
‹*isasat-GC-clauses-prog-wl2 n x0 = iterate-over-VMTF f* (λ*x*. *length* (*fst x*) = *length* (*fst x0*)) (*fst* (*bump-get-heuristics n*), *Some* (*bumped-vmtf-array-fst n*)) *x0*›
(**is** ‹*?A* = *?B*›)
for *n x0*
proof −
have [*refine0*]: ‹((*Some* (*fst* (*snd* (*snd* (*bump-get-heuristics n*)))), *x0*),
snd (*fst* (*bump-get-heuristics n*), *Some* (*fst* ((*snd* (*snd* (*bump-get-heuristics n*)))))), *x0*) ∈ *Id*›
‹((*snd* (*fst* (*bump-get-heuristics n*), *Some* (*fst* ((*snd* (*snd* (*bump-get-heuristics n*)))))), *x0*),
Some (*fst* (*snd* ((*snd* (*bump-get-heuristics n*))))), *x0*) ∈ *Id*›
‹*a*=*a'* ⟹ *b*=*b'* ⟹ *c*=*c'* ⟹ *d*=*d'* ⟹ *isasat-GC-clauses-prog-single-wl a b c d* ≤ ⇓*Id* (*isasat-GC-clauses-prog-single-wl a' b' c' d'*)›
for *a' b' c' d' a b c d*
by *auto*
have ‹*?A* ≤ ⇓*Id ?B*›
unfolding *f-def isasat-GC-clauses-prog-wl2-def iterate-over-VMTF-def*
bumped-vmtf-array-fst-def access-focused-vmtf-array-def nres-monad3
case-prod-beta
by *refine-rcg* (*solves* ‹(*auto simp*: *length-bumped-vmtf-array-def*)›)+
moreover have ‹*?B* ≤ ⇓*Id ?A*›
unfolding *f-def isasat-GC-clauses-prog-wl2-def iterate-over-VMTF-def*
bumped-vmtf-array-fst-def access-focused-vmtf-array-def nres-monad3
case-prod-beta
by *refine-vcg* (*solves* ‹(*auto simp*: *length-bumped-vmtf-array-def*)›)+
ultimately show *?thesis*
by *auto*
qed
have *empty*[*simp*]: ‹*aivdom-inv-dec* (*AIvdom* ([], [], [], [])) {#}›
by (*auto simp*: *aivdom-inv-dec-alt-def*)
obtain *M'* **where** *vmtf'*: ‹(*fst* (*bump-get-heuristics ns*), *fst* (*snd* (*bump-get-heuristics ns*)),
bumped-vmtf-array-fst ns, *fst* (*snd* (*snd* (*snd* (*bump-get-heuristics ns*)))), *snd* (*snd* (*snd* (*snd* (*bump-get-heuristics ns*))))) ∈ *vmtf* *A* *M'*› **and**
‹*M'* = *M* ∨ *M'* = *get-unit-trail M*›
using *vmtf* **unfolding** *bump-heur-def*
by (*cases* ‹*bump-get-heuristics ns*›) (*auto simp*: *bump-get-heuristics-def bumped-vmtf-array-fst-def*
split: *if-splits*)
show *?thesis*
unfolding *isasat-GC-clauses-prog-wl-alt-def prod.case f-def*[*symmetric*] *old*
apply (*rule order-trans*[*OF iterate-over-VMTF-iterate-over-*$\mathcal{L}_{all}$[*OF vmtf' nempty bounded*,
where *I'* = ‹λ- *x*. *length* (*fst x*) = *length* (*fst* (*arena*$_o$, ([], *empty-aivdom aivdom*), *W*))›]])
subgoal by *auto*
unfolding *Down-id-eq iterate-over-*$\mathcal{L}_{all}$*-def cdcl-GC-clauses-prog-wl2-def f-def*
apply (*refine-vcg WHILEIT-refine-with-invariant-and-break*[**where** *R* = *?R*]
isasat-GC-clauses-prog-single-wl)
subgoal by *fast*
subgoal using *assms* **by** (*auto simp*: *valid-arena-empty isasat-GC-refl-def*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
apply (*rule H*; *assumption*; *fail*)
apply (*rule refl*)+
subgoal by (*auto simp add*: *cdcl-GC-clauses-prog-wl-inv-def*)
subgoal by *auto*
subgoal by *auto*
subgoal using *le-all* **by** (*auto simp*: *isasat-GC-refl-def split*: *prod.splits*)

subgoal by (*auto simp*: *isasat-GC-refl-def*)
subgoal by (*auto simp*: *isasat-GC-refl-def*
dest: *cdcl-GC-clauses-prog-wl-inv-cong-empty*)
done
qed

lemma *cdcl-GC-clauses-prog-wl-alt-def*:
‹*cdcl-GC-clauses-prog-wl* = (λ(*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *WS*). *do* {
ASSERT(*cdcl-GC-clauses-pre-wl* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*,*NS*, *US*, *N0*, *U0*, *Q*, *WS*));
(*N*, *N'*, *WS*) ← *cdcl-GC-clauses-prog-wl2 N* (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) *WS*;
RETURN (*M*, *N'*, *D*, *NE*, *UE*, *NEk*, *UEk*,*NS*, *US*, *N0*, *U0*, *Q*, *WS*)
})›
proof −
have [*refine0*]: ‹(*x1c*, *x1*) ∈ *Id* ⟹ *RES* (*set-mset x1c*)
≤ ⇓ *Id* (*RES* (*set-mset x1*))› **for** *x1 x1c*
by *auto*
have [*refine0*]: ‹(*xa*, *x'*) ∈ *Id* ⟹
x2a = (*x1b*, *x2b*) ⟹
x2 = (*x1a*, *x2a*) ⟹
x' = (*x1*, *x2*) ⟹
x2d = (*x1e*, *x2e*) ⟹
x2c = (*x1d*, *x2d*) ⟹
xa = (*x1c*, *x2c*) ⟹
(*A*, *Aa*) ∈ *Id* ⟹
cdcl-GC-clauses-prog-single-wl x1d x2e A x1e
≤ ⇓ *Id*
(*cdcl-GC-clauses-prog-single-wl x1a x2b Aa x1b*)›
for *𝒜 x xa x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e A aaa Aa*
by *auto*
show *?thesis*
unfolding *cdcl-GC-clauses-prog-wl-def cdcl-GC-clauses-prog-wl2-def*
while.imonad3

apply (*intro ext*)
apply (*clarsimp simp add*: *while.imonad3*)
apply (*subst dual-order.eq-iff*[*of* ‹(- :: - *nres*)›])
apply (*intro conjI*)
subgoal
by (*rewrite at* ‹- ≤ ⌑› *Down-id-eq*[*symmetric*])
(*refine-rcg WHILEIT-refine*[**where** *R* = *Id*], *auto simp add*: *all-init-atms-st-def*)
subgoal
by (*rewrite at* ‹- ≤ ⌑› *Down-id-eq*[*symmetric*])
(*refine-rcg WHILEIT-refine*[**where** *R* = *Id*], *auto simp add*: *all-init-atms-st-def*)
done
qed

lemma *length-watched-le''*:
assumes
xb-x'a: ‹(*x1a*, *x1*) ∈ *twl-st-heur-restart*› **and**
prop-inv: ‹*correct-watching'-nobin x1*›
shows ‹∀ *x2* ∈# *ℒ*$_{all}$ (*all-init-atms-st x1*). *length* (*watched-by x1 x2*) ≤ *length* (*get-clauses-wl-heur x1a*)›
proof
fix *x2*

assume *x2*: ‹*x2* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1*)›
have ‹*correct-watching′-nobin x1*›
using *prop-inv* **unfolding** *unit-propagation-outer-loop-wl-inv-def*
unit-propagation-outer-loop-wl-inv-def
by *auto*
then have *dist*: ‹*distinct-watched* (*watched-by x1 x2*)›
using *x2*
by (*cases x1*; *auto simp*: $\mathcal{L}_{all}$*-all-init-atms correct-watching′-nobin.simps*
simp flip: *all-init-lits-def all-init-lits-alt-def*)
then have *dist*: ‹*distinct-watched* (*watched-by x1 x2*)›
using *xb-x′a*
by (*cases x1*; *auto simp*: $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm correct-watching.simps*)
have *dist-vdom*: ‹*distinct* (*get-vdom x1a*)›
using *xb-x′a*
by (*cases x1*)
(*auto simp*: *twl-st-heur-restart-def aivdom-inv-dec-alt-def*)
have *x2*: ‹*x2* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1*)›
using *x2 xb-x′a* **unfolding** *all-init-atms-def all-init-lits-def*
by *auto*

have
valid: ‹*valid-arena* (*get-clauses-wl-heur x1a*) (*get-clauses-wl x1*) (*set* (*get-vdom x1a*))›
using *xb-x′a* **unfolding** *all-atms-def all-lits-def*
by (*cases x1*)
(*auto simp*: *twl-st-heur-restart-def*)

have ‹*vdom-m* (*all-init-atms-st x1*) (*get-watched-wl x1*) (*get-clauses-wl x1*) ⊆ *set* (*get-vdom x1a*)›
using *xb-x′a*
by (*cases x1*)
(*auto simp*: *twl-st-heur-restart-def all-atms-def*[*symmetric*] *all-init-atms-st-def*)
then have *subset*: ‹*set* (*map fst* (*watched-by x1 x2*)) ⊆ *set* (*get-vdom x1a*)›
using *x2* **unfolding** *vdom-m-def*
by (*cases x1*)
(*force simp*: *twl-st-heur-restart-def simp flip*: *all-init-atms-def*
dest!: *multi-member-split*)
have *watched-incl*: ‹*mset* (*map fst* (*watched-by x1 x2*)) ⊆# *mset* (*get-vdom x1a*)›
by (*rule distinct-subseteq-iff*[*THEN iffD1*])
(*use dist*[*unfolded distinct-watched-alt-def*] *dist-vdom subset* **in**
‹*simp-all flip*: *distinct-mset-mset-distinct*›)
have *vdom-incl*: ‹*set* (*get-vdom x1a*) ⊆ {*MIN-HEADER-SIZE*..< *length* (*get-clauses-wl-heur x1a*)}›
using *valid-arena-in-vdom-le-arena*[*OF valid*] *arena-dom-status-iff*[*OF valid*] **by** *auto*

have ‹*length* (*get-vdom x1a*) ≤ *length* (*get-clauses-wl-heur x1a*)›
by (*subst distinct-card*[*OF dist-vdom*, *symmetric*])
(*use card-mono*[*OF - vdom-incl*] **in** *auto*)
then show ‹*length* (*watched-by x1 x2*) ≤ *length* (*get-clauses-wl-heur x1a*)›
using *size-mset-mono*[*OF watched-incl*] *xb-x′a*
by (*auto intro*!: *order-trans*[*of* ‹*length* (*watched-by x1 x2*)› ‹*length* (*get-vdom x1a*)›])
qed

lemma *length-watched-le′′′*:
assumes
xb-x′a: ‹(*x1a*, *x1*) ∈ *twl-st-heur-restart*› **and**
prop-inv: ‹*no-lost-clause-in-WL x1*›
shows ‹∀ *x2* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1*). *length* (*watched-by x1 x2*) ≤ *length* (*get-clauses-wl-heur x1a*)›

proof
fix *x2*
assume *x2*: ‹*x2* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1*)›
have *dist*: ‹*distinct-watched* (*watched-by x1 x2*)›
using *x2 prop-inv*
by (*cases x1*; *auto simp*: $\mathcal{L}_{all}$*-all-init-atms no-lost-clause-in-WL-def*
simp flip: *all-init-lits-def all-init-lits-alt-def*)
then have *dist*: ‹*distinct-watched* (*watched-by x1 x2*)›
using *xb-x'a*
by (*cases x1*; *auto simp*: $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm correct-watching.simps*)
have *dist-vdom*: ‹*distinct* (*get-vdom x1a*)›
using *xb-x'a*
by (*cases x1*)
(*auto simp*: *twl-st-heur-restart-def aivdom-inv-dec-alt-def*)
have *x2*: ‹*x2* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1*)›
using *x2 xb-x'a* **unfolding** *all-init-atms-def all-init-lits-def*
by *auto*

have
valid: ‹*valid-arena* (*get-clauses-wl-heur x1a*) (*get-clauses-wl x1*) (*set* (*get-vdom x1a*))›
using *xb-x'a* **unfolding** *all-atms-def all-lits-def*
by (*cases x1*)
(*auto simp*: *twl-st-heur-restart-def*)

have ‹*vdom-m* (*all-init-atms-st x1*) (*get-watched-wl x1*) (*get-clauses-wl x1*) ⊆ *set* (*get-vdom x1a*)›
using *xb-x'a*
by (*cases x1*)
(*auto simp*: *twl-st-heur-restart-def all-atms-def*[*symmetric*] *all-init-atms-st-def*)
then have *subset*: ‹*set* (*map fst* (*watched-by x1 x2*)) ⊆ *set* (*get-vdom x1a*)›
using *x2* **unfolding** *vdom-m-def*
by (*cases x1*)
(*force simp*: *twl-st-heur-restart-def simp flip*: *all-init-atms-def*
dest!: *multi-member-split*)
have *watched-incl*: ‹*mset* (*map fst* (*watched-by x1 x2*)) ⊆# *mset* (*get-vdom x1a*)›
by (*rule distinct-subseteq-iff*[*THEN iffD1*])
(*use dist*[*unfolded distinct-watched-alt-def*] *dist-vdom subset* **in**
‹*simp-all flip*: *distinct-mset-mset-distinct*›)
have *vdom-incl*: ‹*set* (*get-vdom x1a*) ⊆ {*MIN-HEADER-SIZE*..< *length* (*get-clauses-wl-heur x1a*)}›
using *valid-arena-in-vdom-le-arena*[*OF valid*] *arena-dom-status-iff*[*OF valid*] **by** *auto*

have ‹*length* (*get-vdom x1a*) ≤ *length* (*get-clauses-wl-heur x1a*)›
by (*subst distinct-card*[*OF dist-vdom*, *symmetric*])
(*use card-mono*[*OF - vdom-incl*] **in** *auto*)
then show ‹*length* (*watched-by x1 x2*) ≤ *length* (*get-clauses-wl-heur x1a*)›
using *size-mset-mono*[*OF watched-incl*] *xb-x'a*
by (*auto intro*!: *order-trans*[*of* ‹*length* (*watched-by x1 x2*)› ‹*length* (*get-vdom x1a*)›])
qed

definition *twl-st-heur-restart-strong-aivdom* :: ‹(*isasat* × *nat twl-st-wl*) *set*› **where**
[*unfolded Let-def*]: ‹*twl-st-heur-restart-strong-aivdom* =
{(*S*, *T*).
let M' = *get-trail-wl-heur S*; *N'* = *get-clauses-wl-heur S*; *D'* = *get-conflict-wl-heur S*;
W' = *get-watched-wl-heur S*; *j* = *literals-to-update-wl-heur S*; *outl* = *get-outlearned-heur S*;
cach = *get-conflict-cach S*; *clvls* = *get-count-max-lvls-heur S*;
vm = *get-vmtf-heur S*;
vdom = *get-aivdom S*; *heur* = *get-heur S*; *old-arena* = *get-old-arena S*;

lcount = *get-learned-count S*; *occs* = *get-occs S in*
let M = *get-trail-wl T*; *N* = *get-clauses-wl T*; *D* = *get-conflict-wl T*;
Q = *literals-to-update-wl T*;
W = *get-watched-wl T*; *N0* = *get-init-clauses0-wl T*; *U0* = *get-learned-clauses0-wl T*;
NS = *get-subsumed-init-clauses-wl T*; *US* = *get-subsumed-learned-clauses-wl T*;
NEk = *get-kept-unit-init-clss-wl T*; *UEk* = *get-kept-unit-learned-clss-wl T*;
NE = *get-unkept-unit-init-clss-wl T*; *UE* = *get-unkept-unit-learned-clss-wl T in*
(*M'*, *M*) ∈ *trail-pol* (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) ∧
valid-arena N' N (*set* (*get-vdom-aivdom vdom*)) ∧
(*D'*, *D*) ∈ *option-lookup-clause-rel* (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) ∧
(*D* = *None* ⟶ *j* ≤ *length M*) ∧
Q = *uminus* ‘# *lit-of* ‘# *mset* (*drop j* (*rev M*)) ∧
(*W'*, *W*) ∈ ⟨*Id*⟩*map-fun-rel* (*D*$_0$ (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*))) ∧
vm ∈ *bump-heur* (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) *M* ∧
no-dup M ∧
clvls ∈ *counts-maximum-level M D* ∧
cach-refinement-empty (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) *cach* ∧
out-learned M D outl ∧
clss-size-corr-restart N NE {#} *NEk UEk NS* {#} *N0* {#} *lcount* ∧
vdom-m (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) *W N* ⊆ *set* (*get-vdom-aivdom vdom*) ∧
aivdom-inv-strong-dec vdom (*dom-m N*) ∧
isasat-input-bounded (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) ∧
isasat-input-nempty (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) ∧
old-arena = [] ∧
heuristic-rel (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) *heur* ∧
(*occs*, *empty-occs-list* (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*))) ∈ *occurrence-list-ref*
}›

lemma *isasat-GC-clauses-prog-wl*:
‹(*isasat-GC-clauses-prog-wl*, *cdcl-GC-clauses-prog-wl*) ∈
{(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart* ∧ *learned-clss-count S* ≤ *u*} →$_f$
⟨{(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart-strong-aivdom* ∧ *arena-is-packed* (*get-clauses-wl-heur S*) (*get-clauses-wl T*) ∧
learned-clss-count S ≤ *u*}⟩*nres-rel*›
(**is** ‹- ∈ *?T* →$_f$ -›)
proof−
have [*refine0*]: ‹⋀*x1 x1a x1b x1c x1d x1e x2e x1f x1g x1h x1i x1j x1m x1n x1o x1p x2n x2o x1q x1r x1s x1t x1u x1v x1w x1x x1y x1z x1aa x1ab x2ab NS US NEk UEk stats S*.
(*S*,
x1, *x1a*, *x1b*, *x1c*, *x1d*, *NEk*, *UEk*, *NS*, *US*, *x1e*, *x2e*)
∈ *?T* ⟹
valid-arena (*get-clauses-wl-heur S*) (*x1a*) (*set* (*get-vdom S*))›
unfolding *twl-st-heur-restart-def*
by *auto*
have [*refine0*]: ‹⋀*x1 x1a x1b x1c x1d x1e x2e x1f x1g x1h x1i x1j x1m x1n x1o x1p x2n x2o x1q x1r x1s x1t x1u x1v x1w x1x x1y x1z x1aa x1ab x2ab NS US N0 U0 NEk UEk S*.
(*S*,
x1, *x1a*, *x1b*, *x1c*, *x1d*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *x1e*, *x2e*)
∈ *?T* ⟹
isasat-input-bounded (*all-init-atms x1a* (*x1c* + *NEk* + *NS* + *N0*))›
unfolding *twl-st-heur-restart-def*
by (*auto simp*: *all-init-atms-st-def*)
have [*refine0*]: ‹⋀*x1 x1a x1b x1c x1d x1e x2e x1f x1g x1h x1i x1j x1m x1n x1o x1p x2n x2o x1q x1r x1s x1t x1u x1v x1w x1x x1y x1z x1aa x1ab x2ab NS US N0 U0 NEk UEk S*.
(*S*,
x1, *x1a*, *x1b*, *x1c*, *x1d*, *NEk*, *UEk*,*NS*, *US*, *N0*, *U0*, *x1e*, *x2e*)

```
    ∈ ?T ⟹
    vdom-m (all-init-atms x1a (x1c+NEk+NS+N0)) x2e x1a ⊆ set (get-vdom S)›
  unfolding twl-st-heur-restart-def
  by auto
have [refine0]: ‹⋀x1 x1a x1b x1c x1d x1e x2e x1f x1g x1h x1i x1j x1m x1n x1o x1p x2n x2o x1q
    x1r x1s x1t x1u x1v x1w x1x x1y x1z x1aa x1ab x2ab NS US N0 U0 NEk UEk S.
    (S,
     x1, x1a, x1b, x1c, x1d, NEk, UEk,NS, US, N0, U0, x1e, x2e)
    ∈ ?T ⟹
    all-init-atms x1a (x1c+NEk+NS+N0) ≠ {#}›
  unfolding twl-st-heur-restart-def
  by auto
have [refine0]: ‹⋀a b c x1 x1a x1b x1c x1d x1e x2e x1f x1g x1h x1i x1j x1m x1n x1o x1p x2n x2o x1q
    x1r x1s x1t x1u x1v x1w x1x x1y x1z x1aa x1ab x2ab NS US N0 U0 NEk UEk S.
    (S, x1, x1a, x1b, x1c, x1d, NEk, UEk, NS, US, N0, U0, x1e, x2e) ∈ ?T ⟹
    get-vmtf-heur S ∈ bump-heur (all-init-atms x1a (x1c+NEk+NS+N0)) x1 ›
    ‹⋀x1 x1a x1b x1c x1d x1e x2e x1f x1g x1h x1i x1j x1m x1n x1o x1p x2n x2o x1q
    x1r x1s x1t x1u x1v x1w x1x x1y x1z x1aa x1ab x2ab NS US N0 U0 NEk UEk S.
    (S,
     x1, x1a, x1b, x1c, x1d, NEk, UEk, NS, US, N0, U0, x1e, x2e)
  ∈ ?T ⟹ (get-watched-wl-heur S, x2e) ∈ ⟨Id⟩map-fun-rel (D₀ (all-init-atms x1a (x1c+NEk+NS+N0)))›
  unfolding twl-st-heur-restart-def distinct-atoms-rel-def distinct-hash-atoms-rel-def
    all-init-atms-st-def
  by (case-tac ‹get-vmtf-heur S›; auto; fail)+
have H: ‹vdom-m (all-init-atms x1a x1c) x2ad x1ad ⊆ set x2af›
  if
    empty: ‹∀ A∈#all-init-atms x1a x1c. x2ad (Pos A) = [] ∧ x2ad (Neg A) = []› and
    rem: ‹GC-remap** (x1a, Map.empty, fmempty) (fmempty, m, x1ad)› and
    ‹dom-m x1ad = mset x2af›
  for m :: ‹nat ⇒ nat option› and y :: ‹nat literal multiset› and x :: ‹nat› and
    x1 x1a x1b x1c x1d x1e x2e x1f x1g x1h x1i x1j x1m x1n x1o x1p x2n x2o x1q
      x1r x1s x1t x1u x1v x1w x1x x1y x1z x1aa x1ab x2ab x1ac x1ad x2ad x1ae
      x1ag x2af x2ag
proof −
  have ‹xa ∈# 𝓛_all (all-init-atms x1a x1c) ⟹ x2ad xa = []› for xa
    using empty by (cases xa) (auto simp: in-𝓛_all-atm-of-𝒜_in)
  then show ?thesis
    using ‹dom-m x1ad = mset x2af›
    by (auto simp: vdom-m-def)
qed
have H': ‹mset x2ag ⊆# mset x1ah ⟹ x ∈ set x2ag ⟹ x ∈ set x1ah› for x2ag x1ah x
  by (auto dest: mset-eq-setD)
have [iff]: ‹ (∃ UEa :: 'v clauses. size UE = size UEa)› for UE :: ‹'v clauses›
  by auto
show ?thesis
  supply [[goals-limit=1]]
  unfolding isasat-GC-clauses-prog-wl-def cdcl-GC-clauses-prog-wl-alt-def take-0 Let-def
  apply (intro frefI nres-relI)
  apply (refine-vcg isasat-GC-clauses-prog-wl2[where 𝒜 = ‹all-init-atms - -›]; remove-dummy-vars)
  subgoal
    by (clarsimp simp add: twl-st-heur-restart-def
      cdcl-GC-clauses-prog-wl-inv-def H H'
      rtranclp-GC-remap-all-init-atms
      rtranclp-GC-remap-learned-clss-l)
  subgoal
    unfolding cdcl-GC-clauses-pre-wl-def mem-Collect-eq prod.simps
```

no-lost-clause-in-WL-alt-def
apply *normalize-goal+*
by (*drule length-watched-le'''*)
(*clarsimp-all simp add*: *twl-st-heur-restart-def*
cdcl-GC-clauses-prog-wl-inv-def H H'
rtranclp-GC-remap-all-init-atms all-init-atms-st-def
rtranclp-GC-remap-learned-clss-l
dest!:)
subgoal
by (*clarsimp simp add*: *twl-st-heur-restart-def twl-st-heur-restart-strong-aivdom-def clss-size-corr-restart-def*
cdcl-GC-clauses-prog-wl-inv-def H H' clss-size-def
rtranclp-GC-remap-all-init-atms learned-clss-count-def aivdom-inv-dec-def
rtranclp-GC-remap-learned-clss-l)
done
qed

definition *cdcl-remap-st* :: ‹'*v twl-st-wl* ⇒ '*v twl-st-wl nres*› **where**
‹*cdcl-remap-st* = (λ(*M*, *N*$_0$, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *WS*).
SPEC (λ(*M'*, *N'*, *D'*, *NE'*, *UE'*, *NEk'*, *UEk'*, *NS'*, *US'*, *N0'*, *U0'*, *Q'*, *WS'*).
(*M'*, *D'*, *NE'*, *UE'*, *NEk'*, *UEk'*, *NS'*, *US'*, *N0'*, *U0'*, *Q'*) = (*M*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*) ∧
(∃ *m*. *GC-remap*** (*N*$_0$, (λ-. *None*), *fmempty*) (*fmempty*, *m*, *N'*)) ∧
0 ∉# *dom-m N'*))›

lemma *cdcl-GC-clauses-wl-D-alt-def*:
‹*cdcl-GC-clauses-wl* = (λ*S*. *do* {
ASSERT(*cdcl-GC-clauses-pre-wl S*);
let b = *True*;
if b then do {
S ← *cdcl-remap-st S*;
S ← *rewatch-spec S*;
RETURN S
}
else remove-all-learned-subsumed-clauses-wl S})›
supply [[*goals-limit=1*]]
unfolding *cdcl-GC-clauses-wl-def*
by (*fastforce intro*!: *ext simp*: *RES-RES-RETURN-RES2 cdcl-remap-st-def RES-RES13-RETURN-RES-bound*
RES-RES11-RETURN-RES uncurry-def image-iff
RES-RETURN-RES-RES2 RES-RETURN-RES RES-RES2-RETURN-RES rewatch-spec-def
rewatch-spec-def remove-all-learned-subsumed-clauses-wl-def
literals-are-$\mathcal{L}_{in}$*'-empty blits-in-*$\mathcal{L}_{in}$*'-restart-wl-spec0'*
all-init-lits-alt-def[*symmetric*]
intro!: *bind-cong-nres intro*: *literals-are-*$\mathcal{L}_{in}$*'-empty*)

lemma *cdcl-GC-clauses-prog-wl2-st*:
assumes ‹(*T*, *S*) ∈ *state-wl-l None*›
‹*cdcl-GC-clauses-pre S* ∧
no-lost-clause-in-WL T ∧
literals-are-$\mathcal{L}_{in}$*' T*› **and**
‹*get-clauses-wl T* = *N*$_0$'›
shows
‹*cdcl-GC-clauses-prog-wl T* ≤
⇓ {((*M'*, *N''*, *D'*, *NE'*, *UE'*, *NEk'*, *UEk'*, *NS'*, *US'*, *N0'*, *U0'*, *Q'*, *WS'*), (*N*, *N'*)).
(*M'*, *D'*, *NE'*, *UE'*, *NEk'*, *UEk'*, *NS'*, *US'*, *N0'*, *U0'*, *Q'*) = (*get-trail-wl T*, *get-conflict-wl T*, *get-unkept-unit-init-clss-wl T*,
get-unkept-unit-learned-clss-wl T, *get-kept-unit-init-clss-wl T*,

get-kept-unit-learned-clss-wl T, *get-subsumed-init-clauses-wl T*, *get-subsumed-learned-clauses-wl T*,
get-init-clauses0-wl T, *get-learned-clauses0-wl T*,
literals-to-update-wl T) ∧ *N''* = *N* ∧
(∀ *L*∈#*all-init-lits-of-wl T*. *WS'* *L* = []) ∧
all-init-lits-of-wl T = *all-init-lits N* (*NE'*+*NEk'*+*NS'*+*N0'*) ∧
(∃ *m*. *GC-remap*** (*get-clauses-wl T*, *Map.empty*, *fmempty*)
(*fmempty*, *m*, *N*))}
(*SPEC*(λ(*N'*::(*nat*, *'a literal list* × *bool*) *fmap*, *m*).
*GC-remap*** (*N*$_0$', (λ-. *None*), *fmempty*) (*fmempty*, *m*, *N'*) ∧
0 ∉# *dom-m N'*))›
using *assms* **apply** −
apply (*rule order-trans*)
apply (*rule order-trans*)
prefer *2*
apply (*rule cdcl-GC-clauses-prog-wl2*[*of* ‹*get-trail-wl T*› ‹*get-clauses-wl T*› ‹*get-conflict-wl T*›
‹*get-unkept-unit-init-clss-wl T*› ‹*get-unkept-unit-learned-clss-wl T*›
‹*get-kept-unit-init-clss-wl T*› ‹*get-kept-unit-learned-clss-wl T*› ‹*get-subsumed-init-clauses-wl T*›
‹*get-subsumed-learned-clauses-wl T*› ‹*get-init-clauses0-wl T*› ‹*get-learned-clauses0-wl T*› ‹*literals-to-update-wl T*›
‹*get-watched-wl T*› *S* *N*$_0$'])
apply (*cases T*; *auto simp*: *all-init-atms-st-def*; *fail*)+
apply (*auto 5 3 simp*: *all-init-lits-of-wl-def all-init-lits-def ac-simps*
get-unit-init-clss-wl-alt-def
dest: *rtranclp-GC-remap-init-clss-l-old-new intro*!: *RES-refine*)
done

abbreviation *isasat-GC-clauses-rel* **where**
‹*isasat-GC-clauses-rel y u* ≡ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart-strong-aivdom* ∧
(∀ *L*∈#*all-init-lits-of-wl y*. *get-watched-wl T L* = [])∧
get-trail-wl T = *get-trail-wl y* ∧
get-conflict-wl T = *get-conflict-wl y* ∧
get-unkept-unit-init-clss-wl T = *get-unkept-unit-init-clss-wl y* ∧
get-kept-unit-init-clss-wl T = *get-kept-unit-init-clss-wl y* ∧
get-unkept-unit-learned-clss-wl T = *get-unkept-unit-learned-clss-wl y* ∧
get-kept-unit-learned-clss-wl T = *get-kept-unit-learned-clss-wl y* ∧
get-subsumed-init-clauses-wl T = *get-subsumed-init-clauses-wl y* ∧
get-subsumed-learned-clauses-wl T = *get-subsumed-learned-clauses-wl y* ∧
get-init-clauses0-wl T = *get-init-clauses0-wl y* ∧
get-learned-clauses0-wl T = *get-learned-clauses0-wl y* ∧
learned-clss-count S ≤ *u* ∧
(∃ *m*. *GC-remap*** (*get-clauses-wl y*, (λ-. *None*), *fmempty*) (*fmempty*, *m*, *get-clauses-wl T*)) ∧
arena-is-packed (*get-clauses-wl-heur S*) (*get-clauses-wl T*)}›

lemma *isasat-GC-clauses-prog-wl-cdcl-remap-st*:
assumes
‹(*x*, *y*) ∈ *twl-st-heur-restart'''u r u*› **and**
‹*cdcl-GC-clauses-pre-wl y*›
shows ‹*isasat-GC-clauses-prog-wl x* ≤ ⇓ (*isasat-GC-clauses-rel y u*) (*cdcl-remap-st y*)›
proof −
have *xy*: ‹(*x*, *y*) ∈ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart* ∧ *learned-clss-count S* ≤ *u*}›
using *assms*(*1*) **by** *auto*
have *H*: ‹*isasat-GC-clauses-rel y u*=
{(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart-strong-aivdom* ∧ *arena-is-packed* (*get-clauses-wl-heur S*) (*get-clauses-wl T*) ∧

```
        learned-clss-count S ≤ u} O
    {(S, T). S = T ∧ (∀ L∈#all-init-lits-of-wl y. get-watched-wl T L = [])∧
        get-trail-wl T = get-trail-wl y ∧
        get-conflict-wl T = get-conflict-wl y ∧
        get-unkept-unit-init-clss-wl T = get-unkept-unit-init-clss-wl y ∧
        get-kept-unit-init-clss-wl T = get-kept-unit-init-clss-wl y ∧
        get-unkept-unit-learned-clss-wl T = get-unkept-unit-learned-clss-wl y ∧
        get-kept-unit-learned-clss-wl T = get-kept-unit-learned-clss-wl y ∧
        get-subsumed-init-clauses-wl T = get-subsumed-init-clauses-wl y ∧
        get-subsumed-learned-clauses-wl T = get-subsumed-learned-clauses-wl y ∧
        get-init-clauses0-wl T = get-init-clauses0-wl y ∧
        get-learned-clauses0-wl T = get-learned-clauses0-wl y ∧
        (∃ m. GC-remap** (get-clauses-wl y, (λ-. None), fmempty) (fmempty, m, get-clauses-wl T))}›
    by blast
  show ?thesis
    using assms apply −
    apply (rule order-trans[OF isasat-GC-clauses-prog-wl[THEN fref-to-Down]])
    subgoal by fast
    apply (rule xy)
    unfolding conc-fun-chain[symmetric] H
    apply (rule ref-two-step')
    unfolding cdcl-GC-clauses-pre-wl-def
    apply normalize-goal+
    apply (rule order-trans[OF cdcl-GC-clauses-prog-wl2-st])
    apply assumption
    subgoal for xa
      using assms(2) by (simp add: correct-watching'-nobin-clauses-pointed-to
        cdcl-GC-clauses-pre-wl-def)
    apply (rule refl)
    subgoal by (auto simp: cdcl-remap-st-def conc-fun-RES split: prod.splits)
    done
qed

inductive-cases GC-remapE: ‹GC-remap (a, aa, b) (ab, ac, ba)›
lemma rtranclp-GC-remap-ran-m-remap:
  ‹GC-remap** (old, m, new) (old', m', new') ⟹ C ∈# dom-m old ⟹ C ∉# dom-m old' ⟹
      m' C ≠ None ∧
      fmlookup new' (the (m' C)) = fmlookup old C›
  apply (induction rule: rtranclp-induct[of r ‹(-, -, -)› ‹(-, -, -)›, split-format(complete), of for r])
  subgoal by auto
  subgoal for a aa b ab ac ba
    apply (cases ‹C ∉# dom-m a›)
    apply (auto dest: GC-remap-ran-m-remap GC-remap-ran-m-no-rewrite-map
      GC-remap-ran-m-no-rewrite)
   apply (metis GC-remap-ran-m-no-rewrite-fmap GC-remap-ran-m-no-rewrite-map in-dom-m-lookup-iff
option.sel)
    using GC-remap-ran-m-remap rtranclp-GC-remap-ran-m-no-rewrite by fastforce
  done

lemma GC-remap-ran-m-exists-earlier:
  ‹GC-remap (old, m, new) (old', m', new') ⟹ C ∈# dom-m new' ⟹ C ∉# dom-m new ⟹
      ∃ D. m' D = Some C ∧ D ∈# dom-m old ∧
      fmlookup new' C = fmlookup old D›
  by (induction rule: GC-remap.induct[split-format(complete)]) auto
```

lemma *rtranclp-GC-remap-ran-m-exists-earlier*:
  ‹*GC-remap*** (*old*, *m*, *new*) (*old'*, *m'*, *new'*) ⟹ *C* ∈# *dom-m new'* ⟹ *C* ∉# *dom-m new* ⟹
      ∃ *D*. *m' D* = *Some C* ∧ *D* ∈# *dom-m old* ∧
      fmlookup new' C = *fmlookup old D*›
  apply (*induction rule*: *rtranclp-induct*[*of r* ‹(-, -, -)› ‹(-, -, -)›, *split-format*(*complete*), *of* **for** *r*])
  apply (*auto dest*: *GC-remap-ran-m-exists-earlier*)
  apply (*case-tac* ‹*C* ∈# *dom-m b*›)
  apply (*auto elim*!: *GC-remapE split*: *if-splits*)
  apply *blast*
  using *rtranclp-GC-remap-ran-m-no-new-map rtranclp-GC-remap-ran-m-no-rewrite*
  by (*metis fst-conv*)

lemma *watchlist-put-binaries-first-one-correct-watching*:
  assumes ‹∃ *W'*. *correct-watching'''* *B* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, N_0, U_0, *Q*, *W'*) ∧ (*W*,*W'*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 *A*)› ‹*L* < *length W*›
  shows ‹*watchlist-put-binaries-first-one W L* ≤ ⇓{(*a*,*b*). (*a*,*b*)∈⟨*Id*⟩*map-fun-rel* (D_0 *A*) ∧ *length a* = *length W* ∧ (∀ *K*. *mset* (*a* ! *K*) = *mset* (*W* ! *K*))} (*SPEC* (λ*W*. *correct-watching'''* *B* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, N_0, U_0, *Q*, *W*)))›
proof −
  obtain *W'* **where** *W'*: ‹*correct-watching'''* *B*(*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, N_0, U_0, *Q*, *W'*) ∧ (*W*,*W'*) ∈ ⟨*Id*⟩*map-fun-rel* (D_0 *A*)›
    using *assms* **by** *fast*
  let *?R* = ‹*measure* (λ(*i*, *j*, -). *length* (*W* ! *L*) − *i*)›
  have *R*[*refine*]: ‹*wf ?R*›
    by *auto*
  have *H*: ‹(∀ *K*. *mset* (*Wa* ! *K*) = *mset* (*W* ! *K*)) ⟷ (∀ *K*. *mset* (*Wa* ! *K*) = *mset* (*W* ! *K*)) ∧ (∀ *K*. *set* (*Wa* ! *K*) = *set* (*W* ! *K*)) ∧
   (∀ *K*. *distinct-watched* (*Wa* ! *K*) ⟷ *distinct-watched* (*W* ! *K*))› **for** *W Wa*
   by (*auto dest*: *mset-eq-setD simp del*: *distinct-mset-mset-distinct*
    simp flip: *distinct-mset-mset-distinct*)

  show *?thesis*
    using *assms*(*2*) *W'*
    unfolding *watchlist-put-binaries-first-one-def conc-fun-SPEC* **apply** −
    apply (*refine-vcg*)
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by (*auto simp*: *nth-list-update*)
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal by *auto*
    subgoal for *s a b aa ba*
      apply (*subst* (*asm*) *H*)
      apply (*rule-tac x*= ‹λ*L*. *if L* ∈#$\mathcal{L}_{all}$ *A then ba* ! *nat-of-lit L else W' L*› **in** *exI*)
      apply (*auto simp*: *correct-watching'''.simps all-blits-are-in-problem-init.simps*
        map-fun-rel-def dest: *mset-eq-setD split*: *if-splits*)

```
    done
  done
qed

lemma watchlist-put-binaries-first:
  assumes ‹correct-watching''' 𝓑 (M, N, D, NE, UE, NEk, UEk, NS, US, N₀, U₀, Q, W')›
    ‹(W, W') ∈ ⟨Id⟩map-fun-rel (D₀ 𝓐)›
  shows ‹watchlist-put-binaries-first W ≤ ⇓(⟨Id⟩map-fun-rel (D₀ 𝓐)) (SPEC (λW. correct-watching'''
𝓑 (M, N, D, NE, UE, NEk, UEk, NS, US, N₀, U₀, Q, W)))›
proof −
  let ?R = ‹measure (λ(i, -). length W − i)›
  have [refine]: ‹wf ?R›
    by auto
  note [refine-vcg del] = WHILEIT-rule
  show ?thesis
    unfolding watchlist-put-binaries-first-def conc-fun-SPEC apply −
    apply (refine-vcg
        watchlist-put-binaries-first-one-correct-watching[where 𝓐=𝓐 and 𝓑=𝓑 and M=M and
          N=N and D=D and NE=NE and UE=UE and NEk=NEk and UEk=UEk and NS=NS
and US=US
        and N₀=N₀ and U₀=U₀ and Q=Q, THEN order-trans]
      WHILEIT-rule-stronger-inv[where
      I' = ‹λ(i, W). ∃ W'. (W, W')∈⟨Id⟩map-fun-rel (D₀ 𝓐)∧
        correct-watching''' 𝓑 (M, N, D, NE, UE, NEk, UEk, NS, US, N₀, U₀, Q, W')›])
    subgoal by auto
    subgoal by auto
    subgoal using assms by fast
    subgoal by auto
    subgoal by fast
    subgoal by (auto simp: conc-fun-RES)
    subgoal by fast
    done
qed

lemma rewatch-heur-st-correct-watching:
  assumes
    pre: ‹cdcl-GC-clauses-pre-wl y› and
    S-T: ‹(S, T) ∈ isasat-GC-clauses-rel y u›
  shows ‹rewatch-heur-and-reorder-st (empty-US-heur S) ≤ ⇓ (twl-st-heur-restart'''u (length (get-clauses-wl-heur
S)) u)
    (rewatch-spec (T))›
proof −
  obtain M N D NE UE NEk UEk NS US N0 U0 Q W where
    T: ‹T = (M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, Q, W)›
    by (cases T) auto
  let ?vdom = ‹get-aivdom S›
  let ?N' = ‹get-clauses-wl-heur S›
  let ?W' = ‹get-watched-wl-heur S›

  have
    valid: ‹valid-arena ?N' N (set (get-vdom-aivdom ?vdom))› and
    W: ‹(?W', W) ∈ ⟨Id⟩map-fun-rel (D₀ (all-init-atms-st T))› and
    empty: ‹⋀L. L ∈# all-init-lits-of-wl y ⟹ W L = []› and
    NUE:‹get-unkept-unit-init-clss-wl y = NE ›
      ‹get-unkept-unit-learned-clss-wl y = UE›
      ‹get-kept-unit-init-clss-wl y = NEk›
```

‹get-kept-unit-learned-clss-wl y = UEk›
‹get-trail-wl y = M›
‹get-subsumed-init-clauses-wl y = NS›
‹get-subsumed-learned-clauses-wl y = US›
‹get-init-clauses0-wl y = N0›
‹get-learned-clauses0-wl y = U0› **and**
avdom: ‹aivdom-inv-strong-dec (?vdom) (dom-m N)› **and**
tvdom: ‹get-tvdom-aivdom ?vdom = get-vdom-aivdom ?vdom›
using *assms* **by** (*auto simp*: *twl-st-heur-restart-strong-aivdom-def T all-init-atms-st-def*
aivdom-inv-strong-dec-alt-def)
have
dist: ‹distinct (get-vdom-aivdom ?vdom)› **and**
dom-m-vdom: ‹set-mset (dom-m N) ⊆ set (get-vdom-aivdom ?vdom)›
using *avdom valid* **unfolding** *aivdom-inv-strong-dec-alt-def*
apply (*cases ?vdom*; *cases ‹IsaSAT-VDom.get-aivdom ?vdom›*; *use avdom* **in** *auto*)
apply (*cases ?vdom*; *cases ‹IsaSAT-VDom.get-aivdom ?vdom›*; *use avdom valid* **in** ‹*auto simp*:
aivdom-inv-strong-dec-alt-def
simp del: *distinct-finite-set-mset-subseteq-iff*›)
by (*metis* (*no-types*, *opaque-lifting*) *UnE mset-subset-eqD set-mset-mset subsetD*)

obtain *m* **where**
m: ‹*GC-remap*** (*get-clauses-wl y*, *Map.empty*, *fmempty*)
(*fmempty*, *m*, *N*)›
using *assms* **by** (*auto simp*: *twl-st-heur-restart-def T*)
obtain *x xa xb* **where**
y-x: ‹(*y*, *x*) ∈ *Id*› ‹*x* = *y*› **and**
lits-y: ‹*literals-are-*$\mathcal{L}_{in}$' *y*› **and**
x-xa: ‹(*x*, *xa*) ∈ *state-wl-l None*› **and**
‹*no-lost-clause-in-WL x*› **and**
xa-xb: ‹(*xa*, *xb*) ∈ *twl-st-l None*› **and**
‹*twl-list-invs xa*› **and**
struct-invs: ‹*twl-struct-invs xb*› **and**
‹*get-conflict-l xa* = *None*› **and**
‹*clauses-to-update-l xa* = {#}› **and**
‹*count-decided* (*get-trail-l xa*) = *0*› **and**
‹∀ *L*∈*set* (*get-trail-l xa*). *mark-of L* = *0*›
using *pre*
unfolding *cdcl-GC-clauses-pre-wl-def*
cdcl-GC-clauses-pre-def
by *blast*
have [*iff*]:
‹*distinct-mset* (*mset* (*watched-l C*) + *mset* (*unwatched-l C*)) ⟷ *distinct C*› **for** *C*
unfolding *mset-append*[*symmetric*]
by *auto*

have ‹*twl-st-inv xb*›
using *xa-xb struct-invs*
by (*auto simp*: *twl-struct-invs-def*
cdcl$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*)
then have *A*:
‹⋀*C*. *C* ∈# *dom-m* (*get-clauses-wl x*) ⟹ *distinct* (*get-clauses-wl x* ∝ *C*) ∧ *2* ≤ *length* (*get-clauses-wl*
x ∝ *C*)›
using *xa-xb x-xa*
by (*cases x*; *cases ‹irred* (*get-clauses-wl x*) *C›*)
(*auto simp*: *twl-struct-invs-def twl-st-inv.simps*
twl-st-l-def state-wl-l-def ran-m-def conj-disj-distribR

Collect-disj-eq Collect-conv-if
dest!: *multi-member-split*
split: *if-splits*)
have *struct-wf*:
‹$C \in\# \mathit{dom\text{-}m}\ N \Longrightarrow \mathit{distinct}\ (N \propto C) \wedge 2 \leq \mathit{length}\ (N \propto C)$› **for** C
using *rtranclp-GC-remap-ran-m-exists-earlier*[*OF m*, *of* ‹C›] *A y-x*
by (*auto simp*: *T dest*:)

have *eq-UnD*: ‹$A = A' \cup A'' \Longrightarrow A' \subseteq A$› **for** $A\ A'\ A''$
by *blast*

have *eq3*: ‹*all-init-lits-of-wl y* = *all-init-lits N* (*NE*+*NEk*+*NS*+*N0*)›
using *rtranclp-GC-remap-init-clss-l-old-new*[*OF m*] *NUE*
by (*auto simp*: *all-init-lits-def all-init-lits-of-wl-def ac-simps*
get-unit-init-clss-wl-alt-def)
moreover have ‹*all-lits-st y* = *all-lits-st T*›
using *rtranclp-GC-remap-init-clss-l-old-new*[*OF m*] *rtranclp-GC-remap-learned-clss-l-old-new*[*OF m*]
NUE
apply (*cases y*)
apply (*auto simp*: *all-init-lits-def T NUE all-lits-def all-lits-st-def*)
by (*metis all-clss-l-ran-m all-lits-def*)
moreover have ‹*set-mset* (*all-init-lits-of-wl y*) = *set-mset* (*all-lits-st T*)›
by (*smt* ‹$\bigwedge$*thesis*. ($\bigwedge$*x xa xb*. ⟦(*y*, *x*) ∈ *Id*; *x* = *y*; *literals-are-*$\mathcal{L}_{in}$*'* *y*; (*x*, *xa*) ∈ *state-wl-l None*; *no-lost-clause-in-WL x*; (*xa*, *xb*) ∈ *twl-st-l None*; *twl-list-invs xa*; *twl-struct-invs xb*; *get-conflict-l xa* = *None*; *clauses-to-update-l xa* = {#}; *count-decided* (*get-trail-l xa*) = *0*; ∀ *L*∈*set* (*get-trail-l xa*). *mark-of L* = *0*⟧ ⟹ *thesis*) ⟹ *thesis*› *calculation*(*2*) *literals-are-*$\mathcal{L}_{in}$*'-literals-are-*$\mathcal{L}_{in}$*-iff*(*4*))
ultimately have *lits*: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* (*all-init-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*))
(*mset* ‘# *ran-mf N*)›
using *literals-are-*$\mathcal{L}_{in}$*'-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*OF x-xa xa-xb struct-invs*] *lits-y*
rtranclp-GC-remap-init-clss-l-old-new[*OF m*]
rtranclp-GC-remap-learned-clss-l-old-new[*OF m*]
unfolding *literals-are-in-*$\mathcal{L}_{in}$*-mm-def all-init-lits-alt-def*[*symmetric*] $\mathcal{L}_{all}$*-all-init-atms-all-init-lits*
all-init-atms-st-def
by (*auto simp*: *T all-lits-st-def all-lits-def all-lits-of-mm-union*)

have *eq*: ‹*set-mset* ($\mathcal{L}_{all}$ (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*))) = *set-mset* (*all-init-lits-of-wl y*)›
unfolding *eq3* $\mathcal{L}_{all}$*-all-init-atms-all-init-lits* **..**
then have *vd*: ‹*vdom-m* (*all-init-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*))
W N ⊆ *set-mset* (*dom-m N*)›
using *empty dom-m-vdom*
by (*auto simp*: *vdom-m-def all-init-atms-st-def*)
have ‹{#*i* ∈# *clause-to-update L* (*M*, *N*, *get-conflict-wl y*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, {#}, {#}).
i ∈# *dom-m N*#} =
{#*i* ∈# *clause-to-update L* (*M*, *N*, *get-conflict-wl y*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, {#}, {#}).
True#}› **for** *L*
by (*rule filter-mset-cong2*) (*auto simp*: *clause-to-update-def*)
then have *corr2*: ‹*correct-watching'''*
({#*mset* (*fst x*). *x* ∈# *init-clss-l* (*get-clauses-wl y*)#} + *NE*+*NEk* + *NS*+*N0*)
(*M*, *N*, *get-conflict-wl y*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W'a*) ⟹
correct-watching' (*M*, *N*, *get-conflict-wl y*, *NE*, *UE*,*NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W'a*)› **for** *W'a*
using *rtranclp-GC-remap-init-clss-l-old-new*[*OF m*]
by (*auto simp*: *correct-watching'''.simps correct-watching'.simps all-init-lits-of-wl-def*

ac-simps)
have *eq2*: ‹*all-init-lits* (*get-clauses-wl y*) (*NE*+*NEk*+*NS*+*N0*) = *all-init-lits N* (*NE*+*NEk*+*NS*+*N0*)›
using *rtranclp-GC-remap-init-clss-l-old-new*[*OF m*]
by (*auto simp*: *T all-init-lits-def NUE*
$\mathcal{L}_{all}$-*all-init-atms-all-init-lits*)
have ‹*i* ∈# *dom-m N* ⟹ *set* (*N* ∝ *i*) ⊆ *set-mset* (*all-init-lits N* (*NE*+*NEk*+*NS*+*N0*))› **for** *i*
using *lits* **by** (*auto dest*!: *multi-member-split split-list*
simp: *literals-are-in-*$\mathcal{L}_{in}$*-mm-def ran-m-def all-init-atms-st-def*
all-lits-of-mm-add-mset all-lits-of-m-add-mset
$\mathcal{L}_{all}$-*all-init-atms-all-init-lits*)
then have *blit2*: ‹*correct-watching'''*
({#*mset x*. *x* ∈# *init-clss-lf* (*get-clauses-wl y*)#} + *NE* + *NEk* + *NS* + *N0*)
(*M*, *N*, *get-conflict-wl y*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W'a*) ⟹
blits-in-$\mathcal{L}_{in}$*'* (*M*, *N*, *get-conflict-wl y*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W'a*)› **for** *W'a*
using *rtranclp-GC-remap-init-clss-l-old-new*[*OF m*]
unfolding *correct-watching'''.simps blits-in-*$\mathcal{L}_{in}$*'-def eq2*
$\mathcal{L}_{all}$-*all-init-atms-all-init-lits all-init-lits-alt-def*[*symmetric*] *all-init-lits-of-wl-def*
by (*fastforce simp add*: *all-init-lits-def blits-in-*$\mathcal{L}_{in}$*'-def ac-simps*
get-unit-init-clss-wl-alt-def NUE dest!: *multi-member-split*)
have ‹*correct-watching'''*
({#*mset x*. *x* ∈# *init-clss-lf* (*get-clauses-wl y*)#} + (*NE*+*NEk* + *NS*+*N0*))
(*M*, *N*, *get-conflict-wl y*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W'a*) ⟹
vdom-m (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) *W'a N* ⊆ *set-mset* (*dom-m N*)› **for** *W'a*
using *rtranclp-GC-remap-init-clss-l-old-new*[*OF m*]
unfolding *correct-watching'''.simps blits-in-*$\mathcal{L}_{in}$*'-def eq2*
$\mathcal{L}_{all}$-*all-init-atms-all-init-lits all-init-lits-alt-def*[*symmetric*]
by (*auto simp add*: *all-init-lits-def blits-in-*$\mathcal{L}_{in}$*'-def vdom-m-def NUE all-init-atms-def*
$\mathcal{L}_{all}$-*atm-of-all-lits-of-mm*
simp flip: *all-lits-st-alt-def*
dest!: *multi-member-split*)
then have *st*:
‹(*x*, *W'a*) ∈ ⟨*Id*⟩*map-fun-rel* (*D*$_0$ (*all-init-atms-st* (*M*, *N*, *get-conflict-wl y*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*))) ⟹
correct-watching'''
({#*mset x*. *x* ∈# *init-clss-lf* (*get-clauses-wl y*)#} + (*NE* + (*NEk* + (*NS* + *N0*))))
(*M*, *N*, *get-conflict-wl y*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W'a*) ⟹
(*set-watched-wl-heur x*
(*set-learned-count-wl-heur* (*clss-size-resetUS0* (*get-learned-count S*)) *S*),
M, *N*, *get-conflict-wl y*, *NE*, {#}, *NEk*, *UEk*, *NS*, {#}, *N0*, {#}, *Q*, *W'a*)
∈ *twl-st-heur-restart*› **for** *W'a m x*
using *S-T dom-m-vdom*
by (*auto simp*: *T twl-st-heur-restart-def all-init-atms-st-def y-x NUE ac-simps*
twl-st-heur-restart-strong-aivdom-def aivdom-inv-strong-dec-def2)
have *Su*: ‹*learned-clss-count S* ≤ *u*›
using *S-T* **by** *auto*
have *truc*: ‹*xa* ∈# *all-learned-lits-of-wl* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W'a*)
⟹
xa ∈# *all-init-lits-of-wl* (*M*, *N*, *D*, *NE*, {#}, *NEk*, *UEk*, *NS*, {#}, *N0*, {#}, *Q*, *W'a*)› **for** *W'a M D xa*
using *lits-y eq3 rtranclp-GC-remap-learned-clss-l*[*OF m*]
unfolding *literals-are-in-*$\mathcal{L}_{in}$*'-def all-init-lits-def NUE all-learned-lits-of-wl-def*
all-learned-lits-of-wl-def all-init-lits-of-wl-def
all-lits-of-mm-union all-init-lits-def $\mathcal{L}_{all}$-*all-init-atms-all-init-lits*
by (*auto simp*: *ac-simps all-lits-of-mm-union get-unit-init-clss-wl-alt-def*
NUE get-unit-learned-clss-wl-alt-def)

show *?thesis*
supply [[*goals-limit=1*]]
using *assms*
unfolding *rewatch-heur-st-def T empty-US-heur-def rewatch-heur-and-reorder-st-def rewatch-heur-and-reorder-def Let-def prod.case tvdom isasat-state-simp nres-monad3*
apply *clarify*
apply (*rule ASSERT-leI*)
subgoal by (*auto dest*!: *valid-arena-vdom-subset simp*: *twl-st-heur-restart-strong-aivdom-def aivdom-inv-strong-dec-alt-def*)
apply (*rule bind-refine-res*)
prefer *2*
apply (*rule order.trans*)
apply (*rule rewatch-heur-rewatch*[*OF valid - dist dom-m-vdom W*[*unfolded T*, *simplified*] *lits*])
apply (*solves simp*)
apply (*rule vd*)
apply (*rule order-trans*[*OF ref-two-step'*])
apply (*rule rewatch-correctness*[**where** *M*=*M* **and** *N*=*N* **and** *NE*=*NE* **and** *UE*=*UE* **and** *C*=*D* **and** *Q*=*Q* **and**
NS=*NS* **and** *US*=*US* **and** N_0=*N0* **and** U_0=*U0* **and** *UEk*=*UEk* **and** *NEk*=*NEk* **and** *W*=*W*])
apply (*rule empty*[*unfolded all-init-lits-of-wl-def*]; *assumption*)
apply (*rule struct-wf*; *assumption*)
subgoal using *lits eq2* **by** (*auto simp*: *literals-are-in-*$\mathcal{L}_{in}$*-mm-def all-init-atms-def all-init-lits-def* $\mathcal{L}_{all}$*-atm-of-all-lits-of-mm all-init-atms-st-def ac-simps get-unit-init-clss-wl-alt-def simp del*: *all-init-atms-def*[*symmetric*])
apply (*subst conc-fun-RES*)
apply (*rule order.refl*)
subgoal for *m x*
apply *clarify*
subgoal for *W'*
apply (*rule bind-refine-res*)
prefer *2*
apply (*rule order.trans*)
apply (*rule watchlist-put-binaries-first*[**where** *M*=*M* **and** *N*=*N* **and** *NE*=*NE* **and** *UE*=*UE* **and** *Q*=*Q* **and**
NS=*NS* **and** *US*=*US* **and** N_0=*N0* **and** U_0=*U0* **and** *UEk*=*UEk* **and** *NEk*=*NEk* **and** *D*=*D* **and** *W*= ‹*x*› **and** *W'*=*W'*
and $\mathcal{B}$ = ‹(*mset* '# *get-init-clss-wl y* + *get-unit-init-clss-wl y* +
get-subsumed-init-clauses-wl y +
get-init-clauses0-wl y)›])
apply *auto*[*2*]
apply (*subst conc-fun-RES*)
apply (*rule order.refl*)
apply *clarify*
using *Su*
apply (*auto simp*: *rewatch-spec-def RETURN-RES-refine-iff NUE learned-clss-count-def literals-are-in-*$\mathcal{L}_{in}$*-mm-def literals-are-*$\mathcal{L}_{in}$*'-def add.assoc get-unit-init-clss-wl-alt-def intro*: *corr2 blit2 st dest*: *truc intro*!:)
apply (*rule-tac x*=*xa* **in** *exI*)
apply (*auto simp*: *rewatch-spec-def RETURN-RES-refine-iff NUE learned-clss-count-def literals-are-in-*$\mathcal{L}_{in}$*-mm-def literals-are-*$\mathcal{L}_{in}$*'-def add.assoc get-unit-init-clss-wl-alt-def intro*: *corr2 blit2 st dest*: *truc intro*!:)
done
done
done
qed

lemma *GC-remap-dom-m-subset*:
‹*GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) ⟹ *dom-m old'* ⊆# *dom-m old*›
by (*induction rule*: *GC-remap.induct*[*split-format*(*complete*)]) (*auto dest*!: *multi-member-split*)

lemma *rtranclp-GC-remap-dom-m-subset*:
‹*rtranclp GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) ⟹ *dom-m old'* ⊆# *dom-m old*›
apply (*induction rule*: *rtranclp-induct*[*of r* ‹(-, -, -)› ‹(-, -, -)›, *split-format*(*complete*), *of* **for** *r*])
subgoal by *auto*
subgoal for *old1 m1 new1 old2 m2 new2*
using *GC-remap-dom-m-subset*[*of old1 m1 new1 old2 m2 new2*] **by** *auto*
done

lemma *GC-remap-mapping-unchanged*:
‹*GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) ⟹ *C* ∈ *dom m* ⟹ *m' C* = *m C*›
by (*induction rule*: *GC-remap.induct*[*split-format*(*complete*)]) *auto*

lemma *rtranclp-GC-remap-mapping-unchanged*:
‹*GC-remap*** (*old*, *m*, *new*) (*old'*, *m'*, *new'*) ⟹ *C* ∈ *dom m* ⟹ *m' C* = *m C*›
apply (*induction rule*: *rtranclp-induct*[*of r* ‹(-, -, -)› ‹(-, -, -)›, *split-format*(*complete*), *of* **for** *r*])
subgoal by *auto*
subgoal for *old1 m1 new1 old2 m2 new2*
using *GC-remap-mapping-unchanged*[*of old1 m1 new1 old2 m2 new2*, *of C*]
by (*auto dest*: *GC-remap-mapping-unchanged simp*: *dom-def intro*!: *image-mset-cong2*)
done

lemma *GC-remap-mapping-dom-extended*:
‹*GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) ⟹ *dom m'* = *dom m* ∪ *set-mset* (*dom-m old* − *dom-m old'*)›
by (*induction rule*: *GC-remap.induct*[*split-format*(*complete*)]) (*auto dest*!: *multi-member-split*)

lemma *rtranclp-GC-remap-mapping-dom-extended*:
‹*GC-remap*** (*old*, *m*, *new*) (*old'*, *m'*, *new'*) ⟹ *dom m'* = *dom m* ∪ *set-mset* (*dom-m old* − *dom-m old'*)›
apply (*induction rule*: *rtranclp-induct*[*of r* ‹(-, -, -)› ‹(-, -, -)›, *split-format*(*complete*), *of* **for** *r*])
subgoal by *auto*
subgoal for *old1 m1 new1 old2 m2 new2*
using *GC-remap-mapping-dom-extended*[*of old1 m1 new1 old2 m2 new2*]
GC-remap-dom-m-subset[*of old1 m1 new1 old2 m2 new2*]
rtranclp-GC-remap-dom-m-subset[*of old m new old1 m1 new1*]
by (*auto dest*: *GC-remap-mapping-dom-extended simp*: *dom-def mset-subset-eq-exists-conv*)
done

lemma *GC-remap-dom-m*:
‹*GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) ⟹ *dom-m new'* = *dom-m new* + *the* ‘# *m'* ‘# (*dom-m old* − *dom-m old'*)›
by (*induction rule*: *GC-remap.induct*[*split-format*(*complete*)]) (*auto dest*!: *multi-member-split*)

lemma *rtranclp-GC-remap-dom-m*:
‹*rtranclp GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) ⟹ *dom-m new'* = *dom-m new* + *the* ‘# *m'* ‘# (*dom-m old* − *dom-m old'*)›
apply (*induction rule*: *rtranclp-induct*[*of r* ‹(-, -, -)› ‹(-, -, -)›, *split-format*(*complete*), *of* **for** *r*])
subgoal by *auto*
subgoal for *old1 m1 new1 old2 m2 new2*
using *GC-remap-dom-m*[*of old1 m1 new1 old2 m2 new2*] *GC-remap-dom-m-subset*[*of old1 m1 new1 old2 m2 new2*]

```
    rtranclp-GC-remap-dom-m-subset[of old m new old1 m1 new1]
    GC-remap-mapping-unchanged[of old1 m1 new1 old2 m2 new2]
    rtranclp-GC-remap-mapping-dom-extended[of old m new old1 m1 new1]
    by (auto dest: simp: mset-subset-eq-exists-conv intro!: image-mset-cong2)
  done

lemma isasat-GC-clauses-rel-packed-le:
  assumes
    xy: ‹(x, y) ∈ twl-st-heur-restart''' r› and
    ST: ‹(S, T) ∈ isasat-GC-clauses-rel y u›
  shows ‹length (get-clauses-wl-heur S) ≤ length (get-clauses-wl-heur x)› and
     ‹∀ C ∈ set (get-tvdom S). C < length (get-clauses-wl-heur x)›
proof −
  obtain m where
    ‹(S, T) ∈ twl-st-heur-restart-strong-aivdom› and
    ‹∀ L∈#all-init-lits-of-wl y. get-watched-wl T L = []› and
    ‹get-trail-wl T = get-trail-wl y› and
    ‹get-conflict-wl T = get-conflict-wl y› and
    ‹get-kept-unit-init-clss-wl T = get-kept-unit-init-clss-wl y› and
    ‹get-kept-unit-learned-clss-wl T = get-kept-unit-learned-clss-wl y› and
    ‹get-unkept-unit-init-clss-wl T = get-unkept-unit-init-clss-wl y› and
    ‹get-unkept-unit-learned-clss-wl T = get-unkept-unit-learned-clss-wl y› and
    remap: ‹GC-remap** (get-clauses-wl y, Map.empty, fmempty)
      (fmempty, m, get-clauses-wl T)› and
    packed: ‹arena-is-packed (get-clauses-wl-heur S) (get-clauses-wl T)›
     using ST by auto
  have ‹valid-arena (get-clauses-wl-heur x) (get-clauses-wl y) (set (get-vdom x))›
    using xy unfolding twl-st-heur-restart-def by (cases x; cases y) auto
  from valid-arena-ge-length-clauses[OF this]
  have ‹(∑ C∈#dom-m (get-clauses-wl  y). length (get-clauses-wl y ∝ C) +
            header-size (get-clauses-wl y ∝ C)) ≤ length (get-clauses-wl-heur x)›
   (is ‹?A ≤ -›) .
  moreover have ‹?A = (∑ C∈#dom-m (get-clauses-wl T). length (get-clauses-wl T ∝ C) +
          header-size (get-clauses-wl T ∝ C))›
    using rtranclp-GC-remap-ran-m-remap[OF remap]
    by (auto simp: rtranclp-GC-remap-dom-m[OF remap] intro!: sum-mset-cong)
  ultimately show le: ‹length (get-clauses-wl-heur S) ≤ length (get-clauses-wl-heur x)›
    using packed unfolding arena-is-packed-def by simp

  have ‹valid-arena (get-clauses-wl-heur S) (get-clauses-wl T) (set (get-vdom S))›
    using ST unfolding twl-st-heur-restart-strong-aivdom-def by (cases S; cases T) auto
  moreover have ‹set (get-tvdom S) ⊆ set (get-vdom S)›
    using ST by (auto simp: twl-st-heur-restart-strong-aivdom-def
      aivdom-inv-strong-dec-alt-def)
  ultimately show ‹∀ C ∈ set (get-tvdom S). C < length (get-clauses-wl-heur x)›
    using le
    by (auto dest: valid-arena-in-vdom-le-arena)
qed

lemma isasat-GC-clauses-wl-D:
  ‹(isasat-GC-clauses-wl-D b, cdcl-GC-clauses-wl)
    ∈ twl-st-heur-restart'''u r u →f ⟨twl-st-heur-restart''''u r u⟩nres-rel›
  apply (intro frefI nres-relI)
  unfolding prod-rel-fst-snd-iff
    isasat-GC-clauses-wl-D-def cdcl-GC-clauses-wl-D-alt-def uncurry-def
  apply (refine-vcg isasat-GC-clauses-prog-wl-cdcl-remap-st[where r=r]
```

rewatch-heur-st-correct-watching)
subgoal unfolding *isasat-GC-clauses-pre-wl-D-def* **by** *blast*
subgoal by *fast*
apply *assumption*
subgoal by (*rule isasat-GC-clauses-rel-packed-le*) *fast+*
subgoal by (*rule isasat-GC-clauses-rel-packed-le*(*2*)) *fast+*
apply *assumption+*
subgoal by (*auto*)
subgoal by (*auto*)
done

end
theory *IsaSAT-Restart-Reduce-LLVM*
imports *IsaSAT-Restart-Reduce-Defs IsaSAT-Setup-LLVM IsaSAT-VMTF-State-LLVM*
begin

lemma *schedule-next-reduce-st-alt-def*:
‹*schedule-next-reduce-st b S* = (*let* (*heur*, *S*) = *extract-heur-wl-heur S*; *heur* = *schedule-next-reduce b heur in update-heur-wl-heur heur S*)›
by (*auto simp*: *schedule-next-reduce-st-def state-extractors Let-def intro*!: *ext split*: *isasat-int-splits*)

sepref-def *schedule-next-reduce-st-impl*
is ‹*uncurry* (*RETURN oo schedule-next-reduce-st*)›
:: ‹*word64-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
unfolding *schedule-next-reduce-st-alt-def*
by *sepref*

lemmas [*sepref-fr-rules*] = *irredandant-count.refine*
lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
get-irredundant-count-st-code-def[*unfolded read-all-st-code-def*]

lemma *schedule-next-reduction-stI*: ‹¬*a* < (*10* :: *64 word*) ⟹ *a* > *0*›
unfolding *word-le-not-less*[*symmetric*]
apply (*rule order.strict-trans2*)
prefer *2*
apply *assumption*
by *auto*

sepref-def *reduceint-impl*
is ‹*uncurry0* (*RETURN reduceint*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *word64-assn*›
unfolding *reduceint-def*
by *sepref*

lemma *schedule-next-reduction-stI2*:
‹*1* ≤ *a* ⟹ *0* < *a*› **for** *a* :: ‹*64 word*›
unfolding *word-le-not-less*[*symmetric*]
apply (*rule order.strict-trans2*)
prefer *2*
apply *assumption*
by *auto*

lemma *schedule-next-reduction-stI3*: ‹*word-log2 n div 2* < *64*› **for** *n* :: ‹*64 word*›
using *word-log2-max*[*of n*] **unfolding** *size-word-def*
by *auto*

sepref-def *schedule-next-reduction-st-impl*
  is ‹*RETURN o schedule-next-reduction-st*›
  :: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
  supply [[*goals-limit=1*]]
  supply [*simp*] = *schedule-next-reduction-stI schedule-next-reduction-stI3*
  supply [*intro*] = *schedule-next-reduction-stI2*
  supply [*split*] = *if-splits*
  supply *of-nat-snat*[*sepref-import-param*]
  unfolding *schedule-next-reduction-st-def max-def*
  apply (*rewrite* **in** ‹- >> ⌑› *snat-const-fold*[**where** *'a=64*])

  apply (*annot-unat-const* ‹*TYPE*(*64*)›)
  by *sepref*

definition *vmtf-array-nxt-score* :: ‹*vmtf* ⇒ -› **where** ‹*vmtf-array-nxt-score x* = *fst* (*snd x*)›

lemma ‹*current-vmtf-array-nxt-score x* = (*case x of Bump-Heuristics a b c d* ⇒
  (*vmtf-array-nxt-score b*))›
  by (*cases x*) (*auto simp*: *vmtf-array-nxt-score-def current-vmtf-array-nxt-score-def*
    bump-get-heuristics-def)

lemma *vmtf-array-nxt-score-alt-def*: ‹*RETURN o vmtf-array-nxt-score* = (λ(*a*,*b*,*c*,*d*,*e*) . *let b'* = *COPY b in RETURN b*)›
  by (*auto intro*!: *ext simp*: *vmtf-array-nxt-score-def*)

find-theorems *hn-refine PASS*
sepref-def *vmtf-array-nxt-score-code*
  is ‹*RETURN o vmtf-array-nxt-score*›
  :: ‹*vmtf-assn*^{k} →$_a$ *uint64-nat-assn*›
  unfolding *vmtf-array-nxt-score-alt-def vmtf-assn-def*
  by *sepref*

lemma *current-vmtf-array-nxt-score-alt-def*: ‹*RETURN o current-vmtf-array-nxt-score* = (λ*x. case x of Bump-Heuristics hstable focused foc a* ⇒
    RETURN (*vmtf-array-nxt-score focused*))›
   by (*auto intro*!: *ext simp*: *bump-get-heuristics-def current-vmtf-array-nxt-score-def vmtf-array-nxt-score-def*
      split: *bump-heuristics-splits*)

sepref-def *current-vmtf-array-nxt-score-code*
  is ‹*RETURN o current-vmtf-array-nxt-score*›
  :: ‹*heuristic-bump-assn*^{k} →$_a$ *uint64-nat-assn*›
  unfolding *current-vmtf-array-nxt-score-alt-def*
  by *sepref*

sepref-def *find-local-restart-target-level-fast-code*
  is ‹*uncurry find-local-restart-target-level-int*›
  :: ‹*trail-pol-fast-assn*^{k} *$_a$ *heuristic-bump-assn*^{k} →$_a$ *uint32-nat-assn*›
  supply [[*goals-limit=1*]] *length-rev*[*simp del*]
  unfolding *find-local-restart-target-level-int-def find-local-restart-target-level-int-inv-def*
    length-uint32-nat-def trail-pol-fast-assn-def
  apply (*annot-unat-const* ‹*TYPE*(*32*)›)
   apply (*rewrite* **in** ‹(- ! -)› *annot-unat-snat-upcast*[**where** *'l=64*])
   apply (*rewrite* **in** ‹(- ! ⌑)› *annot-unat-snat-upcast*[**where** *'l=64*])
   apply (*rewrite* **in** ‹(⌑ < *length* -)› *annot-unat-snat-upcast*[**where** *'l=64*])

by *sepref*

definition *find-local-restart-target-level-st-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*find-local-restart-target-level-st-fast-code* = (*read-all-st-code* (λ*M* - - - - *N* - - - - - - - - - - - -. *find-local-restart-target-level-fast-code M N*))›

global-interpretation *find-restart-lvl*: *read-trail-vmtf-param-adder0* **where**
P = ‹λ- -. *True*› **and**
f' = ‹*find-local-restart-target-level-int*› **and**
f = ‹*find-local-restart-target-level-fast-code*› **and**
x-assn = ‹*uint32-nat-assn*›
rewrites
‹(*read-all-st* (λ*M* - - - - *N* - - - - - - - - - - -. *find-local-restart-target-level-int M N*)) = *find-local-restart-target-level-st*› **and**
‹(*read-all-st-code* (λ*M* - - - - *N* - - - - - - - - - - -. *find-local-restart-target-level-fast-code M N*)) = *find-local-restart-target-level-st-fast-code*›
apply *unfold-locales*
apply (*subst lambda-comp-true*)
apply (*rule find-local-restart-target-level-fast-code.refine*)
subgoal by (*auto simp*: *read-all-st-def find-local-restart-target-level-st-def intro*!: *ext split*: *isasat-int-splits*)
subgoal by (*auto simp*: *find-local-restart-target-level-st-fast-code-def*)
done

lemmas [*sepref-fr-rules*] = *find-restart-lvl.refine*
lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] = *find-local-restart-target-level-st-fast-code-def*[*unfolded read-all-st-code-def*]

lemma *empty-Q-alt-def*:
‹*empty-Q* = (λ*S*. *do*{
let (*M*, *S*) = *extract-trail-wl-heur S*;
let (*heur*, *S*) = *extract-heur-wl-heur S*;
j ← *mop-isa-length-trail M*;
RETURN (*update-heur-wl-heur* (*restart-info-restart-done-heur heur*) (*update-literals-to-update-wl-heur j* (*update-trail-wl-heur M S*)))
})›
by (*auto simp*: *state-extractors empty-Q-def intro*!: *ext split*: *isasat-int-splits*)

sepref-def *empty-Q-fast-code*
is ‹*empty-Q*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *empty-Q-alt-def*
by *sepref*

sepref-register *cdcl-twl-local-restart-wl-D-heur empty-Q find-decomp-wl-st-int*

sepref-def *cdcl-twl-local-restart-wl-D-heur-fast-code*
is ‹*cdcl-twl-local-restart-wl-D-heur*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
unfolding *cdcl-twl-local-restart-wl-D-heur-def PR-CONST-def*
supply [[*goals-limit* = *1*]]
by *sepref*

definition *lbd-sort-clauses-raw* :: ‹*arena* ⇒ *vdom* ⇒ *nat* ⇒ *nat* ⇒ *nat list nres*› **where**
‹*lbd-sort-clauses-raw arena N* = *pslice-sort-spec idx-cdom clause-score-less arena N*›

definition *lbd-sort-clauses-avdom* :: ‹*arena* ⇒ *vdom* ⇒ *nat list nres*› **where**
‹*lbd-sort-clauses-avdom arena N* = *lbd-sort-clauses-raw arena N 0* (*length N*)›

lemmas *LBD-introsort*[*sepref-fr-rules*] =
LBD-it.introsort-param-impl-correct[*unfolded lbd-sort-clauses-raw-def*[*symmetric*] *PR-CONST-def*]

sepref-register *lbd-sort-clauses-raw*
sepref-def *lbd-sort-clauses-avdom-impl*
is ‹*uncurry lbd-sort-clauses-avdom*›
:: ‹*arena-fast-assn*^{k} *∗*$_a$ *vdom-fast-assn*^{d} *→*$_a$ *vdom-fast-assn*›
supply[[*goals-limit=1*]]
unfolding *lbd-sort-clauses-avdom-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-register *remove-deleted-clauses-from-avdom arena-status DELETED*
lemma *mark-to-delete-clauses-wl-D-heur-is-Some-iff*:
‹*D* = *Some C* ⟷ *D* ≠ *None* ∧ ((*the D*) = *C*)›
by *auto*

sepref-def *mop-marked-as-used-impl*
is ‹*uncurry mop-marked-as-used*›
:: ‹*arena-fast-assn*^{k} *∗*$_a$ *sint64-nat-assn*^{k} *→*$_a$ *unat-assn*$'$ *TYPE*(*2*)›
supply [[*goals-limit=1*]]
unfolding *mop-marked-as-used-def*
by *sepref*

sepref-def *MINIMUM-DELETION-LBD-impl*
is ‹*uncurry0* (*RETURN MINIMUM-DELETION-LBD*)›
:: ‹*unit-assn*^{k} *→*$_a$ *uint32-nat-assn*›
unfolding *MINIMUM-DELETION-LBD-def*
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

sepref-def *isa-is-candidate-for-removal-impl*
is ‹*uncurry2 isa-is-candidate-for-removal*›
:: ‹*trail-pol-fast-assn*^{k} *∗*$_a$ *sint64-nat-assn*^{k} *∗*$_a$ *arena-fast-assn*^{k} *→*$_a$ *bool1-assn*›
unfolding *isa-is-candidate-for-removal-def*
unfolding
access-avdom-at-def[*symmetric*] *length-avdom-def*[*symmetric*]
get-the-propagation-reason-heur-def[*symmetric*]
clause-is-learned-heur-def[*symmetric*]
clause-lbd-heur-def[*symmetric*]
access-length-heur-def[*symmetric*]
mark-to-delete-clauses-wl-D-heur-is-Some-iff
marked-as-used-st-def[*symmetric*] *if-conn*(*4*)
fold-tuple-optimizations
supply [[*goals-limit* = *1*]] *of-nat-snat*[*sepref-import-param*]
length-avdom-def[*symmetric*, *simp*] *access-avdom-at-def*[*simp*]
apply (*rewrite* **in** ‹*let* - = ¤ *in* -› *short-circuit-conv*)+
apply (*rewrite* **in** ‹- = *0*› *unat-const-fold*[**where** *'a=2*])
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-register *isa-is-candidate-for-removal*

sepref-def *remove-deleted-clauses-from-avdom-fast-code*
is ‹*uncurry2 isa-gather-candidates-for-reduction*›
:: ‹[λ((*M*, *N*), *vdom*). *length* (*get-vdom-aivdom vdom*) ≤ *snat64-max*]$_a$
trail-pol-fast-assn^k *$*_a$ *arena-fast-assn*^{d} *$*_a$ *aivdom-assn*^{d} →
arena-fast-assn ×$_a$ *aivdom-assn*›
supply [[*goals-limit=1*]]
supply [*simp*] = *length-avdom-aivdom-def*
unfolding *isa-gather-candidates-for-reduction-def*
convert-swap gen-swap if-conn(4) length-avdom-aivdom-def[*symmetric*]
avdom-aivdom-at-def[*symmetric*]
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

definition *lbd-sort-clauses* :: ‹*arena* ⇒ *aivdom2* ⇒ *aivdom2 nres*› **where**
‹*lbd-sort-clauses arena N* = *map-tvdom-aivdom-int* (*lbd-sort-clauses-avdom arena*) *N*›

sepref-def *lbd-sort-clauses-impl*
is ‹*uncurry lbd-sort-clauses*›
:: ‹[λ(*N*, *vdom*). *length* (*fst vdom*) ≤ *snat64-max*]$_a$ *arena-fast-assn*^{k} *$*_a$ *aivdom-int-assn*^{d} → *aivdom-int-assn*›
unfolding *lbd-sort-clauses-def map-tvdom-aivdom-int-def*
by *sepref*

lemma
map-vdom-aivdom-int2:
‹(*uncurry* (λ*arena*. *map-vdom-aivdom-int* (*f arena*)), *uncurry* (λ*arena*. *map-vdom-aivdom* (*f arena*)))
∈ *Id* ×$_r$ *aivdom-rel* →$_f$ ⟨*aivdom-rel*⟩*nres-rel*›
apply (*intro frefI nres-relI*)
subgoal for *x y*
using *map-vdom-aivdom-int*[*of* ‹*f* (*fst x*)›]
apply (*cases x*; *cases y*)
apply (*auto intro*!: *frefI nres-relI simp*: *fref-def nres-rel-def*)
done
done

lemma *get-aivdom-eq-aivdom-iff*:
‹*IsaSAT-VDom.get-aivdom b* = (*x1*, *a*, *aa*, *ba*) ⟷ *b* = *AIvdom* (*x1*, *a*, *aa*, *ba*)›
by (*cases b*) *auto*

lemma *quicksort-clauses-by-score-sort*:
‹(*uncurry lbd-sort-clauses*, *uncurry sort-clauses-by-score*) ∈
Id ×$_r$ *aivdom-rel* →$_f$ ⟨*aivdom-rel*⟩*nres-rel*›
apply (*intro fun-relI nres-relI frefI*)
subgoal for *arena arena′*
unfolding *uncurry-def lbd-sort-clauses-def map-tvdom-aivdom-int-def*
lbd-sort-clauses-avdom-def lbd-sort-clauses-raw-def sort-clauses-by-score-def
apply (*refine-vcg*)
apply (*rule specify-left*)
apply (*auto simp*: *lbd-sort-clauses-def lbd-sort-clauses-raw-def*
pslice-sort-spec-def le-ASSERT-iff idx-cdom-def slice-rel-def br-def uncurry-def
conc-fun-RES sort-spec-def map-vdom-aivdom-int-def lbd-sort-clauses-avdom-def
code-hider-rel-def

```
    split:prod.splits
    intro!: ASSERT-leI
    )
  apply (case-tac x2; auto)

  apply (rule order-trans)
  apply (rule slice-sort-spec-refine-sort)
  apply (auto simp: lbd-sort-clauses-def lbd-sort-clauses-raw-def
    pslice-sort-spec-def le-ASSERT-iff idx-cdom-def slice-rel-def br-def uncurry-def
    conc-fun-RES sort-spec-def map-vdom-aivdom-int-def lbd-sort-clauses-avdom-def
    code-hider-rel-def
    split:prod.splits
    intro!: ASSERT-leI
    )
  apply (case-tac x2; auto simp: get-aivdom-eq-aivdom-iff)
  apply (rule-tac x = ‹AIvdom (x1a, ac, ad, x)› in exI)
  apply auto
  by (metis slice-complete)
 done

context
 notes [fcomp-norm-unfold] = aivdom-assn-alt-def[symmetric] aivdom-assn-def[symmetric]
begin

lemma lbd-sort-clauses-impl-lbd-sort-clauses[sepref-fr-rules]:
 ‹(uncurry lbd-sort-clauses-impl, uncurry sort-clauses-by-score)
 ∈ [λ(N, vdom). length (get-avdom-aivdom vdom) ≤ snat64-max]_a (al-assn arena-el-impl-assn)^k *_a
aivdom-assn^d → aivdom-assn›
 (is ‹?c ∈ [?pre]_a ?im → ?f›)
proof −
 have H: ‹?c
∈ [comp-PRE (Id ×_f aivdom-rel) (λ-. True) (λx y. case y of (N, vdom) ⇒ length (fst vdom) ≤
snat64-max)
  (λx. nofail (uncurry sort-clauses-by-score x))]_a ?im → ?f›
   (is ‹- ∈ [?pre']_a ?im' → -›)
   using hfref-compI-PRE[OF lbd-sort-clauses-impl.refine,
 OF quicksort-clauses-by-score-sort, unfolded fcomp-norm-unfold] by blast
 have pre: ‹?pre' x› if ‹?pre x› for x
   using that
   by (case-tac x, case-tac ‹snd x›)
    (auto simp: comp-PRE-def code-hider-rel-def)
 show ?thesis
   apply (rule hfref-weaken-pre[OF ])
    defer
   using H apply assumption
   using pre ..
qed

end

lemma sort-vdom-heur-alt-def:
 ‹sort-vdom-heur = (λS_0. do {
   let (vdom, S) = extract-vdom-wl-heur S_0;
   ASSERT (vdom = get-aivdom S_0);
   let (M', S) = extract-trail-wl-heur S;
   ASSERT (M' = get-trail-wl-heur S_0);
```

let (*arena*, *S*) = *extract-arena-wl-heur S*;
ASSERT (*arena* = *get-clauses-wl-heur* S_0);
ASSERT(*length* (*get-avdom-aivdom vdom*) ≤ *length arena*);
ASSERT(*length* (*get-vdom-aivdom vdom*) ≤ *length arena*);
(*arena'*, *vdom*) ← *isa-gather-candidates-for-reduction M' arena vdom*;
ASSERT(*valid-sort-clause-score-pre arena* (*get-vdom-aivdom vdom*));
ASSERT(*EQ* (*length arena*) (*length arena'*));
ASSERT(*length* (*get-avdom-aivdom vdom*) ≤ *length arena*);
vdom ← *sort-clauses-by-score arena' vdom*;
RETURN (*update-arena-wl-heur arena'* (*update-vdom-wl-heur vdom* (*update-trail-wl-heur M' S*)))
})›
by (*auto intro*!: *ext split*: *isasat-int-splits simp*: *sort-vdom-heur-def state-extractors*)

sepref-def *sort-vdom-heur-fast-code*
is ‹*sort-vdom-heur*›
:: ‹[λ*S*. *length* (*get-clauses-wl-heur S*) ≤ *snat64-max*]$_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *sort-vdom-heur-alt-def EQ-def*
by *sepref*

sepref-def *find-largest-lbd-and-size-impl*
is ‹*uncurry find-largest-lbd-and-size*›
:: ‹*sint64-nat-assn*^{k} *$*_a$ *isasat-bounded-assn*^{k} →$_a$ *uint32-nat-assn* ×$_a$ *sint64-nat-assn*›
supply [*simp*] = *length-tvdom-def*[*symmetric*]
supply [*dest*] = *isasat-bounded-assn-length-arenaD*
unfolding *find-largest-lbd-and-size-def access-tvdom-at-def*[*symmetric*]
length-tvdom-def[*symmetric*] *max-def*
apply (*rewrite at* ‹(-, -, ⌑)› *snat-const-fold*[**where** *'a=64*])
apply (*rewrite at* ‹(⌑, -)› *snat-const-fold*[**where** *'a=64*])
apply (*rewrite at* ‹(-, ⌑, -)› *unat-const-fold*[**where** *'a=32*])
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

experiment
begin
export-llvm *sort-vdom-heur-fast-code remove-deleted-clauses-from-avdom-fast-code*
end

end
theory *IsaSAT-Simplify-Units-Defs*
imports *IsaSAT-Setup*
Watched-Literals.Watched-Literals-Watch-List-Inprocessing
More-Refinement-Libs.WB-More-Refinement-Loops
IsaSAT-Proofs
begin

definition *simplify-clause-with-unit2-pre* **where**
‹*simplify-clause-with-unit2-pre C M N* ⟷
C ∈# *dom-m N* ∧ *no-dup M*›

definition *simplify-clause-with-unit2* **where**
‹*simplify-clause-with-unit2 C M* N_0 = *do* {
ASSERT(*C* ∈# *dom-m* N_0);
let l = *length* (N_0 ∝ *C*);
(*i*, *j*, *N*, *del*, *is-true*) ← *WHILE*$_T$(λ(*i*, *j*, *N*, *del*, *b*). *C* ∈# *dom-m N*)
(λ(*i*, *j*, *N*, *del*, *b*). ¬*b* ∧ *j* < *l*)

```
    (λ(i, j, N, del, is-true). do {
      ASSERT(i < length (N ∝ C) ∧ j < length (N ∝ C) ∧ C ∈# dom-m N ∧ i ≤ j);
      let L = N ∝ C ! j;
      ASSERT(L ∈ set (N₀ ∝ C));
      let val = polarity M L;
      if val = Some True then RETURN (i, j+1, N, add-mset L del, True)
      else if val = Some False
      then RETURN (i, j+1, N, add-mset L del, False)
      else RETURN (i+1, j+1, N(C ↪ ((N ∝ C)[i := L])), del, False)
    })
     (0, 0, N₀, {#}, False);
  ASSERT(C ∈# dom-m N ∧ i ≤ length (N ∝ C));
  ASSERT(is-true ∨ j = l);
  let L = N ∝ C ! 0;
  if is-true ∨ i ≤ 1
  then RETURN (False, fmdrop C N, L, is-true, i)
  else if i = j ∧ ¬is-true then RETURN (True, N, L, is-true, i)
  else do {
    RETURN (False, N(C ↪ (take i (N ∝ C))), L, is-true, i)
  }
 }›

definition simplify-clause-with-unit-st2 :: ‹nat ⇒ nat twl-st-wl ⇒ nat twl-st-wl nres› where
 ‹simplify-clause-with-unit-st2 = (λC (M, N₀, D, NE, UE, NEk, UEk, NS, US, N0, U0, Q, W). do
{
  ASSERT(simplify-clause-with-unit-st-wl-pre C (M, N₀, D, NE, UE, NEk, UEk, NS, US, N0, U0,
Q, W));
  ASSERT (C ∈# dom-m N₀ ∧ count-decided M = 0 ∧ D = None);
  let S = (M, N₀, D, NE, UE, NEk, UEk, NS, US, N0, U0, Q, W);
  let E = mset (N₀ ∝ C);
  let irr = irred N₀ C;
  (unc, N, L, b, i) ← simplify-clause-with-unit2 C M N₀;
  ASSERT(dom-m N ⊆# dom-m N₀);
    if unc then do {
    ASSERT(N = N₀);
    let T = (M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, Q, W);
    RETURN T
  }
  else if b then do {
      let T = (M, N, D, (if irr then add-mset E else id) NE, (if ¬irr then add-mset E else id) UE,
NEk, UEk, NS, US, N0, U0, Q, W);
    ASSERT (set-mset (all-learned-lits-of-wl T) = set-mset (all-learned-lits-of-wl S));
    ASSERT (set-mset (all-init-lits-of-wl T) = set-mset (all-init-lits-of-wl S));
    ASSERT (set-mset (all-atms-st T) = set-mset (all-atms-st S));
    ASSERT (size (learned-clss-lf N) = size (learned-clss-lf N₀) − (if irr then 0 else 1));
    ASSERT(¬irr ⟶ size (learned-clss-lf N₀) ≥ 1);
    RETURN T
  }
  else if i = 1
  then do {
    ASSERT (undefined-lit M L ∧ L ∈# ℒ_all (all-atms-st S));
    M ← cons-trail-propagate-l L 0 M;
    let T = (M, N, D, NE, UE, (if irr then add-mset {#L#} else id) NEk, (if ¬irr then add-mset
```

{#L#} else id) UEk, (if irr then add-mset E else id) NS, (if ¬irr then add-mset E else id) US, N0, U0, add-mset (−L) Q, W);
ASSERT (set-mset (all-learned-lits-of-wl T) = set-mset (all-learned-lits-of-wl S));
ASSERT (set-mset (all-init-lits-of-wl T) = set-mset (all-init-lits-of-wl S));
ASSERT (set-mset (all-atms-st T) = set-mset (all-atms-st S));
ASSERT (size (learned-clss-lf N) = size (learned-clss-lf N_0) − (if irr then 0 else 1));
ASSERT(¬irr ⟶ size (learned-clss-lf N_0) ≥ 1);
RETURN T
}
else if i = 0
then do {
let j = length M;
let T = (M, N, Some {#}, NE, UE, NEk, UEk, (if irr then add-mset E else id) NS, (if ¬irr then add-mset E else id) US, (if irr then add-mset {#} else id) N0, (if ¬irr then add-mset {#} else id) U0, {#}, W);
ASSERT (set-mset (all-learned-lits-of-wl T) = set-mset (all-learned-lits-of-wl S));
ASSERT (set-mset (all-init-lits-of-wl T) = set-mset (all-init-lits-of-wl S));
ASSERT (set-mset (all-atms-st T) = set-mset (all-atms-st S));
ASSERT (size (learned-clss-lf N) = size (learned-clss-lf N_0) − (if irr then 0 else 1));
ASSERT(¬irr ⟶ size (learned-clss-lf N_0) ≥ 1);
RETURN T
}
else do {
let T = (M, N, D, NE, UE, NEk, UEk, (if irr then add-mset E else id) NS, (if ¬irr then add-mset E else id) US, N0, U0, Q, W);
ASSERT (set-mset (all-learned-lits-of-wl T) = set-mset (all-learned-lits-of-wl S));
ASSERT (set-mset (all-init-lits-of-wl T) = set-mset (all-init-lits-of-wl S));
ASSERT (set-mset (all-atms-st T) = set-mset (all-atms-st S));
ASSERT (size (learned-clss-lf N) = size (learned-clss-lf N_0));
ASSERT (C ∈# dom-m N);
RETURN T
}
})›

definition *simplify-clauses-with-unit-st2* :: ‹nat twl-st-wl ⇒ nat twl-st-wl nres› **where**
‹simplify-clauses-with-unit-st2 S =
do {
ASSERT (simplify-clauses-with-unit-st-wl-pre S);
xs ← SPEC(λxs. finite xs);
T ← FOREACHci(λit T. simplify-clauses-with-unit-st-wl-inv S it T)
(xs)
(λS. get-conflict-wl S = None)
(λi S. if i ∈# dom-m (get-clauses-wl S)
then simplify-clause-with-unit-st2 i S else RETURN S)
S;
ASSERT(set-mset (all-learned-lits-of-wl T) ⊆ set-mset (all-learned-lits-of-wl S));
ASSERT(set-mset (all-init-lits-of-wl T) = set-mset (all-init-lits-of-wl S));
RETURN T
}›

definition *isa-simplify-clause-with-unit2* **where**
‹isa-simplify-clause-with-unit2 C M N = do {
l ← mop-arena-length N C;
ASSERT(l < length N ∧ l ≤ Suc (unat32-max div 2));
(i, j, N::arena, is-true) ← WHILE$_T$(λ(i, j, N::arena, b). ¬b ∧ j < l)
(λ(i, j, N, is-true). do {

```
    ASSERT(i ≤ j ∧ j < l);
    L ← mop-arena-lit2 N C j;
    val ← mop-polarity-pol M L;
    if val = Some True then RETURN (i, j+1, N,True)
    else if val = Some False
    then RETURN (i, j+1, N,  False)
      else do {
      N ← mop-arena-update-lit C i L N;
      RETURN (i+1, j+1, N, False)}
  })
    (0, 0, N, False);
  L ← mop-arena-lit2 N C 0;
  if is-true ∨ i ≤ 1
  then do {
   ASSERT(mark-garbage-pre (N, C));
   RETURN (False, extra-information-mark-to-delete N C, L, is-true, i)}
  else if i = j then RETURN (True, N, L, is-true, i)
  else do {
    N ← mop-arena-shorten C i N;
    RETURN (False, N, L, is-true, i)}
  }›
```

definition *set-conflict-to-false* :: ‹*conflict-option-rel* ⇒ *conflict-option-rel*› **where**
‹*set-conflict-to-false* = (λ(*b*, *n*, *xs*). (*False*, *0*, *xs*))›

We butcher our statistics here, but the clauses are deleted later anyway.

definition *isa-simplify-clause-with-unit-st2* :: ‹*nat* ⇒ *isasat* ⇒ *isasat nres*› **where**

```
‹isa-simplify-clause-with-unit-st2 =  (λC S. do {
  let lcount = get-learned-count S; let N = get-clauses-wl-heur S; let M = get-trail-wl-heur S;
  E ← mop-arena-status N C;
  ASSERT(E = LEARNED ⟶ 1 ≤ clss-size-lcount lcount);
  (unc, N, L, b, i) ← isa-simplify-clause-with-unit2 C M N;
  ASSERT (length N ≤ length (get-clauses-wl-heur S));
  if unc then RETURN (set-clauses-wl-heur N S)
  else if b then
  RETURN  (set-clauses-wl-heur N
   (set-stats-wl-heur (if E=LEARNED then (get-stats-heur S) else (decr-irred-clss (get-stats-heur S)))
   (set-learned-count-wl-heur (if E = LEARNED then clss-size-decr-lcount (lcount) else lcount)
   S)))
  else if i = 1
  then do {
    M ← cons-trail-Propagated-tr L 0 M;
    RETURN (set-clauses-wl-heur N
    (set-trail-wl-heur M
    (set-stats-wl-heur (if E=LEARNED then incr-uset (get-stats-heur S) else incr-uset (decr-irred-clss
(get-stats-heur S)))
     (set-learned-count-wl-heur (if E = LEARNED then clss-size-decr-lcount (clss-size-incr-lcountUEk
lcount) else lcount)
    S)))) }
  else if i = 0
  then do {
    j ← mop-isa-length-trail M;
    RETURN (set-clauses-wl-heur N
    (set-conflict-wl-heur (set-conflict-to-false (get-conflict-wl-heur S))
    (set-count-max-wl-heur 0
    (set-literals-to-update-wl-heur j
```

```
    (set-stats-wl-heur (if E=LEARNED then get-stats-heur S else decr-irred-clss (get-stats-heur S))
    (set-learned-count-wl-heur (if E = LEARNED then clss-size-decr-lcount lcount else lcount)
    S))))))
   }
   else do {
     let S = (set-clauses-wl-heur N S);
     - ← log-new-clause-heur S C;
     RETURN S
    }
   })›

definition isa-simplify-clauses-with-unit-st2 :: ‹isasat ⇒ isasat nres› where
  ‹isa-simplify-clauses-with-unit-st2 S =
  do {
    xs ← RETURN (get-avdom S @ get-ivdom S);
   ASSERT(length xs ≤ length (get-vdom S) ∧ length (get-vdom S) ≤ length (get-clauses-wl-heur S));
   (-, T) ← WHILE_T
     (λ(i, T). i < length xs ∧ get-conflict-wl-is-None-heur T)
     (λ(i,T). do {
          ASSERT((i < length (get-avdom T) ⟶ access-avdom-at-pre T i) ∧
          (i ≥ length (get-avdom T) ⟶ access-ivdom-at-pre T (i − length-avdom S)) ∧
          length-avdom T = length-avdom S ∧
          length (get-clauses-wl-heur T) = length (get-clauses-wl-heur S) ∧
           learned-clss-count T ≤ learned-clss-count S);
           let C = (if i < length (get-avdom S) then access-avdom-at T i else access-ivdom-at T (i −
length-avdom S));
         E ← mop-arena-status (get-clauses-wl-heur T) C;
         if E ≠ DELETED then do {
         T ← isa-simplify-clause-with-unit-st2 C T;
         ASSERT(i < length xs);
         RETURN (i+1, T)
       }
       else do {ASSERT(i < length xs); RETURN (i+1,T)}
     })
    (0, S);
   RETURN (reset-units-since-last-GC-st T)
  }›

definition simplify-clauses-with-units-st-wl2 :: ‹-› where
  ‹simplify-clauses-with-units-st-wl2 S = do {
  b ← SPEC(λb::bool. b ⟶ get-conflict-wl S =None);
  if b then simplify-clauses-with-unit-st2 S else RETURN S
  }›

definition isa-simplify-clauses-with-units-st-wl2 :: ‹-› where
  ‹isa-simplify-clauses-with-units-st-wl2 S = do {
  b ← RETURN (get-conflict-wl-is-None-heur S ∧ units-since-last-GC-st S > 0);
  if b then isa-simplify-clauses-with-unit-st2 S else RETURN S
  }›

end
theory IsaSAT-Simplify-Clause-Units-LLVM
  imports IsaSAT-Setup-LLVM IsaSAT-Trail-LLVM
    IsaSAT-Simplify-Units-Defs
    IsaSAT-Proofs-LLVM
begin
```

sepref-register *0 1*

sepref-register *mop-arena-update-lit*

lemma *isa-simplify-clause-with-unit2-alt-def*:
  ‹*isa-simplify-clause-with-unit2 C M N = do* {
    l ← *mop-arena-length N C*;
    ASSERT(*l* < *length N* ∧ *l* ≤ *Suc* (*unat32-max div 2*));
    (*i*, *j*, *N*::*arena*, *is-true*) ← *WHILE*$_T$(λ(*i*, *j*, *N*::*arena*, *b*). ¬*b* ∧ *j* < *l*)
    (λ(*i*, *j*, *N*, *is-true*). *do* {
      ASSERT(*i* ≤ *j* ∧ *j* < *l*);
      L ← *mop-arena-lit2 N C j*;
      let - = *mark-literal-for-unit-deletion L*;
      val ← *mop-polarity-pol M L*;
      if val = *Some True then RETURN* (*i*, *j*+*1*, *N*,*True*)
      else if val = *Some False*
      then RETURN (*i*, *j*+*1*, *N*, *False*)
        else do {
        N ← *mop-arena-update-lit C i L N*;
        RETURN (*i*+*1*, *j*+*1*, *N*, *False*)}
    })
      (*0*, *0*, *N*, *False*);
   L ← *mop-arena-lit2 N C 0*;
   if is-true ∨ *i* ≤ *1*
   then do {
     ASSERT(*mark-garbage-pre* (*N*, *C*));
     RETURN (*False*, *extra-information-mark-to-delete N C*, *L*, *is-true*, *i*)}
   else if i = *j then RETURN* (*True*, *N*, *L*, *is-true*, *i*)
   else do {
      N ← *mop-arena-shorten C i N*;
      RETURN (*False*, *N*, *L*, *is-true*, *i*)}
    }›
  by (*auto simp*: *mark-literal-for-unit-deletion-def isa-simplify-clause-with-unit2-def*)

sepref-def *isa-simplify-clause-with-unit2-code*
  is ‹*uncurry2 isa-simplify-clause-with-unit2*›
  :: ‹[λ((-, -), *N*). *length* (*N*) ≤ *snat64-max*]$_a$ *sint64-nat-assn*^{k} ∗$_a$ *trail-pol-fast-assn*^{k} ∗$_a$ *arena-fast-assn*^{d}
→
  bool1-assn ×$_a$ *arena-fast-assn* ×$_a$ *unat-lit-assn* ×$_a$ *bool1-assn* ×$_a$ *uint32-nat-assn*›
  unfolding *isa-simplify-clause-with-unit2-alt-def*
    length-avdom-def[*symmetric*] *Suc-eq-plus1*[*symmetric*]
    mop-arena-status-st-def[*symmetric*] *isasat-bounded-assn-def*
    SET-TRUE-def[*symmetric*] *SET-FALSE-def*[*symmetric*]
    tri-bool-eq-def[*symmetric*]
  apply (*rewrite at* ‹(⌑, -, -)› *unat-const-fold*[**where** *'a*=*32*])
  apply (*rewrite at* ‹(- ≤ ⌑)› *unat-const-fold*[**where** *'a*=*32*])
  apply (*annot-snat-const* ‹*TYPE*(*64*)›)
  apply (*rewrite at* ‹*mop-arena-update-lit* - ⌑› *annot-unat-snat-upcast*[**where** *'l*=*64*])
  apply (*rewrite at* ‹*If* (⌑ = -)› *annot-unat-snat-upcast*[**where** *'l*=*64*])
  supply [[*goals-limit*=*1*]]
  by *sepref*

sepref-register *cons-trail-Propagated-tr set-conflict-to-false*

lemma *set-conflict-to-false-alt-def*:
  ‹*RETURN o set-conflict-to-false* = (λ(*b*, *n*, *xs*). *RETURN* (*False*, *0*, *xs*))›
  unfolding *set-conflict-to-false-def* **by** *auto*

sepref-def *set-conflict-to-false-code*
  is ‹*RETURN o set-conflict-to-false*›
  :: ‹*conflict-option-rel-assn*^{d} →$_a$ *conflict-option-rel-assn*›
  unfolding *set-conflict-to-false-alt-def conflict-option-rel-assn-def*
    lookup-clause-rel-assn-def
  apply (*annot-unat-const* ‹*TYPE*(*32*)›)
  supply [[*goals-limit=1*]]
  by *sepref*

lemma *isa-simplify-clause-with-unit-st2-alt-def*:
  ‹*isa-simplify-clause-with-unit-st2* = (λ*C* *S*$_0$. *do* {
  let (*lcount*, *S*) = *extract-lcount-wl-heur* *S*$_0$; *let* (*N*, *S*) = *extract-arena-wl-heur* *S*; *let* (*M*, *S*) = *extract-trail-wl-heur* *S*;
  ASSERT (*N* = *get-clauses-wl-heur* *S*$_0$ ∧ *lcount* = *get-learned-count* *S*$_0$ ∧ *M* = *get-trail-wl-heur* *S*$_0$);
  E ← *mop-arena-status* *N* *C*;
   ASSERT(*E* = *LEARNED* ⟶ *1* ≤ *clss-size-lcount lcount*);
  (*unc*, *N*, *L*, *b*, *i*) ← *isa-simplify-clause-with-unit2* *C* *M* *N*;
  ASSERT (*length* *N* ≤ *length* (*get-clauses-wl-heur* *S*$_0$));
   if unc then do {
     let - = *mark-clause-for-unit-as-unchanged* *0*;
     RETURN (*update-arena-wl-heur* *N* (*update-trail-wl-heur* *M* (*update-lcount-wl-heur lcount* *S*)))
   }
   else if b then
   let (*stats*, *S*) = *extract-stats-wl-heur* *S* *in*
   let - = *mark-clause-for-unit-as-unchanged* *0* *in*
   RETURN (*update-trail-wl-heur* *M*
     (*update-arena-wl-heur* *N*
     (*update-stats-wl-heur* (*if E=LEARNED then stats else decr-irred-clss* (*stats*))
     (*update-lcount-wl-heur* (*if E* = *LEARNED then clss-size-decr-lcount* (*lcount*) *else lcount*)
     S))))
   else if i = *1*
   then do {
     M ← *cons-trail-Propagated-tr* *L* *0* *M*;
    let (*stats*, *S*) = *extract-stats-wl-heur* *S*;
    let - = *mark-clause-for-unit-as-unchanged* *0*;
     RETURN (*update-arena-wl-heur* *N*
     (*update-trail-wl-heur* *M*
     (*update-stats-wl-heur* (*if E=LEARNED then incr-uset stats else incr-uset* (*decr-irred-clss stats*))
    (*update-lcount-wl-heur* (*if E* = *LEARNED then clss-size-decr-lcount* (*clss-size-incr-lcountUEk lcount*) *else lcount*)
     S)))) }
   else if i = *0*
   then do {
     j ← *mop-isa-length-trail* *M*;
     let - = *mark-clause-for-unit-as-unchanged* *0*;
     let (*stats*, *S*) = *extract-stats-wl-heur* *S*; *let* (*confl*, *S*) = *extract-conflict-wl-heur* *S*;
     RETURN (*update-trail-wl-heur* *M*
     (*update-arena-wl-heur* *N*
     (*update-conflict-wl-heur* (*set-conflict-to-false confl*)
     (*update-clvls-wl-heur* *0*
     (*update-literals-to-update-wl-heur* *j*

```
    (update-stats-wl-heur (if E=LEARNED then stats else decr-irred-clss stats)
    (update-lcount-wl-heur (if E = LEARNED then clss-size-decr-lcount lcount else lcount)
    S)))))))
   }
   else do {
    let S = (update-trail-wl-heur M
    (update-lcount-wl-heur lcount
    (update-arena-wl-heur N
    S)));
    - ← log-new-clause-heur S C;
    let - = mark-clause-for-unit-as-changed 0;
    RETURN S
   }
  })›
  unfolding isa-simplify-clause-with-unit-st2-def
  apply (auto simp: state-extractors Let-def split: isasat-int-splits intro!: ext bind-cong[OF refl])
  done


lemma [simp]:
  ‹get-clauses-wl-heur (update-trail-wl-heur M S) = get-clauses-wl-heur S›
  ‹get-clauses-wl-heur (update-lcount-wl-heur lc S) = get-clauses-wl-heur S›
  ‹get-clauses-wl-heur (update-arena-wl-heur N S) = N›
  by (cases S) (auto simp: update-a-def update-b-def update-n-def split: isasat-int-splits)

sepref-def isa-simplify-clause-with-unit-st2-code
  is ‹uncurry isa-simplify-clause-with-unit-st2›
  :: ‹[λ(-, S). length (get-clauses-wl-heur S) ≤ snat64-max ∧ learned-clss-count S ≤ unat64-max]_a
  sint64-nat-assn^k *_a isasat-bounded-assn^d → isasat-bounded-assn›
  supply [simp] = learned-clss-count-def
  unfolding isa-simplify-clause-with-unit-st2-alt-def
    length-avdom-def[symmetric] Suc-eq-plus1[symmetric]
    mop-arena-status-st-def[symmetric]
  apply (rewrite at ‹(cons-trail-Propagated-tr - ⌑)› snat-const-fold[where 'a=64])
  apply (rewrite at ‹(mark-clause-for-unit-as-changed ⌑)› unat-const-fold[where 'a=64])
  apply (rewrite at ‹(mark-clause-for-unit-as-unchanged ⌑)› unat-const-fold[where 'a=64])+
  apply (annot-unat-const ‹TYPE(32)›)
  supply [[goals-limit=1]]
  by sepref

end
theory IsaSAT-Simplify-Units
  imports IsaSAT-Setup
    IsaSAT-Simplify-Units-Defs
    IsaSAT-Restart
begin
```

Makes the simplifier loop...

```
definition simplify-clause-with-unit2-rel-simp-wo where
  ‹simplify-clause-with-unit2-rel-simp-wo unc N N_0 N' ⟷
  (unc ⟶ N = N_0 ∧ N' = N_0)›

lemma simplify-clause-with-unit2-rel-simp-wo[iff]:
  ‹simplify-clause-with-unit2-rel-simp-wo True N N_0 N' ⟷
  (N = N_0 ∧ N' = N_0)›
  ‹simplify-clause-with-unit2-rel-simp-wo False N N_0 N'›
```

unfolding *simplify-clause-with-unit2-rel-simp-wo-def* **by** *auto*

definition *simplify-clause-with-unit2-rel* **where**
‹*simplify-clause-with-unit2-rel* N_0 C=
{((*unc*, N, L, b, i), (*unc'*, b', N')).
($C \in\#$ *dom-m* $N \longrightarrow N' = N$) ∧
($C \notin\#$ *dom-m* $N \longrightarrow$ *fmdrop* $C\ N' = N$) ∧
($\neg b \longrightarrow$ *length* $(N' \propto C) = 1 \longrightarrow C \notin\#$*dom-m* $N \wedge N' \propto C = [L]$) ∧
(*if* $b \vee i \leq 1$ *then* $C \notin\#$ *dom-m* N *else* $C \in\#$*dom-m* N) ∧
(b=b') ∧
($\neg b \longrightarrow i =$ *size* $(N' \propto C)$) ∧
$C \in\#$ *dom-m* N' ∧
($b \vee i \leq 1 \longrightarrow$ *size* (*learned-clss-lf* N) = *size* (*learned-clss-lf* N_0) − (*if irred* N_0 C *then 0 else 1*))∧
($\neg(b \vee i \leq 1) \longrightarrow$ *size* (*learned-clss-lf* N) = *size* (*learned-clss-lf* N_0)) ∧
($C \in\#$ *dom-m* $N \longrightarrow$ *dom-m* N = *dom-m* N_0) ∧
($C \in\#$ *dom-m* $N \longrightarrow$ *irred* $N\ C$ = *irred* $N_0\ C \wedge$ *irred* $N\ C$ = *irred* $N'\ C$) ∧
($C \notin\#$ *dom-m* $N \longrightarrow$ *dom-m* N = *remove1-mset* C (*dom-m* N_0)) ∧
unc=*unc'* ∧
simplify-clause-with-unit2-rel-simp-wo unc $N\ N_0\ N'$}›

lemma *simplify-clause-with-unit2-simplify-clause-with-unit*:
fixes $N\ N_0$ :: ‹'*v clauses-l*› **and** N' :: ‹'*a*›
assumes ‹$C \in\#$ *dom-m* N› ‹*no-dup* M› **and**
st: ‹(M,M') ∈ *Id*› ‹(C,C') ∈ *Id*› ‹(N,N_0)∈ *Id*›
shows
‹*simplify-clause-with-unit2* $C\ M\ N \leq \Downarrow$ (*simplify-clause-with-unit2-rel* $N_0\ C$)
(*simplify-clause-with-unit* $C'\ M'\ N_0$)›
proof −
have *simplify-clause-with-unit-alt-def*:
‹*simplify-clause-with-unit* = (λ$C\ M\ N$. *do* {
(*unc*, b, N') ←
SPEC(λ(*unc*, b, N'). *fmdrop* $C\ N$ = *fmdrop* $C\ N' \wedge$ *mset* $(N' \propto C) \subseteq\#$ *mset* $(N \propto C) \wedge C \in\#$ *dom-m* $N' \wedge$
($\neg b \longrightarrow (\forall L \in\#$ *mset* $(N' \propto C)$. *undefined-lit* $M\ L)) \wedge$
($\forall L \in\#$ *mset* $(N \propto C) -$ *mset* $(N' \propto C)$. *defined-lit* $M\ L) \wedge$
(*irred* $N\ C$ = *irred* $N'\ C$) ∧
($b \longleftrightarrow (\exists L.\ L \in\#$ *mset* $(N \propto C) \wedge L \in$ *lits-of-l* $M)) \wedge$
(*unc* $\longrightarrow N = N' \wedge \neg b$));
RETURN (*unc*, b, N')
})› (**is** ‹- = (λ$C\ M\ N$. *do* {
(-, -, -) ← *SPEC* (*?P* $C\ M\ N$);
RETURN -})›)
unfolding *simplify-clause-with-unit-def* **by** *auto*
have *st*: ‹$M' = M$› ‹C'=C› ‹N_0=N›
using *st* **by** *auto*
let *?R* = ‹*measure* (λ(i, j, N', *is-true*). *Suc* (*length* $(N \propto C)) - j$)›
define I **where**
‹I = (λ(i::*nat*, j::*nat*, N':: '*v clauses-l*, *del*:: '*v clause*, *is-true*). $i \leq j \wedge$
$j \leq$ *length* $(N \propto C) \wedge$
$C \in\#$ *dom-m* $N' \wedge$
dom-m N' = *dom-m* $N \wedge$
(
($\forall L \in$*set* (*take* i $(N' \propto C)$). *undefined-lit* $M\ L) \wedge$
($\forall L \in\#$ *del*. *defined-lit* $M\ L) \wedge$
drop j $(N' \propto C)$ = *drop* j $(N \propto C) \wedge$
length $(N' \propto C)$ = *length* $(N \propto C) \wedge$

```
    mset (take j (N ∝ C)) = del + mset (take i (N' ∝ C))) ∧
   fmdrop C N' = fmdrop C N ∧
   (irred N' C = irred N C) ∧
   (is-true ⟷ (∃L ∈ set (take j (N ∝ C)). L ∈ lits-of-l M)) ∧
   (i = j ⟶ take i (N' ∝ C) = take j (N ∝ C)))›
 have I0: ‹I (0, 0, N, {#}, False)›
   using assms unfolding I-def
   by auto
 have H: ‹(if b then RETURN P else RETURN Q) = RETURN (if b then P else Q)› for b P Q
   by auto
 have I-Suc: ‹I (if polarity M (ab ∝ C ! aa) = Some True
   then (a, aa + 1, ab, add-mset (ab ∝ C ! aa) del,True)
   else if polarity M (ab ∝ C ! aa) = Some False
   then (a, aa + 1, ab, add-mset (ab ∝ C ! aa) del, False)
   else (a + 1, aa + 1, ab(C ↪ (ab ∝ C)[a := ab ∝ C ! aa]), del, False))›
   if
     I: ‹I s› and
     ‹case s of (i, j, -, -, b) ⇒ ¬ b ∧ j < length (N ∝ C)› and
     st: ‹s = (a, b)›
       ‹b = (aa, ba)›
       ‹ba = (ab, bdel)›
     ‹bdel = (del, bb)› and
     le: ‹a < length (ab ∝ C) ∧ aa < length (ab ∝ C) ∧ C ∈# dom-m ab ∧ a ≤ aa›
   for s a b aa ba ab bb el del bdel
 proof −
   have[simp]: ‹C ∉# remove1-mset C (dom-m N)›
     using assms distinct-mset-dom[of N]
     by (auto dest!: multi-member-split)
   have [simp]: ‹(take a (ab ∝ C) @ [ab ∝ C ! a])[a := N ∝ C ! aa] =
     take a (ab ∝ C) @ [N ∝ C ! aa]›
     using I le unfolding I-def st
     by (auto simp: list-update-append)

   consider
       ‹polarity M (ab ∝ C ! aa) = Some True› |
       ‹polarity M (ab ∝ C ! aa) = Some False› |
       ‹polarity M (ab ∝ C ! aa) = None›
     by (cases ‹polarity M (ab ∝ C ! aa)›) auto
   then show ?thesis
     using that
     apply cases
     subgoal
       by (auto simp: I-def take-Suc-conv-app-nth fmdrop-fmupd-same
         polarity-spec' assms
         simp flip: Cons-nth-drop-Suc
         dest: in-lits-of-l-defined-litD)
     subgoal
       by (auto simp: I-def take-Suc-conv-app-nth fmdrop-fmupd-same
           polarity-spec' assms
           dest: uminus-lits-of-l-definedD
         simp flip: Cons-nth-drop-Suc)
     subgoal
       by (simp add: I-def take-Suc-conv-app-nth polarity-spec' assms(2)
         fmdrop-fmupd-same nth-append list-update-append
         flip: Cons-nth-drop-Suc)
          (clarsimp simp only: Decided-Propagated-in-iff-in-lits-of-l
```

```
        simp-thms)
    done
qed
have [simp]: ‹C ∉# remove1-mset C (dom-m x1b)› for x1b
  using distinct-mset-dom[of x1b]
  by (cases ‹C ∈# dom-m x1b›) (auto dest!: multi-member-split)
have H0: ‹C = [c] ⟷ mset C = {#c#}› for C c
  by auto
have filt: ‹(⋀x. x ∈# C ⟹ P x) ⟹ filter-mset P C = C›
  ‹(⋀x. x ∈# C ⟹ ¬P x) ⟹ filter-mset P C = {#}›
  ‹filter P D = [] ⟷ (∀ L∈#mset D. ¬P L)›for C P D
  by (simp-all add: filter-mset-eq-conv filter-empty-conv)
have [simp]: ‹take (Suc 0) C = [C!0] ⟷ C ≠ []› for C
  by (cases C) auto
have in-set-dropp-begin:
  ‹drop n xs = drop n ys ⟹ n < length xs ⟹ xs ! n ∈ set ys› for n xs ys
  by (metis in-set-dropD in-set-dropI le-cases)

let ?Q = ‹λ(i::nat, j::nat, N', del, is-true) (unc, b, N'').
  (let P = (if is-true
    then N'(C ↪ filter (Not o defined-lit M) (N ∝ C))
  else N'(C ↪ take i (N' ∝ C)))in
      (P, N'') ∈ Id ∧ ?P C M N (unc, b, N'') ∧
      (is-true ∨ j = length (N ∝ C)) ∧
      (unc ⟷ ¬is-true ∧ i = j ∧ i > 1))›
 have H3: ‹∀ x∈#ab. defined-lit M x ⟹
      undefined-lit M a ⟹
      mset (N ∝ C) = add-mset a ab ⟹
      filter (undefined-lit M) (N ∝ C) = [a]› for a ab
   by (simp add: H0 filt)
 have H4: ‹fmdrop C N = fmdrop C x1a ⟹ C ∈# dom-m x1a ⟹
   size (learned-clss-l x1a) = size (learned-clss-l N) ⟷
   (irred x1a C ⟷ irred N C)› for x1a
   using assms(1)
   apply (auto simp: ran-m-def dest!: multi-member-split split: if-splits
     intro!: filter-mset-cong2)
     apply (smt (verit, best) ‹⋀x1b. C ∉# remove1-mset C (dom-m x1b)› add-mset-remove-trivial
dom-m-fmdrop fmdrop-eq-update-eq fmupd-lookup image-mset-cong2 n-not-Suc-n union-single-eq-member)
     apply (smt (verit, best) ‹⋀x1b. C ∉# remove1-mset C (dom-m x1b)› add-mset-remove-trivial
dom-m-fmdrop fmdrop-eq-update-eq fmupd-lookup image-mset-cong2 n-not-Suc-n union-single-eq-member)
     apply (smt (verit, best) ‹⋀x1b. C ∉# remove1-mset C (dom-m x1b)› add-mset-remove-trivial
dom-m-fmdrop fmdrop-eq-update-eq fmupd-lookup image-mset-cong2 union-single-eq-member)
     by (smt (verit, ccfv-SIG) ‹⋀x1b. C ∉# remove1-mset C (dom-m x1b)› add-mset-remove-trivial
dom-m-fmdrop fmdrop-eq-update-eq fmupd-lookup image-mset-cong2 union-single-eq-member)
 have H5: ‹irred x2c C ⟹
   size (learned-clss-l (fmupd C (x, True) x2c)) =
   size (learned-clss-l x2c)› for x x2c
   using distinct-mset-dom[of x2c]
   by (cases ‹C ∈# dom-m x2c›)
   (force dest!: multi-member-split simp: ran-m-def
     intro: filter-mset-cong2 image-mset-cong2
     intro: multiset.map-cong multiset.map-cong0
     intro!: arg-cong[of - - size])+
have H6: ‹¬irred x2c C ⟹ C ∈# dom-m x2c ⟹
  size (learned-clss-l (fmupd C (x, False) x2c)) =
  (size (learned-clss-l x2c))› for x x2c
```

using *distinct-mset-dom*[*of x2c*]
apply (*cases* ‹*C* ∈# *dom-m x2c*›)
by (*force dest*!: *multi-member-split simp*: *ran-m-def*
intro: *filter-mset-cong2 image-mset-cong2*
intro: *multiset.map-cong multiset.map-cong0*
intro: *arg-cong*[*of - - size*])+
have *H7*: ‹¬*irred x1a C* ⟹ *C* ∈# *dom-m x1a* ⟹
size (*remove1-mset* (*the* (*fmlookup x1a C*)) (*learned-clss-l x1a*)) =
size (*learned-clss-l x1a*) − *Suc 0* › **for** *x1a*
by (*auto simp*: *ran-m-def dest*!: *multi-member-split*)
have *H8*: ‹*fmdrop C x1a* = *fmdrop C N* ⟹ *C* ∈# *dom-m x1a* ⟹
irred x1a C = *irred N C* ⟹ *size* (*learned-clss-l x1a*) − *Suc 0* = *size* (*learned-clss-l N*) − *Suc 0*
› **for** *x1a*
using *assms*(*1*) *distinct-mset-dom*[*of x1a*]
apply (*auto dest*!: *multi-member-split simp*: *ran-m-def*)
apply (*smt* (*verit*, *best*) ‹⋀*x1b*. *C* ∉# *remove1-mset C* (*dom-m x1b*)› *add-mset-remove-trivial dom-m-fmdrop fmdrop-eq-update-eq2 fmupd-lookup image-mset-cong2 union-single-eq-member*)
by (*metis* (*no-types*, *lifting*) *add-mset-remove-trivial dom-m-fmdrop fmdrop-eq-update-eq fmupd-lookup image-mset-cong2 union-single-eq-member*)

have *H9*: ‹*fmdrop C N* = *fmdrop C x1a* ⟹ *remove1-mset C* (*dom-m x1a*) = *remove1-mset C* (*dom-m N*)› **for** *x1a*
by (*metis dom-m-fmdrop*)
have *eq-upd-same*: ‹*fmdrop C aa* = *fmdrop C N* ⟹ *b*= *irred N C* ⟹
N = *fmupd C* (*filter* (*undefined-lit M*) (*N* ∝ *C*), *b*) *aa* ⟷
(∀ *x* ∈ *set* (*N* ∝ *C*). *undefined-lit M x*)› **for** *aa b*
apply (*rule iffI*)
subgoal
by (*subst arg-cong*[*of* ‹*N*› ‹*fmupd C* (*filter* (*undefined-lit M*) (*N* ∝ *C*), *b*) *aa*›
‹λ*N*. *N* ∝ *C*›, *unfolded fmupd-lookup*])
simp-all
subgoal
apply (*subst fmap.fmlookup-inject*[*symmetric*])
apply (*cases* ‹*the* (*fmlookup N C*)›; *cases* ‹*fmlookup N C*›)
using *fmupd-same*[*OF assms*(*1*)] *assms*(*1*)
arg-cong[*of* ‹*fmdrop C aa*› ‹*fmdrop C N*› ‹λ*N*. *fmlookup N x*› **for** *x*, *unfolded fmlookup-drop*]
apply (*auto intro*!: *ext split*: *if-splits*)
by *metis*
done
have *H11*: ‹¬ *irred N C* ⟹ *C* ∈# *dom-m N* ⟹
size (*learned-clss-l* (*fmdrop C N*)) = *size* (*learned-clss-l N*) − *Suc 0*› **for** *N*
using *distinct-mset-dom*[*of N*]
by (*auto simp*: *learned-clss-l-l-fmdrop ran-m-def dest*!: *multi-member-split*
intro!: *arg-cong*[*of - - size*] *image-mset-cong2 filter-mset-cong2*)

have *fmdrop-eq-update-eq'*: ‹*fmdrop C aa* = *fmdrop C N* ⟹ *b* = *irred N C* ⟹ *N* = *fmupd C* (*N* ∝ *C*, *b*) *aa*› **for** *aa b*
using *assms*(*1*) *fmdrop-eq-update-eq* **by** *blast*
have [*simp*]: ‹*fmupd C* (*D*) *aa* = *fmupd C* (*E*) *aa* ⟷ *D* = *E*› **for** *aa D E*
apply *auto*
by (*metis fmupd-lookup option.sel*)
have [*simp*]: ‹(∀ *a*. *a*) ⟷ *False*›
by *blast*
define *simp-work-around* **where** ‹*simp-work-around unc b b'* ≡ *unc* ⟶ *N* = *b* ∧ ¬*b'*› **for** *unc b b'*
have *simp-work-around-simp*[*simp*]: ‹*simp-work-around True b b'* ⟷ *b* = *N* ∧ ¬*b'*› **for** *b b'*
unfolding *simp-work-around-def* **by** *auto*

```
  term ‹{(a, b). I a ∧ ?Q a (b) ∧ (fst b ⟷ ((snd o snd o snd o snd) a))}›
have hd-nth-take: ‹length C > 0 ⟹ [C!0] = take (Suc 0) C› for C
  by (cases C; auto)
show ?thesis
  unfolding simplify-clause-with-unit-alt-def simplify-clause-with-unit2-def
    Let-def H conc-fun-RES st simplify-clause-with-unit2-rel-def
  apply (rule ASSERT-leI)
  subgoal using assms by auto
  apply (refine-vcg WHILEIT-rule-stronger-inv-RES'[where I'=I and R = ‹?R› and
    H = ‹{(a, b). I a ∧ ?Q a b ∧ (fst (snd b) ⟷ ((snd o snd o snd o snd) a))}›])
  subgoal by auto
  subgoal by (auto simp: I-def)
  subgoal by (rule I0)
  subgoal by (auto simp: I-def)
  subgoal by (auto simp: I-def)
  subgoal by (auto simp: I-def)
  subgoal by (auto simp: I-def)
  subgoal by (auto simp: I-def intro: in-set-dropp-begin)
  subgoal by (auto simp: I-def split: if-splits)
  subgoal by (rule I-Suc)
  subgoal by (auto simp: I-def)
  subgoal for s
    apply (cases s)
    apply (clarsimp intro!: RETURN-SPEC-refine)
    apply (intro conjI)
    subgoal
      apply (intro impI)
      apply (clarsimp simp add: I-def fmdrop-fmupd-same)
      apply (auto simp add: I-def mset-remove-filtered
        dest: in-set-takeD)
      done
    subgoal
      by (intro impI)
       (auto simp add: I-def fmdrop-fmupd-same
        intro!: fmdrop-eq-update-eq')
    done
  subgoal
    unfolding I-def simp-work-around-def[symmetric]
    by simp
  subgoal
    unfolding I-def simp-work-around-def[symmetric]
    by simp
  subgoal
    unfolding I-def simp-work-around-def[symmetric]
    by clarsimp
  subgoal for x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e
    unfolding I-def simp-work-around-def[symmetric]
    apply (cases ‹x2e ∨ x1b ≤ 1›)
    apply (simp only: if-True split: )
    subgoal
      apply (simp add: hd-nth-take learned-clss-l-l-fmdrop-irrelev H5 H4 H9[of x2a] H11
          ;
        (subst (asm) eq-commute[of ‹If - (fmupd C (-, -) -) -› x2a])?)
      apply (intro conjI impI allI)
      apply (simp add: hd-nth-take)
      apply (clarsimp simp only:)
```

```
      apply (simp add: )
      apply (clarsimp simp only:)
      apply (clarsimp simp only:)
      apply (simp add: )
      apply (metis length-0-conv)
      apply (clarsimp simp only:; fail)+
      apply (simp add: )
      apply (clarsimp simp only: if-True if-False H11 H8[of ‹fmupd - - x1d›])
      apply (clarsimp simp only: if-True if-False H11 H8[of ‹fmupd - - x1d›] refl
        split: if-splits)
        apply (metis (no-types, lifting) H8)
        apply (metis (no-types, lifting) H8)
      apply (clarsimp simp only: if-True if-False H11 H8[of x1d])
        apply (metis (no-types, lifting))
      apply (clarsimp simp only: if-True if-False H11 H8[of x1d])
      done
    apply (cases ‹x1b = x1c ∧ ¬ x2e›)
    subgoal
      using fmupd-same[of C x1d]
      apply (cases ‹the (fmlookup x1d C)›)
      apply (simp only: if-True if-False simp-thms mem-Collect-eq prod.case
        Let-def linorder-class.not-le[symmetric] simp-work-around-simp
        take-all[OF order.refl] fmupd-lookup refl if-True simp-thms
        option.sel fst-conv simp-work-around-simp eq-commute[of ‹fmupd - - -› N]
        eq-commute [of x2a N]
        fst-conv snd-conv)
      apply (intro conjI impI allI)
      apply (clarsimp simp :; fail)+
      done
   subgoal
      using fmupd-same[of C x1d]
      apply (cases ‹the (fmlookup x1d C)›)
      apply (cases ‹irred x2a C›)
      apply (simp-all only: if-True if-False simp-thms mem-Collect-eq prod.case
        Let-def linorder-class.not-le[symmetric] simp-work-around-simp
        take-all[OF order.refl] fmupd-lookup refl if-True simp-thms H4 H5
        option.sel fst-conv simp-work-around-simp eq-commute[of ‹fmupd - - -› x2a]
        eq-commute [of x2a N] H4 H5
        fst-conv snd-conv;
        intro conjI impI allI)
      apply (clarsimp simp :; fail)+
      apply (clarsimp simp add: eq-commute[of ‹fmupd - - -› x2a])
      apply (metis set-mset-mset union-iff)
      apply (clarsimp simp: H4 H5; fail)+
      done
    done
  done
qed

lemma all-learned-all-lits-all-atms-st:
  ‹set-mset (all-learned-lits-of-wl T) = set-mset (all-learned-lits-of-wl S) ⟹
  set-mset (all-init-lits-of-wl T) = set-mset (all-init-lits-of-wl S) ⟹
  set-mset (all-atms-st T) = set-mset (all-atms-st S)›
  by (metis ℒ_all-all-atms
    all-lits-st-init-learned
    atms-of-ℒ_all-𝒜_in atms-of-cong-set-mset)
```

lemma *simplify-clause-with-unit-st2-simplify-clause-with-unit-st*:
fixes S :: ‹*nat twl-st-wl*›
assumes ‹$(C,C')\in Id$› ‹$(S,S')\in Id$›
shows
‹*simplify-clause-with-unit-st2 C S* ≤ ⇓*Id* (*simplify-clause-with-unit-st-wl C' S'*)›
proof −
show *?thesis*
using *assms*
unfolding *simplify-clause-with-unit-st2-def simplify-clause-with-unit-st-wl-def*
if-False Let-def cons-trail-propagate-l-def nres-monad3 bind-to-let-conv
supply [[*goals-limit=1*]]
apply (*refine-rcg simplify-clause-with-unit2-simplify-clause-with-unit*[*unfolded simplify-clause-with-unit2-rel-def*])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal for *x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i x1j*
x2j x1k x2k x1l x2l x1m x2m x1n - - - -
x2n x1o x2o x1p x2p x1q x2q x1r x2r x1s x2s x x' x1t x2t x1u x2u x1v x2v x1w x2w x1x x2x x1y x2y
by (*cases* ‹*C' ∉# dom-m x1y*›)
(*simp-all add*: *eq-commute*[*of* ‹*remove1-mset - -*› ‹*dom-m -*›])
subgoal by *auto*
subgoal
by (*clarsimp simp only*: *pair-in-Id-conv prod.inject mem-Collect-eq prod.case*)
subgoal by *auto*
subgoal by *auto*
subgoal by (*clarsimp simp only*: *pair-in-Id-conv prod.inject mem-Collect-eq prod.case*) *auto*
subgoal by (*clarsimp simp only*: *pair-in-Id-conv prod.inject mem-Collect-eq prod.case*) *auto*
subgoal by (*rule all-learned-all-lits-all-atms-st*)
subgoal by (*clarsimp simp add*: *learned-clss-l-l-fmdrop-irrelev learned-clss-l-l-fmdrop*
learned-clss-l-fmdrop-if)
subgoal by (*clarsimp simp add*: *ran-m-def dest*!: *multi-member-split*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*simp flip*: *all-lits-of-all-atms-of add*: *all-atms-st-def all-lits-st-def*)
subgoal by *auto*
subgoal by (*clarsimp simp only*: *pair-in-Id-conv prod.inject mem-Collect-eq prod.case*) *auto*
subgoal by (*clarsimp simp only*: *pair-in-Id-conv prod.inject mem-Collect-eq prod.case*) *auto*
subgoal by (*rule all-learned-all-lits-all-atms-st*)
subgoal by (*clarsimp simp add*: *learned-clss-l-l-fmdrop-irrelev learned-clss-l-l-fmdrop*
learned-clss-l-fmdrop-if)
subgoal by (*clarsimp simp add*: *ran-m-def dest*!: *multi-member-split*)
subgoal by *auto*
subgoal by *auto*
subgoal by (*clarsimp simp only*: *pair-in-Id-conv prod.inject mem-Collect-eq prod.case*) *auto*
subgoal by (*clarsimp simp only*: *pair-in-Id-conv prod.inject mem-Collect-eq prod.case*) *auto*
subgoal by (*rule all-learned-all-lits-all-atms-st*)
subgoal by (*clarsimp simp add*: *learned-clss-l-l-fmdrop-irrelev learned-clss-l-l-fmdrop*
learned-clss-l-fmdrop-if)
subgoal by (*clarsimp simp add*: *ran-m-def dest*!: *multi-member-split*)

subgoal by *auto*
subgoal by (*clarsimp simp only*: *pair-in-Id-conv prod.inject mem-Collect-eq prod.case disj.left-neutral*)
subgoal by (*clarsimp simp only*: *pair-in-Id-conv prod.inject mem-Collect-eq prod.case disj.left-neutral*)
subgoal by (*rule all-learned-all-lits-all-atms-st*)
subgoal by (*clarsimp simp add*: *learned-clss-l-l-fmdrop-irrelev learned-clss-l-l-fmdrop learned-clss-l-fmdrop-if*)
subgoal by *auto*
subgoal by *auto*
done
qed

lemma *simplify-clauses-with-unit-st2-simplify-clauses-with-unit-st*:
assumes ‹$(S,S')\in Id$›
shows
‹*simplify-clauses-with-unit-st2* $S \leq \Downarrow Id$ (*simplify-clauses-with-unit-st-wl* S')›
proof −
have *inj*: ‹*inj-on id* x› **for** x
by *auto*
show *?thesis*
using *assms*
unfolding *simplify-clauses-with-unit-st2-def simplify-clauses-with-unit-st-wl-def*
by (*refine-vcg simplify-clause-with-unit-st2-simplify-clause-with-unit-st inj*)
auto
qed

lemma *simplify-clause-with-unit2-alt-def*:
‹*simplify-clause-with-unit2* $C\ M\ N_0$ = *do* {
ASSERT($C \in\#$ *dom-m* N_0);
let l = *length* ($N_0 \propto C$);
(i, j, N, *del*, *is-true*) ← $WHILE_T^{\lambda(i,\ j,\ N,\ del,\ b).\ C \in\#\ dom\text{-}m\ N}$
($\lambda(i,\ j,\ N,\ del,\ b).\ \neg b \wedge j < l$)
($\lambda(i,\ j,\ N,\ del,\ is\text{-}true).$ *do* {
ASSERT($i <$ *length* ($N \propto C$) $\wedge\ j <$ *length* ($N \propto C$) $\wedge\ C \in\#$ *dom-m* $N \wedge\ i \leq j$);
let $L = N \propto C\ !\ j$;
ASSERT($L \in$ *set* ($N_0 \propto C$));
let val = *polarity* $M\ L$;
if val = *Some True then RETURN* (i, j+*1*, N, *add-mset* L *del*, *True*)
else if val = *Some False*
then RETURN (i, j+*1*, N, *add-mset* L *del*, *False*)
else let $N = N(C \hookrightarrow ((N \propto C)[i := L]))$ *in RETURN* (i+*1*, j+*1*, N, *del*, *False*)
})
(*0*, *0*, N_0, {#}, *False*);
ASSERT ($C \in\#$ *dom-m* $N \wedge\ i \leq$ *length* ($N \propto C$));
ASSERT (*is-true* $\vee\ j =$ *length* ($N_0 \propto C$));
let $L = N \propto C\ !\ 0$;
if is-true $\vee\ i \leq 1$
then RETURN (*False*, *fmdrop* $C\ N$, L, *is-true*, i)
else if i=$j \wedge \neg$ *is-true then RETURN* (*True*, N, L, *is-true*, i)
else do {
let $N = N(C \hookrightarrow$ (*take* i ($N \propto C$))) *in RETURN* (*False*, N, L, *is-true*, i)}
}›
unfolding *Let-def simplify-clause-with-unit2-def*

by (*auto intro*!: *bind-cong*[*OF refl*])

lemma *normalize-down-return-spec*: ‹⇓*A* ((*RETURN o f*) *c*) = *SPEC* (λ*a*. (*a*, *f c*) ∈ {(*a*,*b*). (*a*,*b*) ∈ *A* ∧ *b* = *f c*})›
by (*auto simp*: *conc-fun-RES RETURN-def*)

lemma *arena-length-le-length-arena*:
‹*C'* ∈# *dom-m N* ⟹
valid-arena arena N vdom ⟹
arena-length arena C' < *length arena*›
by (*smt* (*verit*, *best*) *Nat.le-diff-conv2 STATUS-SHIFT-def Suc-le-lessD arena-lifting*(10) *arena-lifting*(16) *arena-lifting*(4) *diff-self-eq-0 le-trans less-Suc-eq not-less-eq not-less-eq-eq numeral-2-eq-2*)

lemma *simplify-clause-with-unit-st-wl-preD*:
assumes ‹*simplify-clause-with-unit-st-wl-pre C S*›
shows
simplify-clause-with-unit-st-wl-pre-all-init-atms-all-atms:
‹*set-mset* (*all-init-atms-st S*) = *set-mset* (*all-atms-st S*)› **and**
‹*isasat-input-bounded* (*all-init-atms-st S*) ⟹ *length* (*get-clauses-wl S* ∝ *C*) ≤ *Suc* (*unat32-max div* 2)›
proof −
obtain *x xa* **where**
Sx: ‹(*S*, *x*) ∈ *state-wl-l None*› **and**
C: ‹*C* ∈# *dom-m* (*get-clauses-l x*)› **and**
‹*count-decided* (*get-trail-l x*) = *0*› **and**
‹*get-conflict-l x* = *None*› **and**
‹*clauses-to-update-l x* = {#}› **and**
xxa: ‹(*x*, *xa*) ∈ *twl-st-l None*› **and**
‹*twl-st-inv xa*› **and**
‹*valid-enqueued xa*› **and**
‹*cdcl*$_W$*-restart-mset.no-smaller-propa* (*state*$_W$*-of xa*)› **and**
alien: ‹*cdcl*$_W$*-restart-mset.no-strange-atm* (*state*$_W$*-of xa*)› **and**
‹*entailed-clss-inv* (*pstate*$_W$*-of xa*)› **and**
‹*twl-st-exception-inv xa*› **and**
‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-M-level-inv* (*state*$_W$*-of xa*)› **and**
‹*psubsumed-invs* (*pstate*$_W$*-of xa*)› **and**
‹*clauses0-inv* (*pstate*$_W$*-of xa*)› **and**
‹*no-duplicate-queued xa*› **and**
‹∀ *s*∈#*learned-clss* (*state*$_W$*-of xa*). ¬ *tautology s*› **and**
‹*distinct-queued xa*› **and**
dist: ‹*cdcl*$_W$*-restart-mset.distinct-cdcl*$_W$*-state* (*state*$_W$*-of xa*)› **and**
‹*confl-cands-enqueued xa*› **and**
‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-conflicting* (*state*$_W$*-of xa*)› **and**
‹*propa-cands-enqueued xa*› **and**
‹*all-decomposition-implies-m* (*cdcl*$_W$*-restart-mset.clauses* (*state*$_W$*-of xa*))
(*get-all-ann-decomposition* (*trail* (*state*$_W$*-of xa*)))› **and**
‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-learned-clause* (*state*$_W$*-of xa*)› **and**
‹*get-conflict xa* ≠ *None* ⟶ *clauses-to-update xa* = {#} ∧ *literals-to-update xa* = {#}› **and**
‹*clauses-to-update-inv xa*› **and**
‹*past-invs xa*› **and**
list: ‹*twl-list-invs x*›
using *assms*
unfolding *simplify-clause-with-unit-st-wl-pre-def twl-struct-invs-def*
simplify-clause-with-unit-st-pre-def pcdcl-all-struct-invs-def
cdcl$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*

```
    state_W-of-def[symmetric] apply −
   by blast

  show H: ‹set-mset (all-init-atms-st S) = set-mset (all-atms-st S)›
   using Sx xxa alien
   unfolding all-init-atms-def all-init-atms-st-def all-atms-st-def all-init-lits-def
    all-lits-of-mm-union image-mset-union get-unit-clauses-wl-alt-def all-atms-def all-lits-def
    set-mset-union atm-of-all-lits-of-mm cdcl_W-restart-mset.no-strange-atm-def
   apply (subst (2)all-clss-l-ran-m[symmetric])
   apply (subst all-clss-l-ran-m[symmetric])
   unfolding image-mset-union filter-union-mset atms-of-ms-union set-mset-union
   by auto
  have ‹distinct-mset (mset (get-clauses-wl S ∝ C))›
   using Sx xxa dist C
   by (auto simp: cdcl_W-restart-mset.distinct-cdcl_W-state-def ran-m-def conj-disj-distribR image-Un
    Collect-disj-eq Collect-conv-if split: if-splits
    dest!: multi-member-split)
  moreover have ‹literals-are-in-ℒ_in (all-init-atms-st S) (mset (get-clauses-wl S ∝ C))›
   using C Sx xxa unfolding H literals-are-in-ℒ_in-def ℒ_all-cong[OF H]
   by (auto simp:all-atms-st-def ran-m-def in-ℒ_all-atm-of-𝒜_in
    all-lits-of-mm-add-mset all-lits-def
    all-atms-def all-init-lits-def dest!: multi-member-split
    simp del: all-atms-def[symmetric])
  moreover have ‹¬tautology (mset (get-clauses-wl S ∝ C))›
   using list C Sx unfolding twl-list-invs-def
   by auto
  ultimately show ‹length (get-clauses-wl S ∝ C) ≤ Suc (unat32-max div 2)›
   if ‹isasat-input-bounded (all-init-atms-st S)›
   using simple-clss-size-upper-div2[OF that, of ‹mset (get-clauses-wl S ∝ C)›] by auto
qed

lemma isa-simplify-clause-with-unit2-isa-simplify-clause-with-unit:
  assumes ‹valid-arena arena N vdom› and
   trail: ‹(M, M') ∈ trail-pol 𝒜› and
   lits: ‹literals-are-in-ℒ_in-mm 𝒜 (mset `# ran-mf N)› and
   C: ‹(C,C')∈Id› and
   le: ‹length (N ∝ C) ≤ Suc (unat32-max div 2)›
  shows ‹isa-simplify-clause-with-unit2 C M arena ≤ ⇓
   (bool-rel ×_r {(arena', N). valid-arena arena' N vdom ∧ length arena' = length arena} ×_r
   Id ×_r bool-rel ×_r nat-rel)
   (simplify-clause-with-unit2 C' M' N)›
proof −
  have C: ‹C'=C›
   using C by auto

  have [refine0]: ‹C ∈# dom-m N ⟹
  ((0, 0, arena, False), 0, 0, N, {#}, False) ∈ {((i, j, N, is-true),
   (i', j', N', del', is-true')).
   ((i, j, N, is-true), (i', j', N', is-true')) ∈
   nat-rel ×_r nat-rel ×_r {(arena', N). valid-arena arena' N vdom ∧ length arena' = length arena ∧ C
∈# dom-m N} ×_r
   bool-rel}›
   using assms by auto
  show ?thesis
   supply [[goals-limit=1]]
   unfolding isa-simplify-clause-with-unit2-def simplify-clause-with-unit2-alt-def
```

mop-polarity-pol-def nres-monad3 C
apply (*refine-rcg mop-arena-lit*[**where** *vdom*=*vdom*]
polarity-pol-pre[*OF trail*]
polarity-pol-polarity[*of* $\mathcal{A}$, *unfolded option-rel-id-simp*,
THEN fref-to-Down-unRET-uncurry]
mop-arena-shorten-spec[**where** *vdom*=*vdom*]
mop-arena-length[*THEN fref-to-Down-curry*, *of N C - - vdom*, *unfolded normalize-down-return-spec*]
)
subgoal using *assms* **by** (*auto simp add*: *arena-lifting*)
subgoal using *assms* **by** (*auto simp add*: *arena-lifting*)
subgoal
using *assms* **by** (*auto simp add*: *arena-lifting arena-length-le-length-arena*)

subgoal using *le* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal
using *lits*
by (*auto simp*:
all-lits-of-mm-def ran-m-def dest!: *multi-member-split*
dest!: *literals-are-in-*$\mathcal{L}_{in}$*-mm-add-msetD*)
subgoal
using *lits*
by (*auto simp*:
all-lits-of-mm-def ran-m-def dest!: *multi-member-split*
dest!: *literals-are-in-*$\mathcal{L}_{in}$*-mm-add-msetD*)
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
apply (*rule-tac mop-arena-update-lit-spec*[**where** *vdom*=*vdom*])
subgoal by *auto*
subgoal by (*auto simp*: *arena-lifting*)
subgoal by (*auto simp*: *arena-lifting*)
subgoal by (*auto simp*: *arena-lifting*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal for - *x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f*
using *arena-lifting*(*4*,*19*)[*of x1f x1b vdom C*] **by** *auto*
subgoal by *auto*
subgoal by (*auto simp*: *mark-garbage-pre-def arena-is-valid-clause-idx-def*)
subgoal by (*auto intro*!: *valid-arena-extra-information-mark-to-delete'*)
subgoal by *auto*
subgoal by (*auto simp*: *arena-lifting*)

subgoal by *auto*
subgoal by (*auto simp*: *arena-lifting*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

lemma *literals-are-in-mm-clauses*:
‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* (*all-atms-st T*) (*mset* ‘# *ran-mf* (*get-clauses-wl T*))›
unfolding *all-atms-st-def all-atms-def all-lits-def*
by (*auto simp*: *all-lits-of-mm-union*
literals-are-in-$\mathcal{L}_{in}$*-mm-def in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$)

lemma *mop-arena-status*:
assumes ‹*C* ∈# *dom-m N*› **and** ‹(*C*,*C'*)∈*nat-rel*›
‹*valid-arena arena N vdom*›
shows
‹*mop-arena-status arena C*
≤ *SPEC*
(λ*c*. (*c*, *irred N C'*)
∈ {(*a*,*b*). (*a* = *IRRED* ⟷ *b*) ∧ (*a* = *LEARNED* ⟷ ¬*b*) ∧ (*irred N C'* = *b*)})›
using *assms* **unfolding** *mop-arena-status-def*
by (*auto intro*!: *ASSERT-leI simp*: *arena-is-valid-clause-vdom-def*
arena-lifting)

lemma *twl-st-heur-restart-alt-def*[*unfolded Let-def*]:
‹*twl-st-heur-restart* =
{(*S*, *T*).
let M' = *get-trail-wl-heur S*; *N'* = *get-clauses-wl-heur S*; *D'* = *get-conflict-wl-heur S*;
W' = *get-watched-wl-heur S*; *j* = *literals-to-update-wl-heur S*; *outl* = *get-outlearned-heur S*;
cach = *get-conflict-cach S*; *clvls* = *get-count-max-lvls-heur S*;
vm = *get-vmtf-heur S*;
vdom = *get-aivdom S*; *heur* = *get-heur S*; *old-arena* = *get-old-arena S*;
lcount = *get-learned-count S*; *occs* = *get-occs S in*
let M = *get-trail-wl T*; *N* = *get-clauses-wl T*; *D* = *get-conflict-wl T*;
Q = *literals-to-update-wl T*;
W = *get-watched-wl T*; *N0* = *get-init-clauses0-wl T*; *U0* = *get-learned-clauses0-wl T*;
NS = *get-subsumed-init-clauses-wl T*; *US* = *get-subsumed-learned-clauses-wl T*;
NEk = *get-kept-unit-init-clss-wl T*; *UEk* = *get-kept-unit-learned-clss-wl T*;
NE = *get-unkept-unit-init-clss-wl T*; *UE* = *get-unkept-unit-learned-clss-wl T in*
let $\mathcal{A}$ = *all-init-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*) *in*
(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$ ∧
valid-arena N' N (*set* (*get-vdom-aivdom vdom*)) ∧
(*D'*, *D*) ∈ *option-lookup-clause-rel* $\mathcal{A}$ ∧
(*D* = *None* ⟶ *j* ≤ *length M*) ∧
Q = *uminus* ‘# *lit-of* ‘# *mset* (*drop j* (*rev M*)) ∧
(*W'*, *W*) ∈ ⟨*Id*⟩*map-fun-rel* (*D*$_0$ $\mathcal{A}$) ∧
vm ∈ *bump-heur* $\mathcal{A}$ *M* ∧
no-dup M ∧
clvls ∈ *counts-maximum-level M D* ∧
cach-refinement-empty $\mathcal{A}$ *cach* ∧

out-learned M D outl ∧
clss-size-corr-restart N NE {#} *NEk UEk NS* {#} *N0* {#} *lcount* ∧
vdom-m $\mathcal{A}$ *W N* ⊆ *set* (*get-vdom-aivdom vdom*) ∧
aivdom-inv-dec vdom (*dom-m N*) ∧
isasat-input-bounded $\mathcal{A}$ ∧
isasat-input-nempty $\mathcal{A}$ ∧
old-arena = [] ∧
heuristic-rel $\mathcal{A}$ *heur* ∧
(*occs*, *empty-occs-list* $\mathcal{A}$) ∈ *occurrence-list-ref*
}›
unfolding *twl-st-heur-restart-def all-init-atms-st-def Let-def* **by** *auto*

lemma *literals-are-in-*$\mathcal{L}_{in}$*-mm-cong*:
‹*set-mset A* = *set-mset B* ⟹ *literals-are-in-*$\mathcal{L}_{in}$*-mm A* = *literals-are-in-*$\mathcal{L}_{in}$*-mm B*›
unfolding *literals-are-in-*$\mathcal{L}_{in}$*-mm-def* $\mathcal{L}_{all}$*-cong* **by** *force*

lemma *aivdom-inv-mono*:
‹*B* ⊆# *A* ⟹ *aivdom-inv* (*v*, *x1y*, *x2aa*, *tvdom*) *A* ⟹ *aivdom-inv* (*v*, *x1y*, *x2aa*, *tvdom*) *B*›
using *distinct-mset-mono*[*of B A*]
by (*auto simp*: *aivdom-inv-alt-def*)

lemma *aivdom-inv-dec-mono*:
‹*B* ⊆# *A* ⟹ *aivdom-inv-dec vdom A* ⟹ *aivdom-inv-dec vdom B*›
using *aivdom-inv-mono*[*of B A* ‹*get-vdom-aivdom vdom*› ‹*get-avdom-aivdom vdom*› ‹*get-ivdom-aivdom vdom*›
‹*get-tvdom-aivdom vdom*›]
by (*cases vdom*) (*auto simp*: *aivdom-inv-dec-def*)

lemma *simplify-clause-with-unit-st2-alt-def*:
‹*simplify-clause-with-unit-st2* = (λ*C* (*M*, N_0, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*). *do* {
ASSERT(*simplify-clause-with-unit-st-wl-pre C* (*M*, N_0, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*));
ASSERT (*C* ∈# *dom-m* N_0 ∧ *count-decided M* = 0 ∧ *D* = *None*);
let S = (*M*, N_0, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*);
let E = *mset* (N_0 ∝ *C*);
let irr = *irred* N_0 *C*;
(*unc*, *N*, *L*, *b*, *i*) ← *simplify-clause-with-unit2 C M* N_0;
ASSERT(*dom-m N* ⊆# *dom-m* N_0);
if unc then do {
ASSERT(*N* = N_0);
let T = (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*);
RETURN T
}
else if b then do {
let T = (*M*, *N*, *D*, (*if irr then add-mset E else id*) *NE*, (*if* ¬*irr then add-mset E else id*) *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*);
ASSERT (*set-mset* (*all-learned-lits-of-wl T*) = *set-mset* (*all-learned-lits-of-wl S*));
ASSERT (*set-mset* (*all-init-lits-of-wl T*) = *set-mset* (*all-init-lits-of-wl S*));
ASSERT (*set-mset* (*all-atms-st T*) = *set-mset* (*all-atms-st S*));
ASSERT (*size* (*learned-clss-lf N*) = *size* (*learned-clss-lf* N_0) − (*if irr then 0 else 1*));
ASSERT(¬*irr* ⟶ *size* (*learned-clss-lf* N_0) ≥ 1);
RETURN T
}

else if $i = 1$
then do {
ASSERT (*undefined-lit M L* $\wedge$ $L \in\#$ $\mathcal{L}_{all}$ (*all-atms-st S*));
$M \leftarrow$ *cons-trail-propagate-l L 0 M*;
let T = (M, N, D, NE, UE, (*if irr then add-mset* {#L#} *else id*) NEk, (*if* ¬*irr then add-mset* {#L#} *else id*) UEk, (*if irr then add-mset E else id*) NS, (*if* ¬*irr then add-mset E else id*) US, $N0$, $U0$, *add-mset* ($-L$) Q, W);
ASSERT (*set-mset* (*all-learned-lits-of-wl T*) = *set-mset* (*all-learned-lits-of-wl S*));
ASSERT (*set-mset* (*all-init-lits-of-wl T*) = *set-mset* (*all-init-lits-of-wl S*));
ASSERT (*set-mset* (*all-atms-st T*) = *set-mset* (*all-atms-st S*));
ASSERT (*size* (*learned-clss-lf N*) = *size* (*learned-clss-lf* N_0) − (*if irr then 0 else 1*));
ASSERT(¬*irr* ⟶ *size* (*learned-clss-lf* N_0) ≥ 1);
RETURN T
}
else if $i = 0$
then do {
let j = *length* M;
let T = (M, N, *Some* {#}, NE, UE, NEk, UEk, (*if irr then add-mset E else id*) NS, (*if* ¬*irr then add-mset E else id*) US, (*if irr then add-mset* {#} *else id*) $N0$, (*if* ¬*irr then add-mset* {#} *else id*) $U0$, {#}, W);
ASSERT (*set-mset* (*all-learned-lits-of-wl T*) = *set-mset* (*all-learned-lits-of-wl S*));
ASSERT (*set-mset* (*all-init-lits-of-wl T*) = *set-mset* (*all-init-lits-of-wl S*));
ASSERT (*set-mset* (*all-atms-st T*) = *set-mset* (*all-atms-st S*));
ASSERT (*size* (*learned-clss-lf N*) = *size* (*learned-clss-lf* N_0) − (*if irr then 0 else 1*));
ASSERT(¬*irr* ⟶ *size* (*learned-clss-lf* N_0) ≥ 1);
RETURN T
}
else do {
let T = (M, N, D, NE, UE, NEk, UEk, (*if irr then add-mset E else id*) NS, (*if* ¬*irr then add-mset E else id*) US, $N0$, $U0$, Q, W);
ASSERT (*set-mset* (*all-learned-lits-of-wl T*) = *set-mset* (*all-learned-lits-of-wl S*));
ASSERT (*set-mset* (*all-init-lits-of-wl T*) = *set-mset* (*all-init-lits-of-wl S*));
ASSERT (*set-mset* (*all-atms-st T*) = *set-mset* (*all-atms-st S*));
ASSERT (*size* (*learned-clss-lf N*) = *size* (*learned-clss-lf* N_0));
ASSERT ($C \in\#$ *dom-m N*);
let - = *log-clause T C*;
RETURN T
}
})›
by (*auto simp*: *simplify-clause-with-unit-st2-def log-clause-def intro*!: *ext Let-def cong*: *if-cong*)

lemma *isa-simplify-clause-with-unit-st2-simplify-clause-with-unit-st2*:
assumes ‹(S, S') ∈ {(a,b). (a,b) ∈ *twl-st-heur-restart* ∧ *get-avdom a* = u∧ *get-vdom a* = v∧
get-ivdom a = x ∧*length* (*get-clauses-wl-heur a*) = r ∧
learned-clss-count a ≤ w ∧ *get-vmtf-heur a* = *vm* ∧
length (*get-watched-wl-heur a*) = *lw*}›
‹(C,C')∈ *nat-rel*›
shows ‹*isa-simplify-clause-with-unit-st2 C S* ≤
⇓{(a,b). (a,b) ∈ *twl-st-heur-restart* ∧ *get-avdom a* = u∧ *get-vdom a* = v∧ *get-ivdom a* = x ∧
length (*get-clauses-wl-heur a*) = r∧
learned-clss-count a ≤ w ∧ *get-vmtf-heur a* = *vm* ∧
learned-clss-count a ≤ *learned-clss-count S* ∧
length (*get-watched-wl-heur a*) = *lw*} (*simplify-clause-with-unit-st2 C' S'*)› (**is** ‹- ≤ ⇓*?A* -›)
proof −
have H: ‹$A = B \Longrightarrow x \in A \Longrightarrow x \in B$› **for** A B x
by *auto*

have H': ‹$A = B \Longrightarrow A\ x \Longrightarrow B\ x$› **for** $A\ B\ x$
by *auto*
have H'': ‹$A = B \Longrightarrow A \subseteq c \Longrightarrow B \subseteq c$› **for** $A\ B\ c$
by *auto*

have *vdom-m-cong2*: ‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B} \Longrightarrow$ *vdom-m* $\mathcal{A}\ W\ N \subseteq vd \Longrightarrow$ *dom-m* N' ⊆# *dom-m* $N \Longrightarrow$
vdom-m $\mathcal{B}\ W\ N' \subseteq vd$›
for $\mathcal{A}\ W\ N\ N'\ vd\ \mathcal{B}$
by (*subst vdom-m-cong*[*of* $\mathcal{B}\ \mathcal{A}$])
(*auto simp*: *vdom-m-def*)
note *cong* = *trail-pol-cong heuristic-rel-cong*
option-lookup-clause-rel-cong isa-vmtf-cong
vdom-m-cong[*THEN H*] *isasat-input-nempty-cong*[*THEN iffD1*]
isasat-input-bounded-cong[*THEN iffD1*]
cach-refinement-empty-cong[*THEN H'*]
phase-saving-cong[*THEN H'*]
$\mathcal{L}_{all}$-*cong*[*THEN H*]
D_0-*cong*[*THEN H*]
D_0-*cong*[*OF sym*]
vdom-m-cong[*THEN H''*]
vdom-m-cong2
empty-occs-list-cong
have *set-conflict-to-false*: ‹(*a*, *None*) ∈ *option-lookup-clause-rel* $\mathcal{A} \Longrightarrow$
(*set-conflict-to-false a*, *Some* {#}) ∈ *option-lookup-clause-rel* $\mathcal{A}$› **for** *a* $\mathcal{A}$
by (*auto simp*: *option-lookup-clause-rel-def set-conflict-to-false-def*
lookup-clause-rel-def)
have *outl*: ‹*out-learned x1 None x1s* $\Longrightarrow$ *out-learned x1* (*Some* {#}) (*x1s*)›
‹*0* ∈ *counts-maximum-level x1* (*Some* {#})› **for** *x1 x1s*
by (*cases x1s*, *auto simp*: *out-learned-def counts-maximum-level-def*)
have *all-init-lits-of-wl*:
‹*set-mset* (*all-init-lits-of-wl a*) = *set-mset* (*all-init-lits-of-wl b*) ⟷
set-mset (*all-init-atms-st a*) = *set-mset* (*all-init-atms-st b*)› **for** *a b*
by (*metis* $\mathcal{L}_{all}$-*all-init-atms*(*2*) $\mathcal{L}_{all}$-*cong atms-of-*$\mathcal{L}_{all}$-$\mathcal{A}_{in}$ *atms-of-cong-set-mset*)
have *log-clause*[*refine0*]: ‹y' ∈# *dom-m* (*get-clauses-wl* x') $\Longrightarrow$ (x, x')∈ *?A* $\Longrightarrow$ (y,y')∈ *nat-rel* $\Longrightarrow$
log-new-clause-heur x y ≤ *SPEC*(λ*c*. (*c*, *log-clause* x' y') ∈ *unit-rel*)› **for** *x* x' *y* y'
unfolding *log-new-clause-heur-alt-def*
apply (*rule log-clause-heur-log-clause2-ana*[*THEN order-trans*])
apply (*auto simp add*: *twl-st-heur-restart-ana-def*)
by (*rule log-clause2-log-clause*[*THEN fref-to-Down-curry*, *unfolded prod.simps*, *THEN order-trans*])
auto

note *accessors-def* = *get-trail-wl.simps get-clauses-wl.simps get-conflict-wl.simps literals-to-update-wl.simps*
get-watched-wl.simps get-init-clauses0-wl.simps get-learned-clauses0-wl.simps
get-subsumed-init-clauses-wl.simps get-subsumed-learned-clauses-wl.simps
get-kept-unit-init-clss-wl.simps get-kept-unit-learned-clss-wl.simps isasat-state-simp
get-unkept-unit-init-clss-wl.simps get-unkept-unit-learned-clss-wl.simps
show *?thesis*
supply [[*goals-limit=1*]]
using *assms*
unfolding *isa-simplify-clause-with-unit-st2-def*
simplify-clause-with-unit-st2-alt-def Let-def[*of* (-,-)] *Let-def*[*of* ‹*mset* -›]
all-init-lits-of-wl
apply (*rewrite at* ‹*let* - = *set-clauses-wl-heur* - - *in* -› *Let-def*)
apply (*refine-rcg isa-simplify-clause-with-unit2-isa-simplify-clause-with-unit*[**where**
vdom=‹*set* (*get-vdom S*)› **and** $\mathcal{A}$ = ‹*all-init-atms-st S'*›]

mop-arena-status[**where** *vdom* = ‹*set* (*get-vdom S*)›]
cons-trail-Propagated-tr2[**where** $\mathcal{A}$ = ‹*all-init-atms-st S'*›]
mop-isa-length-trail-length-u[*of* ‹*all-init-atms-st* (*S'*)›, *THEN fref-to-Down-Id-keep*,
unfolded length-uint32-nat-def comp-def])
subgoal by *auto*
subgoal by (*auto simp*: *twl-st-heur-restart-def*)
subgoal by (*auto simp*: *twl-st-heur-restart-def clss-size-corr-def ran-m-def*
clss-size-def
dest!: *multi-member-split clss-size-corr-restart-rew*)
subgoal
by (*auto simp*: *twl-st-heur-restart-def*)
subgoal
by (*auto simp*: *twl-st-heur-restart-def all-init-atms-st-def*)
subgoal
using *literals-are-in-mm-clauses*[*of S'*]
simplify-clause-with-unit-st-wl-pre-all-init-atms-all-atms[*of C' S'*,
THEN literals-are-in-$\mathcal{L}_{in}$*-mm-cong*]
by (*auto simp*: *twl-st-heur-restart-def*)
subgoal
by (*drule simplify-clause-with-unit-st-wl-preD*(*2*)[*of C'*])
(*auto dest*!: *simp*: *twl-st-heur-restart-def all-init-atms-st-def*
simp del: *isasat-input-bounded-def*)
subgoal
by *auto*
subgoal
by (*auto simp*: *twl-st-heur-restart-def learned-clss-count-def*)
subgoal
by (*auto simp*: *twl-st-heur-restart-def*)
subgoal
by (*auto simp*: *twl-st-heur-restart-def*)
subgoal for *x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n*
x1o x2o x1p x2p x1q x2q x1r x2r x1s x2s x1t x2t x1u
by (*clarsimp simp only*: *twl-st-heur-restart-alt-def in-pair-collect-simp prod.simps*
prod-rel-iff TrueI refl
cong[*of* ‹*all-init-atms-st* (*x1*, *x1a*, *None*, *x1c*, *x1d*, *x1e*, *x1f*, *x1g*, *x1h*,
x1i, *x1j*, *uminus* ‘# *lit-of* ‘# *mset* (*drop* (*literals-to-update-wl-heur S*) (*rev x1*)), *x2k*)›
‹*all-init-atms-st* (-, -, -, (*If* - - -) -, -)›] *clss-size-corr-restart-def isasat-state-simp*
clss-size-def clss-size-incr-lcountUE-def learned-clss-count-def aivdom-inv-dec-mono
empty-occs-list-cong[*of* ‹*all-init-atms-st* (-, -, -, (*If* - - -) -, -)›]
clss-size-decr-lcount-def accessors-def)
(*auto split*: *if-splits intro*: *aivdom-inv-dec-mono simp*:
clss-size-decr-lcount-def clss-size-lcount-def clss-size-lcountUS-def
clss-size-lcountU0-def clss-size-lcountUE-def clss-size-lcountUEk-def)
subgoal by *simp*
subgoal by (*auto simp*: *twl-st-heur-restart-def all-init-atms-st-def*)
subgoal
using *simplify-clause-with-unit-st-wl-pre-all-init-atms-all-atms*[*of C' S'*, *THEN* $\mathcal{L}_{all}$*-cong*]
by *auto*
subgoal by (*auto simp*: *DECISION-REASON-def*)
subgoal for *x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p*
x2p x1q x2q x1r x2r x1s x2s x1t x2t x1u x2u x1v
by (*clarsimp simp only*: *twl-st-heur-restart-alt-def in-pair-collect-simp prod.simps*
prod-rel-iff TrueI refl accessors-def
cong[*of* ‹*all-init-atms-st* (*x1*, *x1a*, *None*, *x1c*, *x1d*, *x1e*, *x1f*, *x1g*, *x1h*,

x1i, *x1j*, *uminus ‘# lit-of ‘# mset* (*drop* (*literals-to-update-wl-heur S*) (*rev x1*)), *x2k*)›
‹*all-init-atms-st* (-, -, -, -, -, (*If* - - -) -, -)›] *isa-vmtf-consD clss-size-corr-restart-def*
clss-size-def clss-size-incr-lcountUEk-def learned-clss-count-def aivdom-inv-dec-mono empty-occs-list-cong[*of* ‹*all-init-atms-st* (-, -, -, -, -, (*If* - - -) -, -)›]
clss-size-decr-lcount-def)
(*auto split*: *if-splits intro*: *aivdom-inv-dec-mono simp*:
clss-size-decr-lcount-def clss-size-lcount-def clss-size-lcountUS-def
clss-size-lcountU0-def clss-size-lcountUE-def clss-size-lcountUEk-def)
subgoal by *simp*
subgoal by *simp*
subgoal by (*auto simp add*: *twl-st-heur-restart-def all-init-atms-st-def*)
subgoal for *x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p*
x2p x1q x2q x1r x2r x1s x2s x1t x2t x1u x2u
by (*clarsimp simp only*: *twl-st-heur-restart-alt-def in-pair-collect-simp prod.simps*
isasat-state-simp prod-rel-iff TrueI refl accessors-def
cong[*of* ‹*all-init-atms-st* (*x1*, *x1a*, *None*, *x1c*, *x1d*, *x1e*, *x1f*, *x1g*, *x1h*,
x1i, *x1j*, *uminus ‘# lit-of ‘# mset* (*drop* (*literals-to-update-wl-heur S*) (*rev x1*)), *x2k*)›
‹*all-init-atms-st* (-, -, -, -, -, -, -, (*If* - - -) -, -)›] *isa-vmtf-consD*
clss-size-def clss-size-incr-lcountUE-def clss-size-incr-lcountUS-def
clss-size-incr-lcountU0-def
clss-size-decr-lcount-def clss-size-corr-restart-def empty-occs-list-cong[*of* ‹*all-init-atms-st* (-, -, -, -, -, -, -, (*If* - - -) -, -)›]
option-lookup-clause-rel-cong[*of*
‹*all-init-atms-st* (*x1*, *x1a*, *None*, *x1c*, *x1d*, *x1e*, *x1f*, *x1g*, *x1h*,
x1i, *x1j*, *uminus ‘# lit-of ‘# mset* (*drop* (*literals-to-update-wl-heur S*) (*rev x1*)), *x2k*)›
‹*all-init-atms-st* (-, -, -, -, -, -, -, (*If* - - -) -, -)›, *OF sym*] *outl*
set-conflict-to-false aivdom-inv-dec-mono
clss-size-def clss-size-incr-lcountUE-def learned-clss-count-def
clss-size-decr-lcount-def)
(*auto split*: *if-splits simp*:
clss-size-decr-lcount-def clss-size-lcount-def clss-size-lcountUS-def
clss-size-lcountU0-def clss-size-lcountUE-def clss-size-lcountUEk-def)
subgoal by *simp*
subgoal *state-conv* **for** *x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o x1p*
x2p x1q x2q x1r x2r x1s x2s x1t x2t x1u x2u
apply (*clarsimp simp only*: *twl-st-heur-restart-alt-def in-pair-collect-simp prod.simps*
prod-rel-iff TrueI refl accessors-def isasat-state-simp empty-occs-list-cong[*of* ‹*all-init-atms-st* (-, -, -, -, -, -,(*If* - - -) -, -)›]
cong[*of* ‹*all-init-atms-st* (*x1*, *x1a*, *None*, *x1c*, *x1d*, *x1e*, *x1f*, *x1g*, *x1h*,
x1i, *x1j*, *uminus ‘# lit-of ‘# mset* (*drop* (*literals-to-update-wl-heur S*) (*rev x1*)), *x2k*)›
‹*all-init-atms-st* (-, -, -, -, -, -, -, (*If* - - -) -, -)›] *isa-vmtf-consD*
clss-size-def clss-size-incr-lcountUE-def clss-size-incr-lcountUS-def
clss-size-incr-lcountU0-def aivdom-inv-dec-mono
clss-size-decr-lcount-def clss-size-corr-restart-def
option-lookup-clause-rel-cong[*of* ‹*all-init-atms-st* (*x1*, *x1a*, *None*, *x1c*, *x1d*, *x1e*, *x1f*, *x1g*, *x1h*,
x1i, *x1j*, *uminus ‘# lit-of ‘# mset* (*drop* (*literals-to-update-wl-heur S*) (*rev x1*)), *x2k*)›
‹*all-init-atms-st* (-, -, -, -, -, -, -, (*If* - - -) -, -)›, *OF sym*] *outl*
set-conflict-to-false)
apply (*auto split*: *if-splits*)
done
subgoal premises *p*
by (*rule state-conv*[*OF p*(*1*−*43*)])
done
qed

lemma *learned-clss-count-reset-units-since-last-GC-st*[*simp*]:
‹*learned-clss-count* (*reset-units-since-last-GC-st T*) =
learned-clss-count T›
‹(*reset-units-since-last-GC-st T*, *Ta*) ∈ *twl-st-heur-restart* ⟷
(*T*, *Ta*) ∈ *twl-st-heur-restart*›
‹*get-clauses-wl-heur* (*reset-units-since-last-GC-st T*) = *get-clauses-wl-heur T*›
by (*cases Ta*; *auto simp*: *reset-units-since-last-GC-st-def twl-st-heur-restart-def*; *fail*)+

lemma *isa-simplify-clauses-with-unit-st2-simplify-clauses-with-unit-st2*:
assumes ‹(*S*, *S'*) ∈ *twl-st-heur-restart-ana'* *r u*›
shows ‹*isa-simplify-clauses-with-unit-st2 S* ≤
⇓(*twl-st-heur-restart-ana'* *r u*) (*simplify-clauses-with-unit-st2 S'*)›
proof −
have *isa-simplify-clauses-with-unit-st2-def*: ‹*isa-simplify-clauses-with-unit-st2 S* =
do {
xs ← *RETURN* (*get-avdom S* @ *get-ivdom S*);
ASSERT(*length xs* ≤ *length* (*get-vdom S*) ∧ *length* (*get-vdom S*) ≤ *length* (*get-clauses-wl-heur S*));
T ← *do* {
(-, *T*) ← *WHILE*$_T$
(λ(*i*, *T*). *i* < *length xs* ∧ *get-conflict-wl-is-None-heur T*)
(λ(*i*,*T*). *do* {
(*T*) ←
do {
ASSERT((*i* < *length* (*get-avdom T*) ⟶ *access-avdom-at-pre T i*) ∧
(*i* ≥ *length* (*get-avdom T*) ⟶ *access-ivdom-at-pre T* (*i* − *length-avdom S*)) ∧
length-avdom T = *length-avdom S* ∧
length (*get-clauses-wl-heur T*) = *length* (*get-clauses-wl-heur S*) ∧
learned-clss-count T ≤ *learned-clss-count S*);
C ← *RETURN* (*if i* < *length* (*get-avdom S*) *then access-avdom-at T i else access-ivdom-at T*
(*i* − *length-avdom S*));
E ← *mop-arena-status* (*get-clauses-wl-heur T*) *C*;
if E ≠ *DELETED then do* {
isa-simplify-clause-with-unit-st2 C T
}
else RETURN (*T*)
};
ASSERT(*i* < *length xs*);
RETURN (*i*+*1*, *T*)})
(*0*, *S*);
RETURN T
};
RETURN (*reset-units-since-last-GC-st T*)
}›
unfolding *isa-simplify-clauses-with-unit-st2-def Let-def*
by (*auto simp*: *intro*!: *bind-cong*[*OF refl*] *cong*: *if-cong*)
have *simplify-clauses-with-unit-st2-def*:
‹*simplify-clauses-with-unit-st2 S* =
do {
ASSERT (*simplify-clauses-with-unit-st-wl-pre S*);
xs ← *SPEC*(λ*xs*. *finite xs*);
T ← *FOREACHci*(λ*it T*. *simplify-clauses-with-unit-st-wl-inv S it T*)
(*xs*)
(λ*S*. *get-conflict-wl S* = *None*)

```
      (λi S. let - =i; b = i ∈# dom-m (get-clauses-wl S) in
      if b then simplify-clause-with-unit-st2 i S else RETURN S)
     S;
     ASSERT(set-mset (all-learned-lits-of-wl T) ⊆ set-mset (all-learned-lits-of-wl S));
     ASSERT(set-mset (all-init-lits-of-wl T) = set-mset (all-init-lits-of-wl S));
     RETURN T
   }› for S
  unfolding simplify-clauses-with-unit-st2-def by (auto simp: Let-def)
 have dist-vdom: ‹distinct (get-vdom S)› and
   valid: ‹valid-arena (get-clauses-wl-heur S) (get-clauses-wl S') (set (get-vdom S))›
 using assms by (auto simp: twl-st-heur-restart-def twl-st-heur-restart-ana-def aivdom-inv-dec-alt-def)


 have vdom-incl: ‹set (get-vdom S) ⊆ {MIN-HEADER-SIZE..< length (get-clauses-wl-heur S)}›
  using valid-arena-in-vdom-le-arena[OF valid] arena-dom-status-iff[OF valid] by auto

 have le-vdom-arena: ‹length (get-vdom S) ≤ length (get-clauses-wl-heur S)›
  by (subst distinct-card[OF dist-vdom, symmetric])
   (use card-mono[OF - vdom-incl] in auto)

 have [refine]: ‹RETURN (get-avdom S@ get-ivdom S) ≤ ⇓ {(xs, a). a = set xs ∧ distinct xs ∧ set xs
⊆ set (get-vdom S) ∧
     xs = get-avdom S@ get-ivdom S} (SPEC (λxs. finite xs))›
  (is ‹- ≤ ⇓?A -›)
  using assms distinct-mset-mono[of ‹mset (get-avdom S)› ‹mset (get-vdom S)›]
  distinct-mset-mono[of ‹mset (get-ivdom S)› ‹mset (get-vdom S)›] apply -
  by (rule RETURN-RES-refine)
   (auto intro!:  simp: twl-st-heur-restart-def twl-st-heur-restart-ana-def aivdom-inv-dec-alt-def)
 let ?R = ‹{(a, b). (a, b) ∈ twl-st-heur-restart ∧ get-avdom a = get-avdom S ∧ get-vdom a = get-vdom
S∧
   get-ivdom a = get-ivdom S ∧
  length (get-clauses-wl-heur a) = r ∧ learned-clss-count a ≤ u ∧
       learned-clss-count a ≤ learned-clss-count S}›
  have [refine]: ‹(xs, xsa) ∈ ?A ⟹
   xsa ∈ {xs:: nat set. finite xs} ⟹
   ([0..<length xs], xsa) ∈ ⟨{(i, a). xs ! i =a ∧ i < length xs}⟩list-set-rel›
   (is ‹- ⟹ - ⟹ - ∈ ⟨?B⟩list-set-rel›)  for xs xsa
    by (auto simp: list-set-rel-def br-def
      intro!: relcompI[of - xs])
     (auto simp: list-rel-def intro!: list-all2-all-nthI)

  have H: ‹(xi, x) ∈ ?B xs ⟹
   (xi, x) ∈ {(i, a). xs ! i = a}› for xi x xs
    by auto
 have H2: ‹(si, s) ∈ ?R ⟹
   valid-arena (get-clauses-wl-heur si) (get-clauses-wl s) (set (get-vdom S))› for si s
   by (auto simp: twl-st-heur-restart-def)
  have H3: ‹(if xi < length (get-avdom S) then access-avdom-at si xi else access-ivdom-at si (xi −
length-avdom S), x)
  ∈ {(C,C'). (C,C')∈ nat-rel ∧ C ∈ set (get-vdom S)}› (is ‹- ∈ ?access›)
    if
     ‹(xs, xsa)
    ∈ {(xs, a). a = set xs ∧ distinct xs ∧ set xs ⊆ set (get-vdom S) ∧ xs = get-avdom S @ get-ivdom
S}› and
     ‹(xi, x) ∈ {(i, a). xs ! i = a ∧ i < length xs}› and
     ‹(si, s) ∈ ?R›
```

```
    for xs xsa x xi s si
    using that by (auto simp: access-ivdom-at-def access-avdom-at-def nth-append length-avdom-def)
  have H5: ‹(s,si)∈ ?R ⟹ (xi, x) ∈ ?B xs ⟹ (xs, xsa) ∈ ?A ⟹ (C,C') ∈ ?access ⟹
    (xa, C ∈# dom-m (get-clauses-wl si))
    ∈ {(a, b).
    (b ⟶
     (a = IRRED) = irred (get-clauses-wl si) (C) ∧
     (a = LEARNED) = (¬ irred (get-clauses-wl si) C)) ∧
    (a = DELETED) = (¬ b)} ⟹
    (xa, C' ∈# dom-m (get-clauses-wl si)) ∈ {(a, b).
    (b ⟶
     (a = IRRED) = irred (get-clauses-wl si) C' ∧
     (a = LEARNED) = (¬ irred (get-clauses-wl si) C')) ∧
    (a = DELETED) = (¬ b)}› for xi xs x s xa si xsa C C'
    by (auto simp: access-avdom-at-def)
  have H4: ‹(C,C')∈?access ⟹ (C,C')∈ nat-rel› for C C' by auto
  show ?thesis
    unfolding isa-simplify-clauses-with-unit-st2-def simplify-clauses-with-unit-st2-def
      nfoldli-upt-by-while[symmetric]
    unfolding nres-monad3
    apply (refine-vcg
      LFOci-refine[where R= ?R]
      mop-arena-status2[where vdom = ‹set (get-vdom S)›])
    subgoal using assms by (auto simp: twl-st-heur-restart-ana-def twl-st-heur-restart-def card-Un-Int
        aivdom-inv-dec-alt-def
      simp flip: distinct-card intro!: card-mono)
    subgoal using le-vdom-arena by auto
    subgoal
    by (subst get-conflict-wl-is-None-heur-get-conflict-wl-is-None-restart[THEN fref-to-Down-unRET-Id])
       (auto simp: get-conflict-wl-is-None-def)
    subgoal by (auto simp: access-avdom-at-pre-def)
    subgoal by (auto simp: access-ivdom-at-pre-def length-avdom-def less-diff-conv2)
    subgoal by (auto simp: length-avdom-def)
    subgoal using assms by (auto simp: twl-st-heur-restart-ana-def)
    subgoal by auto
    apply (rule H3; assumption)
    apply (rule H4; assumption)
    subgoal by auto
    apply (rule H2; assumption)
    apply (rule H5; assumption)
    subgoal by auto
    apply (rule isa-simplify-clause-with-unit-st2-simplify-clause-with-unit-st2[where
     u = ‹(get-avdom S)› and v = ‹(get-vdom S)› and x = ‹(get-ivdom S)› and r=r, THEN order-trans];
assumption?)
    apply (auto; fail)[]
    apply (auto; fail)[]
    subgoal
      by (clarsimp intro!: conc-fun-R-mono)
   subgoal using assms by (auto simp: twl-st-heur-restart-ana-def)
   subgoal by (auto simp: twl-st-heur-restart-ana-def reset-units-since-last-GC-def)
   done
qed


lemma simplify-clauses-with-units-st-wl2-simplify-clauses-with-units-st-wl:
  ‹(S, S') ∈ Id ⟹ simplify-clauses-with-units-st-wl2 S ≤ ⇓Id (simplify-clauses-with-units-st-wl S)›
```

unfolding *simplify-clauses-with-units-st-wl2-def simplify-clauses-with-units-st-wl-def*
by (*refine-vcg simplify-clauses-with-unit-st2-simplify-clauses-with-unit-st*) *auto*

lemma *isa-simplify-clauses-with-units-st2-simplify-clauses-with-units-st2*:
assumes ‹(*S*, *S'*) ∈ *twl-st-heur-restart-ana'* *r u*›
shows ‹*isa-simplify-clauses-with-units-st-wl2 S* ≤
⇓(*twl-st-heur-restart-ana'* *r u*) (*simplify-clauses-with-units-st-wl2 S'*)›
unfolding *isa-simplify-clauses-with-units-st-wl2-def simplify-clauses-with-units-st-wl2-def*
by (*refine-vcg isa-simplify-clauses-with-unit-st2-simplify-clauses-with-unit-st2*)
(*use assms* **in** ‹*auto simp*: *get-conflict-wl-is-None-def*
get-conflict-wl-is-None-heur-get-conflict-wl-is-None-ana[*THEN fref-to-Down-unRET-Id*]›)

end
theory *IsaSAT-Simplify-Units-LLVM*
imports
IsaSAT-Simplify-Clause-Units-LLVM
begin

lemma *isa-simplify-clauses-with-unit-st2-alt-def*:
‹*isa-simplify-clauses-with-unit-st2 S* =
do {
ASSERT(*length* (*get-avdom S*)+*length* (*get-ivdom S*) ≤ *length* (*get-vdom S*) ∧
length (*get-vdom S*) ≤ *length* (*get-clauses-wl-heur S*));
let m = *length* (*get-avdom S*);
let n = *length* (*get-ivdom S*);
let mn = *m*+*n*;
(-, *T*) ← *WHILE*$_T$
(λ(*i*, *T*). *i* < *mn* ∧ *get-conflict-wl-is-None-heur T*)
(λ(*i*,*T*). *do* {
ASSERT((*i* < *length* (*get-avdom T*) ⟶ *access-avdom-at-pre T i*) ∧
(*i* ≥ *length* (*get-avdom T*) ⟶ *access-ivdom-at-pre T* (*i* − *length-avdom S*)) ∧
length-avdom T = *length-avdom S* ∧
length (*get-clauses-wl-heur T*) = *length* (*get-clauses-wl-heur S*) ∧
learned-clss-count T ≤ *learned-clss-count S*);
let C = (*if i* < *m then access-avdom-at T i else access-ivdom-at T* (*i* − *m*));
E ← *mop-arena-status* (*get-clauses-wl-heur T*) *C*;
if E ≠ *DELETED then do* {
T ← *isa-simplify-clause-with-unit-st2 C T*;
ASSERT(*i* < *length* (*get-avdom S*)+*length* (*get-ivdom S*));
RETURN (*i*+*1*, *T*)
}
else do {*ASSERT*(*i* < *length* (*get-avdom S*)+*length* (*get-ivdom S*)); *RETURN* (*i*+*1*,*T*)}
})
(*0*, *S*);
RETURN (*reset-units-since-last-GC-st T*)
}›
unfolding *isa-simplify-clauses-with-unit-st2-def bind-to-let-conv Let-def length-avdom-def*
by (*auto cong*: *if-cong intro*!: *bind-cong*[*OF refl*])

sepref-register *length-ivdom access-ivdom-at*

sepref-register *isa-simplify-clauses-with-unit-st2*
sepref-def *isa-simplify-clauses-with-unit-st2-code*
is *isa-simplify-clauses-with-unit-st2*
:: ‹[λ*S*. *length* (*get-clauses-wl-heur S*) ≤ *snat64-max* ∧ *learned-clss-count S* ≤ *unat64-max*]$_a$

isasat-bounded-assn^d → *isasat-bounded-assn*›
unfolding *isa-simplify-clauses-with-unit-st2-alt-def*
length-avdom-def[*symmetric*] *Suc-eq-plus1*[*symmetric*] *length-ivdom-def*[*symmetric*]
mop-arena-status-st-def[*symmetric*]
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
supply [[*goals-limit=1*]]
by *sepref*

sepref-def *isa-simplify-clauses-with-units-st-wl2-code*
is *isa-simplify-clauses-with-units-st-wl2*
:: ‹[λ*S*. *length* (*get-clauses-wl-heur S*) ≤ *snat64-max* ∧ *learned-clss-count S* ≤ *unat64-max*]$_a$
isasat-bounded-assn^d → *isasat-bounded-assn*›
unfolding *isa-simplify-clauses-with-units-st-wl2-def*
supply [[*goals-limit=1*]]
by *sepref*

end
theory *IsaSAT-Simplify-Binaries-Defs*
imports *IsaSAT-Setup*
IsaSAT-Restart-Defs
IsaSAT-Simplify-Units-Defs
begin

type-synonym *'a ahm* = ‹*'a list* × *nat list*›

definition *ahm-get-marked* :: ‹*'a*::*zero ahm* ⇒ *nat literal* ⇒ -› **where**
‹*ahm-get-marked* = (λ(*C*,*stamp*) *L*. *do* {
ASSERT(*nat-of-lit L* < *length C*);
RETURN (*C*!*nat-of-lit L*)
})›

definition *get-marked* :: ‹(*nat literal* ⇒ *'a option*) × *nat* ⇒ *nat literal* ⇒ -› **where**
‹*get-marked C L* = *do* {
ASSERT(*nat-of-lit L* < *snd C* ∧ *fst C L* ≠ *None*);
RETURN (*the* (*fst C L*))
}›

definition *array-hash-map-rel* :: ‹(*'a* :: *zero* × *'b*) *set* ⇒ (*'a ahm* × -) *set*› **where**
‹*array-hash-map-rel R* = {((*xs*, *support*), (*ys*, *m*)). *m* = *length xs* ∧
(*set support* =*nat-of-lit* ‘ *dom ys*) ∧
(∀ *i*∈*set support*. *i* < *m*) ∧ *distinct support* ∧
(∀ *L*. *nat-of-lit L* < *m* ⟶ (*ys L* = *None* ⟷ *xs* ! *nat-of-lit L* = *0*)) ∧
(∀ *L*. *nat-of-lit L* < *m* ⟶ (∀ *a*. *ys L* = *Some a* ⟶ *xs* ! *nat-of-lit L* ≠ *0* ∧ (*xs* ! *nat-of-lit L*, *a*) ∈ *R*))}›

definition *ahm-is-marked* :: ‹*'a*::*zero ahm* ⇒ *nat literal* ⇒ -› **where**
‹*ahm-is-marked* = (λ(*C*,*stamp*) *L*. *do* {
ASSERT(*nat-of-lit L* < *length C*);
RETURN (*C*!*nat-of-lit L* ≠ *0*)
})›
definition *is-marked* :: ‹(*nat literal* ⇒ *'a option*) × *nat* ⇒ *nat literal* ⇒ -› **where**
‹*is-marked C L* = *do* {
ASSERT(*nat-of-lit L* < *snd C*);

```
    RETURN (fst C L ≠ None)
  }›

definition update-marked :: ‹(nat literal ⇒ - option) × nat ⇒ nat literal ⇒ - ⇒
  ((nat literal ⇒ - option) × nat) nres› where
  ‹update-marked C L v = do {
    ASSERT (nat-of-lit L < snd C ∧ (fst C)L ≠ None);
    RETURN ((fst C)(L := Some v), snd C)
  }›

definition set-marked :: ‹(nat literal ⇒ - option) × nat ⇒ nat literal ⇒ - ⇒
  ((nat literal ⇒ - option) × nat) nres› where
  ‹set-marked C L v = do {
    ASSERT (nat-of-lit L < snd C ∧ (fst C)L = None);
    RETURN ((fst C)(L := Some v), snd C)
  }›



definition empty :: ‹(nat literal ⇒ 'b option) × nat ⇒ ((nat literal ⇒ 'b option) × nat) nres› where
  ‹empty = (λ(-, n). do {
    RETURN (λ-. None, n)
  })›

lemma ahm-is-marked-is-marked:
   ‹(uncurry ahm-is-marked, uncurry is-marked)
    ∈ (array-hash-map-rel R) ×f nat-lit-lit-rel → ⟨bool-rel⟩nres-rel›
  unfolding ahm-is-marked-def is-marked-def uncurry-def
  apply (intro frefI nres-relI fun-relI)
  apply refine-vcg
  apply (auto simp: array-hash-map-rel-def ahm-is-marked-def is-marked-def
    intro!: ASSERT-leI)[]
  apply simp
  by (auto simp: array-hash-map-rel-def ahm-is-marked-def is-marked-def
    intro!: ASSERT-leI)



lemma ahm-get-marked-get-marked:
   ‹(uncurry ahm-get-marked, uncurry get-marked)
    ∈ (array-hash-map-rel R) ×f nat-lit-lit-rel → ⟨R⟩nres-rel›
  unfolding ahm-get-marked-def get-marked-def uncurry-def
  apply (intro frefI nres-relI fun-relI)
  apply refine-vcg
  apply (auto simp: array-hash-map-rel-def ahm-is-marked-def is-marked-def
    intro!: ASSERT-leI)[]
  apply clarsimp
  apply (auto simp: array-hash-map-rel-def ahm-is-marked-def is-marked-def
    intro!: ASSERT-leI)[]
  done


definition ahm-set-marked :: ‹'a :: zero ahm ⇒ nat literal ⇒ -› where
  ‹ahm-set-marked= (λ(C,stamp) L v. do {
    ASSERT(nat-of-lit L < length C ∧ Suc (length stamp) ≤ length C);
```

```
    RETURN (C[nat-of-lit L := v], stamp @ [nat-of-lit L])
  })›


lemma ahm-set-marked-set-marked:
  assumes [simp]: ‹⋀a. (0, a) ∉ R›
  shows ‹(uncurry2 ahm-set-marked, uncurry2 set-marked)
    ∈ (array-hash-map-rel R) ×f nat-lit-lit-rel ×f R →f ⟨array-hash-map-rel R⟩nres-rel›
proof −
  have [intro!]: ‹Suc (length x2b) ≤ length x1d›
    if
      ‹nat-of-lit x2 < length x1d› and
      ‹x1d ! nat-of-lit x2 = 0› and
      ‹set x2b = nat-of-lit ‘ {a. ∃ y. ac a = Some y}› and
      ‹∀ i. (∃ y. ac i = Some y) ⟶ nat-of-lit i < length x1d› and
      dist: ‹distinct x2b› and
      ‹∀ L. nat-of-lit L < length x1d ⟶ (ac L = None) = (x1d ! nat-of-lit L = 0)›
    for ac :: ‹nat literal ⇒ 'a option› and x2 :: ‹nat literal› and x2a :: 'a and x1d :: ‹'b list› and x2b
:: ‹nat list› and x2d :: 'b
  proof −
    have ‹(set x2b) ⊆ nat-of-lit ‘ (dom ac) ∩ {0..<length x1d}›
      using that by (auto simp add: dom-def)
    moreover have ‹nat-of-lit x2 ∈ {0..<length x1d}› and notin: ‹nat-of-lit x2 ∉ set x2b›
      using that by auto
    ultimately have H: ‹insert (nat-of-lit x2) (set x2b) ⊆ {0..<length x1d}›
      by blast
    show ?thesis
      using card-mono[OF - H] notin dist
      by (auto simp: distinct-card)
  qed
  show ?thesis
    unfolding ahm-set-marked-def set-marked-def uncurry-def
    apply (intro frefI fun-relI nres-relI)
    apply refine-vcg
    apply (auto simp: array-hash-map-rel-def ahm-is-marked-def is-marked-def dom-def
      intro!: ASSERT-leI)
   done
qed


definition ahm-update-marked :: ‹'a :: zero ahm ⇒ nat literal ⇒ -› where
  ‹ahm-update-marked= (λ(C,stamp) L v. do {
     ASSERT(nat-of-lit L < length C);
     RETURN (C[nat-of-lit L := v], stamp)
  })›


lemma ahm-update-marked-update-marked:
  assumes [simp]: ‹⋀a. (0, a) ∉ R›
  shows ‹(uncurry2 ahm-update-marked, uncurry2 update-marked)
    ∈ (array-hash-map-rel R) ×f nat-lit-lit-rel ×f R →f ⟨array-hash-map-rel R⟩nres-rel›
  unfolding ahm-update-marked-def update-marked-def uncurry-def
  apply (intro frefI nres-relI)
  by refine-vcg
   (auto simp: array-hash-map-rel-def ahm-is-marked-def is-marked-def dom-def
    intro!: ASSERT-leI)
```

definition *ahm-create* :: ‹*nat* ⇒ *'a*::*zero ahm nres*› **where**
‹*ahm-create m* = *do* {
RETURN (*replicate m 0*, [])
}›

definition *create* :: ‹*nat* ⇒ -› **where**
‹*create m* = *do* {
RETURN (λ-. *None*, *m*)
}›

lemma *ahm-create-create*:
‹(*ahm-create*, *create*) ∈ *nat-rel* → ⟨*array-hash-map-rel R*⟩*nres-rel*›
unfolding *ahm-create-def create-def uncurry-def*
by *refine-vcg*
(*auto simp*: *array-hash-map-rel-def ahm-is-marked-def is-marked-def*
intro!: *ASSERT-leI*)[]

definition *ahm-empty* :: ‹*'a*::*zero ahm* ⇒ *'a ahm nres*› **where**
‹*ahm-empty* = (λ(*CS*, *support*). *do* {
let n = *length support*;
(-, *CS*) ← *WHILE*$_T$
(λ(*i*, *CS*). *i* < *n*)
(λ(*i*, *CS*). *do* {*ASSERT* (*i* < *length support* ∧ *support* ! *i* < *length CS*); *RETURN* (*i*+*1*, *CS*[*support* ! *i* := *0*])})
(*0*, *CS*);
RETURN (*CS*, *take 0 support*)
})›

lemma *ahm-empty-empty*:
‹(*ahm-empty*, *empty*) ∈ (*array-hash-map-rel R*) → ⟨*array-hash-map-rel R*⟩*nres-rel*›
proof −
have [*simp*]: ‹*dom* (λ*aa*. *if nat-of-lit aa* ∈ *xs then None else m aa*) =
dom m − {*a*. *nat-of-lit a* ∈ *xs*}› **for** *xs m*
by (*auto split*: *if-splits*)
have [*simp*]: ‹*nat-of-lit* ‘ *dom x1* = *set x2a* ⟹ *distinct x2a* ⟹
nat-of-lit ‘ (*dom x1* − {*aa*. *nat-of-lit aa* ∈ *set* (*take a x2a*)}) = *set* (*drop a x2a*)› **for** *x2a a x1*
apply (*drule eq-commute*[*of* - *set x2a*, *THEN iffD1*])
apply (*auto simp*: *dom-def*)
apply (*metis* (*mono-tags*, *lifting*) *Reversed-Bit-Lists.atd-lem UnE image-subset-iff mem-Collect-eq set-append set-mset-mono set-mset-mset subset-mset.dual-order.refl*)
apply (*frule in-set-dropD*)
apply *simp*
by (*smt* (*verit*, *ccfv-threshold*) *DiffI IntI Reversed-Bit-Lists.atd-lem distinct-append empty-iff image-iff mem-Collect-eq*)

show *?thesis*
unfolding *ahm-empty-def empty-def uncurry-def fref-param1*
apply (*intro ext frefI nres-relI*)
subgoal for *x y*
apply (*refine-vcg WHILET-rule*[**where** *I* = ‹λ(*i*, *CS*). ((*CS*, *drop i* (*snd x*)), (λ*a*. *if nat-of-lit a* ∈ *set* (*take i* (*snd x*)) *then None else fst y a*, *snd y*)) ∈*array-hash-map-rel R*› **and**
R = ‹*measure* (λ(*i*,-). *length* (*snd x*) −*i*)›])

```
      subgoal by auto
      subgoal by auto
      subgoal by (clarsimp simp: array-hash-map-rel-def)
      subgoal by (auto simp: array-hash-map-rel-def)
      subgoal premises p for x1 x2 x1a x2a s a b x1b x2b
        using p
        by (clarsimp simp: array-hash-map-rel-def less-Suc-eq  eq-commute[of - ‹nat-of-lit ‘ dom -›]
          simp flip: Cons-nth-drop-Suc)
          (auto simp: take-Suc-conv-app-nth)
      subgoal by (auto simp: nth-list-update' less-Suc-eq array-hash-map-rel-def)
      subgoal for x1 x2 x1a x2a s a b
        apply (auto simp: array-hash-map-rel-def)
        apply (drule-tac x=L in spec)
        apply (auto split: if-splits)
        done
      done
    done
qed

definition isa-clause-remove-duplicate-clause-wl :: ‹nat ⇒ isasat ⇒ isasat nres› where
  ‹isa-clause-remove-duplicate-clause-wl C S = (do{
    - ← log-del-clause-heur S C;
    let N' = get-clauses-wl-heur S;
    st ← mop-arena-status N' C;
    let st = st = IRRED;
    ASSERT (mark-garbage-pre (N', C) ∧ arena-is-valid-clause-vdom (N') C);
    let N' = extra-information-mark-to-delete (N') C;
    let lcount = get-learned-count S;
    ASSERT(¬st ⟶ clss-size-lcount lcount ≥ 1);
    let lcount = (if st then lcount else (clss-size-decr-lcount lcount));
    let stats = get-stats-heur S;
    let stats = (incr-binary-red-removed (if st then decr-irred-clss stats else stats));
    let S = set-clauses-wl-heur N' S;
    let S = set-learned-count-wl-heur lcount S;
    let S = set-stats-wl-heur stats S;
    RETURN S
   })›

definition isa-binary-clause-subres-lits-wl-pre :: ‹-› where
  ‹isa-binary-clause-subres-lits-wl-pre C L L' S ⟷
    (∃ T r u. (S, T)∈ twl-st-heur-restart-ana' r u ∧  binary-clause-subres-lits-wl-pre C L L' T)›

definition isa-binary-clause-subres-wl :: ‹-› where
  ‹isa-binary-clause-subres-wl C L L' S = do {
      ASSERT (isa-binary-clause-subres-lits-wl-pre C L L' S);
      let - = log-del-binary-clause L (−L');
      let M = get-trail-wl-heur S;
      M ← cons-trail-Propagated-tr L 0 M;
      let lcount = get-learned-count S;
      let N' = get-clauses-wl-heur S;
      st ← mop-arena-status N' C;
      let st = st = IRRED;
      ASSERT (mark-garbage-pre (N', C) ∧ arena-is-valid-clause-vdom (N') C);
      let N' = extra-information-mark-to-delete (N') C;
      ASSERT(¬st ⟶ (clss-size-lcount lcount ≥ 1 ∧ clss-size-lcountUEk (clss-size-decr-lcount lcount)
< learned-clss-count S));
```

```
    let lcount = (if st then lcount else (clss-size-incr-lcountUEk (clss-size-decr-lcount lcount)));
    let stats = get-stats-heur S;
    let stats = incr-binary-unit-derived (if st then decr-irred-clss stats else stats);
    let stats = incr-units-since-last-GC (incr-uset stats);
    let S = set-trail-wl-heur M S;
    let S = set-clauses-wl-heur N' S;
    let S = set-learned-count-wl-heur lcount S;
    let S = set-stats-wl-heur stats S;
    RETURN S
  }›

definition isa-deduplicate-binary-clauses-wl :: ‹nat literal ⇒ - ⇒ isasat ⇒ (- × isasat) nres› where
‹isa-deduplicate-binary-clauses-wl L CS S0 = do {
    let CS = CS;
    l ← mop-length-watched-by-int S0 L;
    ASSERT (l ≤ length (get-clauses-wl-heur S0) − 2);
    val ← mop-polarity-pol (get-trail-wl-heur S0) L;
    (-, -, CS, S) ← WHILE_T(λ(abort, i, CS, S). ¬abort ∧ i < l ∧ get-conflict-wl-is-None-heur S)
      (λ(abort, i, CS, S).
      do {
        ASSERT (i < l);
        ASSERT (length (get-clauses-wl-heur S) = length (get-clauses-wl-heur S0));
        ASSERT (learned-clss-count S ≤ learned-clss-count S0);
        (C,L', b) ← mop-watched-by-app-heur S L i;
        ASSERT (C > 0 ∧ C < length (get-clauses-wl-heur S));
        st ← mop-arena-status (get-clauses-wl-heur S) C;
        if st = DELETED ∨ ¬b then
          RETURN (abort, i+1, CS, S)
        else do {
          val ← mop-polarity-pol (get-trail-wl-heur S) L';
          if val ≠ UNSET then do {
            S ← isa-simplify-clause-with-unit-st2 C S;
            val ← mop-polarity-pol (get-trail-wl-heur S) L;
            RETURN (val ≠ UNSET, i+1, CS, S)
          }
          else do {
            m ← is-marked CS (L');
            n ← (if m then get-marked CS L' else RETURN (1, True));
            if m then do {
              let C' = (if ¬snd n ⟶ st = LEARNED then C else fst n);
              CS ← (if ¬snd n ⟶ st = LEARNED then RETURN CS else update-marked CS (L') (C,
st = IRRED));
              S ← isa-clause-remove-duplicate-clause-wl C' S;
              RETURN (abort, i+1, CS, S)
            } else do {
              m ← is-marked CS (−L') ;
              if m then do {
                S ← isa-binary-clause-subres-wl C L (−L') S;
                RETURN (True, i+1, CS, S)
              }
              else do {
                CS ← set-marked CS (L') (C, st = IRRED);
                RETURN (abort, i+1, CS, S)
              }
            }
          }
        }
```

```
      }
    })
    (val ≠ UNSET, 0, CS, S₀);
  CS ← empty CS;
  RETURN (CS, S)
}›

definition mark-duplicated-binary-clauses-as-garbage-pre-wl-heur :: ‹isasat ⇒ bool› where
  ‹mark-duplicated-binary-clauses-as-garbage-pre-wl-heur S ⟷
  (∃ S' r u. (S, S') ∈ twl-st-heur-restart-ana' r u ∧
    mark-duplicated-binary-clauses-as-garbage-pre-wl S')›

definition isa-mark-duplicated-binary-clauses-as-garbage-wl :: ‹isasat ⇒ - nres› where
  ‹isa-mark-duplicated-binary-clauses-as-garbage-wl S₀ = (do {
    let ns = (get-vmtf-heur-array S₀);
    ASSERT (mark-duplicated-binary-clauses-as-garbage-pre-wl-heur S₀);
    skip ← RETURN (should-eliminate-pure-st S₀);
    CS ← create (length (get-watched-wl-heur S₀));
   (-, CS, S) ← WHILE_T λ(n,CS,S). ns = (get-vmtf-heur-array S)(λ(n, CS,S). n ≠ None ∧ get-conflict-wl-is-None-heur
S)
     (λ(n, CS, S). do {
       ASSERT (n ≠ None);
       let A = the n;
       ASSERT (A < length ns);
       ASSERT (A ≤ unat32-max div 2);
       S ← do {ASSERT (ns = (get-vmtf-heur-array S));
       - ← mop-is-marked-added-heur-st S A;
       if ¬skip then RETURN (CS, S)
       else do {
         ASSERT (length (get-clauses-wl-heur S) ≤ length (get-clauses-wl-heur S₀) ∧ learned-clss-count
S ≤ learned-clss-count S₀);
         (CS, S) ← isa-deduplicate-binary-clauses-wl (Pos A) CS S;
         ASSERT (length (get-clauses-wl-heur S) ≤ length (get-clauses-wl-heur S₀) ∧ learned-clss-count
S ≤ learned-clss-count S₀);
         (CS, S) ← isa-deduplicate-binary-clauses-wl (Neg A) CS S;
         ASSERT (ns = (get-vmtf-heur-array S));
         RETURN (CS, S)
       }};
       RETURN (get-next (ns ! A), S)
    })
    (Some (get-vmtf-heur-fst S₀), CS, S₀);
   RETURN S
  })›


definition isa-mark-duplicated-binary-clauses-as-garbage-wl2 :: ‹isasat ⇒ - nres› where
  ‹isa-mark-duplicated-binary-clauses-as-garbage-wl2 S₀ = (do {
    let ns = get-vmtf-heur-array S₀;
    ASSERT (mark-duplicated-binary-clauses-as-garbage-pre-wl-heur S₀);
    dedup ← RETURN (should-eliminate-pure-st S₀);
    CS ← create (length (get-watched-wl-heur S₀));
   (-, CS, S) ← WHILE_T λ(n,CS, S). get-vmtf-heur-array S₀ = (get-vmtf-heur-array S)(λ(n, CS, S). n ≠
None ∧ get-conflict-wl-is-None-heur S)
     (λ(n, CS, S). do {
       ASSERT (n ≠ None);
```

let A = the n;
ASSERT (*A* < *length* (*get-vmtf-heur-array S*));
ASSERT (*A* ≤ *unat32-max div 2*);
(*CS*, *S*) ← *do* {
- ← *mop-is-marked-added-heur-st S A*;
if ¬*dedup then RETURN* (*CS*, *S*)
else do {
ASSERT (*length* (*get-clauses-wl-heur S*) ≤ *length* (*get-clauses-wl-heur* S_0) ∧ *learned-clss-count S* ≤ *learned-clss-count* S_0);
(*CS*, *S*) ← *isa-deduplicate-binary-clauses-wl* (*Pos A*) *CS S*;
ASSERT (*length* (*get-clauses-wl-heur S*) ≤ *length* (*get-clauses-wl-heur* S_0) ∧ *learned-clss-count S* ≤ *learned-clss-count* S_0);
(*CS*, *S*) ← *isa-deduplicate-binary-clauses-wl* (*Neg A*) *CS S*;
ASSERT (*ns* = *get-vmtf-heur-array S*);
RETURN (*CS*, *S*)
}};
RETURN (*get-next* (*get-vmtf-heur-array S* ! *A*), *CS*, *S*)
})
(*Some* (*get-vmtf-heur-fst* S_0), *CS*, S_0);
RETURN S
})›

end
theory *IsaSAT-Simplify-Binaries*
imports *IsaSAT-Setup*
Watched-Literals.Watched-Literals-Watch-List-Inprocessing
More-Refinement-Libs.WB-More-Refinement-Loops
IsaSAT-Simplify-Binaries-Defs
IsaSAT-Simplify-Units
IsaSAT-Restart
begin

22.1 Simplification of binary clauses

Special handling of binary clauses is required to avoid special cases of units in the general forward subsumption algorithm (which, as of writing, does not exist).

lemma *all-atms-st-add-remove*[*simp*]:
‹*C* ∈# *dom-m N* ⟹ *all-atms-st* (*M*, *fmdrop C N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *add-mset* (*mset* (*N* ∝ *C*)) *NS*, *US*, *N0*, *U0*, *Q*, *W*) =
all-atms-st (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*)›
‹*C* ∈# *dom-m N* ⟹ *all-atms-st* (*M*, *fmdrop C N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *add-mset* (*mset* (*N* ∝ *C*)) *US*, *N0*, *U0*, *Q*, *W*) =
all-atms-st (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*)›
‹*C* ∈# *dom-m N* ⟹ *all-atms-st* (*M*, *fmdrop C N*, *D*, *NE*, *UE*, *add-mset* (*mset* (*N* ∝ *C*)) *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*) =
all-atms-st (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*)›
‹*C* ∈# *dom-m N* ⟹ *all-atms-st* (*M*, *fmdrop C N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *add-mset* (*mset* (*N* ∝ *C*)) *NS*, *US*, *N0*, *U0*, *Q*, *W*) =
all-atms-st (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*)›
‹*all-atms-st* (*L* # *M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*) = *all-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*)›
‹*all-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *add-mset K Q*, *W*) = *all-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*)›
apply (*auto simp*: *all-atms-st-def all-atms-def all-lits-def all-lits-of-mm-union all-lits-of-mm-add-mset*

distinct-mset-remove1-All dest!: *multi-member-split*
simp del: *all-atms-def*[*symmetric*])
by (*metis* (*no-types*, *lifting*) *Watched-Literals-Clauses.ran-m-fmdrop Watched-Literals-Clauses.ran-m-mapsto-upd all-lits-of-mm-add-mset*
fmupd-same image-mset-add-mset image-mset-union union-single-eq-member)+

lemma *all-atms-st-add-kep*[*simp*]:
‹*L* ∈# $\mathcal{L}_{all}$ (*all-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*)) ⟹
set-mset (*all-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *add-mset* {#*L*#} *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*))
= *set-mset* (*all-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*))›
‹*L* ∈# $\mathcal{L}_{all}$ (*all-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*)) ⟹
set-mset (*all-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *add-mset* {#*L*#} *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*))
= *set-mset* (*all-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*))›
by (*auto simp*: *all-atms-st-def in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *all-atms-def all-lits-def all-lits-of-mm-union all-lits-of-mm-add-mset*
all-lits-of-m-add-mset
simp del: *all-atms-def*[*symmetric*])

lemma *clss-size-corr-in-dom-red-clss-size-lcount-ge0*:
‹*C* ∈# *dom-m N* ⟹ ¬*irred N C* ⟹ *clss-size-corr N NE UE NEk UEk NS US N0 U0 lcount* ⟹
clss-size-lcount lcount ≥ *Suc 0*›
apply (*auto dest*!: *multi-member-split simp*: *clss-size-corr-def clss-size-def*)
by (*metis member-add-mset red-in-dom-number-of-learned-ge1*)

abbreviation *twl-st-heur-restart-ana''* :: ‹-› **where**
‹*twl-st-heur-restart-ana'' r u ns lw* ≡
{(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart-ana r* ∧ *learned-clss-count S* ≤ *u* ∧ *get-vmtf-heur S* = *ns* ∧
length (*get-watched-wl-heur S*) = *lw*}›

lemma *isa-clause-remove-duplicate-clause-wl-clause-remove-duplicate-clause-wl*:
‹(*uncurry isa-clause-remove-duplicate-clause-wl*, *uncurry clause-remove-duplicate-clause-wl*) ∈ [λ(*C*, *S*). *C* ∈# *dom-m* (*get-clauses-wl S*)]$_f$
nat-rel ×$_f$ *twl-st-heur-restart-ana'' r u ns lw* →
⟨*twl-st-heur-restart-ana'' r u ns lw*⟩*nres-rel*›
proof −
show *?thesis*
supply [[*goals-limit=1*]]
unfolding *isa-clause-remove-duplicate-clause-wl-def clause-remove-duplicate-clause-wl-def uncurry-def*
mop-arena-status-def nres-monad3
apply (*intro frefI nres-relI*)
apply (*refine-vcg log-del-clause-heur-log-clause*[*THEN order-trans*])
apply (*solves auto*)[]
apply (*solves auto*)[]
apply (*solves auto*)[]
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i x1j x2j x1k a b c d e*
unfolding *arena-is-valid-clause-vdom-def*
apply (*rule exI*[*of* - ‹*get-clauses-wl x2*›], *rule exI*[*of* - ‹*set* (*get-vdom e*)›])
by (*simp add*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*)
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i x1j x2j x1k a b c d e*
unfolding *mark-garbage-pre-def arena-is-valid-clause-idx-def prod.simps*
apply (*rule exI*[*of* - ‹*get-clauses-wl x2*›], *rule exI*[*of* - ‹*set* (*get-vdom e*)›])
by (*simp add*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*)
subgoal for *x y x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i x1j x2j x1k*

x2k x1l x2l x1m x2m
by (*auto simp*: *clause-remove-duplicate-clause-wl-pre-def clause-remove-duplicate-clause-pre-def state-wl-l-def red-in-dom-number-of-learned-ge1*
twl-st-heur-restart-def twl-st-heur-restart-ana-def clss-size-corr-in-dom-red-clss-size-lcount-ge0 arena-lifting clss-size-corr-restart-def)
subgoal
by (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def valid-arena-extra-information-mark-to-delete' aivdom-inv-dec-remove-clause arena-lifting*
all-init-atms-fmdrop-add-mset-unit learned-clss-count-def
dest!: *in-vdom-m-fmdropD*)
done
qed

lemma [*simp*]: ‹(*S*, *x*) ∈ *state-wl-l None* ⟹
defined-lit (*get-trail-l x*) *L* ⟷ *defined-lit* (*get-trail-wl S*) *L*›
by (*auto simp*: *state-wl-l-def*)

lemma *binary-clause-subres-wl-alt-def*:
‹*binary-clause-subres-wl C L L'* = (λ(*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*). *do* {
ASSERT (*binary-clause-subres-lits-wl-pre C L L'* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*));
ASSERT (*L'* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*)));
ASSERT (*L* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*)));
ASSERT (*get-conflict-wl* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*) = *None* ∧ *undefined-lit M L* ∧ *C* ∈# *dom-m N*);
M' ← *cons-trail-propagate-l L 0 M*;
ASSERT (*M'* = *Propagated L 0* # *M*);
let S = (*M'*, *fmdrop C N*, *D*, *NE*, *UE*,
(*if irred N C then add-mset* {#*L*#} *else id*) *NEk*, (*if irred N C then id else add-mset* {#*L*#}) *UEk*,
(*if irred N C then add-mset* (*mset* (*N* ∝ *C*)) *else id*) *NS*, (*if irred N C then id else add-mset* (*mset* (*N* ∝ *C*))) *US*,
N0, *U0*, *add-mset* (−*L*) *Q*, *W*);
ASSERT (*set-mset* (*all-init-atms-st* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*)) = *set-mset* (*all-init-atms-st S*));
RETURN S
})› (**is** ‹*?A* = *?B*›)
proof −
have *H*: ‹*binary-clause-subres-lits-wl-pre C L L' S* ⟹ *set-mset* (*all-atms-st S*) = *set-mset* (*all-init-atms-st S*)› **for** *S*
unfolding *binary-clause-subres-lits-wl-pre-def binary-clause-subres-lits-pre-def*
apply *normalize-goal*+
using *literals-are-*$\mathcal{L}_{in}$*'-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*) **by** *fast*
have *H*: ‹*binary-clause-subres-lits-wl-pre C L L' S* ⟷ *binary-clause-subres-lits-wl-pre C L L' S* ∧ *L'* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st S*) ∧ *L* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st S*) ∧
undefined-lit (*get-trail-wl S*) *L* ∧ *get-conflict-wl S* = *None* ∧ *C* ∈# *dom-m* (*get-clauses-wl S*)› **for** *S*
apply (*rule iffI*)
apply *simp-all*
apply (*frule* $\mathcal{L}_{all}$*-cong*[*OF H*, *symmetric*])
unfolding *binary-clause-subres-lits-wl-pre-def binary-clause-subres-lits-pre-def binary-clause-subres-lits-pre-def*
apply *normalize-goal*+
apply (*simp add*:)
by (*cases S*; *auto simp*: *all-atms-st-def* $\mathcal{L}_{all}$*-all-atms-all-lits all-lits-def ran-m-def all-lits-of-mm-add-mset all-lits-of-m-add-mset*

dest!: *multi-member-split*)
have [*simp*]: ‹*L* ∈# *all-init-lits x1a* (*x1c* + *x1e* + *x1g* + *x1i*) ⟹
set-mset (*all-init-atms x1a* (*add-mset* {#*L*#} (*x1c* + *x1e* + *x1g* + *x1i*))) = *set-mset* (*all-init-atms x1a* ((*x1c* + *x1e* + *x1g* + *x1i*)))› **for** *x1a x1c x1e x1g x1i*
by (*auto simp*: *all-init-atms-def all-init-lits-def all-lits-of-mm-add-mset all-lits-of-m-add-mset*
simp del: *all-init-atms-def*[*symmetric*])
have ‹*?A S* ≤ ⇓*Id* (*?B S*)› **for** *S*
unfolding *binary-clause-subres-wl-def summarize-ASSERT-conv cons-trail-propagate-l-def nres-monad3 Let-def bind-to-let-conv*
apply (*subst* (*2*) *H*)
by *refine-vcg auto*
moreover have ‹*?B S* ≤ ⇓*Id* (*?A S*)› **for** *S*
unfolding *binary-clause-subres-wl-def summarize-ASSERT-conv cons-trail-propagate-l-def nres-monad3 Let-def bind-to-let-conv*
apply (*subst* (*2*) *H*)
by *refine-vcg*
(*auto simp*: *all-init-atms-st-def all-init-atms-fmdrop-add-mset-unit* $\mathcal{L}_{all}$*-all-init-atms*)
ultimately show *?thesis*
unfolding *Down-id-eq*
by (*intro ext*, *rule antisym*)
qed

lemma *isa-binary-clause-subres-isa-binary-clause-subres-wl*:
‹(*uncurry3 isa-binary-clause-subres-wl*, *uncurry3 binary-clause-subres-wl*)
∈ *nat-rel* ×$_f$ *nat-lit-lit-rel* ×$_f$ *nat-lit-lit-rel* ×$_f$ *twl-st-heur-restart-ana''* *r u ns lw* →$_f$ ⟨*twl-st-heur-restart-ana''* *r u ns lw*⟩*nres-rel*›
proof −
have *A*: ‹*A* ∈ *twl-st-heur-restart-ana''* *r u ns lw* ⟹ *A* ∈ *twl-st-heur-restart-ana''* *r u ns lw*› **for** *A*
by *auto*
note *cong* = *trail-pol-cong option-lookup-clause-rel-cong map-fun-rel-D*$_0$*-cong isa-vmtf-cong phase-saving-cong cach-refinement-empty-cong' vdom-m-cong' vdom-m-cong'' isasat-input-bounded-cong*[*THEN iffD1*] *isasat-input-nempty-cong*[*THEN iffD1*]
heuristic-rel-cong empty-occs-list-cong'
show *?thesis*
unfolding *isa-binary-clause-subres-wl-def binary-clause-subres-wl-alt-def uncurry-def Let-def mop-arena-status-def nres-monad3*
apply (*intro frefI nres-relI*)
subgoal for *S T*
apply (*refine-vcg cons-trail-Propagated-tr*[*of* ‹*all-init-atms-st* (*snd T*)›, *THEN fref-to-Down-curry2*])
subgoal unfolding *isa-binary-clause-subres-lits-wl-pre-def twl-st-heur-restart-ana-def* **by** *force*
subgoal by *auto*
subgoal by (*auto simp*: *DECISION-REASON-def*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def all-init-atms-st-def*)
subgoal for *x1 x1a x1b x2 x2a x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x1p x1q x2o x2p x2q M M'*
unfolding *arena-is-valid-clause-vdom-def*
apply (*rule exI*[*of* - ‹*get-clauses-wl x2b*›], *rule exI*[*of* - ‹*set* (*get-vdom x2q*)›])
by (*simp add*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*)
subgoal for *x1 x1a x1b x2 x2a x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x1p x1q x2o x2p x2q M M'*
unfolding *mark-garbage-pre-def arena-is-valid-clause-idx-def prod.simps*
by (*rule exI*[*of* - ‹*get-clauses-wl x2b*›], *rule exI*[*of* - ‹*set* (*get-vdom x2q*)›])
(*simp add*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*)
subgoal *H*
by (*auto simp*: *clause-remove-duplicate-clause-wl-pre-def clause-remove-duplicate-clause-pre-def state-wl-l-def red-in-dom-number-of-learned-ge1*

twl-st-heur-restart-def twl-st-heur-restart-ana-def clss-size-corr-in-dom-red-clss-size-lcount-ge0 arena-lifting
clss-size-corr-restart-def)
subgoal by (*frule H*; *assumption?*) (*auto simp*: *learned-clss-count-def*)
apply (*rule A*)
subgoal premises *p*
using *p*
apply (*simp only*: *twl-st-heur-restart-alt-def2 Let-def twl-st-heur-restart-ana-def in-pair-collect-simp prod.simps prod-rel-fst-snd-iff get-trail-wl.simps fst-conv snd-conv*)
apply *normalize-goal+*
apply (*drule cong*[*OF p*(*28*)])+
apply (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def valid-arena-extra-information-mark-to-delete' aivdom-inv-dec-remove-clause*
arena-lifting isa-vmtf-consD all-init-atms-st-def
dest!: *in-vdom-m-fmdropD*)
apply (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def valid-arena-extra-information-mark-to-delete' aivdom-inv-dec-remove-clause*
arena-lifting isa-vmtf-consD clss-size-corr-restart-def clss-size-def learned-clss-count-def
dest!: *in-vdom-m-fmdropD*)
done
done
done
qed

lemma *deduplicate-binary-clauses-inv-wl-strengthen-def*:
‹*deduplicate-binary-clauses-inv-wl S L* (*abort*, *i*, *a*, *T*) ⟷ *deduplicate-binary-clauses-inv-wl S L* (*abort*, *i*, *a*, *T*) ∧
set-mset (*all-init-atms-st T*) = *set-mset* (*all-init-atms-st S*)›
apply (*rule iffI*)
subgoal
apply (*intro conjI*)
apply (*solves simp*)
unfolding *deduplicate-binary-clauses-inv-wl-def prod.simps*
deduplicate-binary-clauses-inv-def
apply *normalize-goal+*
apply *simp*
subgoal for *xa xb xc*
apply − **unfolding** *mem-Collect-eq prod.simps deduplicate-binary-clauses-inv-def*
using *rtranclp-cdcl-twl-inprocessing-l-all-init-lits-of-l*[*of xa xb*]
rtranclp-cdcl-twl-inprocessing-l-all-learned-lits-of-l[*of xa xb*]
by (*auto simp add*: $\mathcal{L}_{all}$-*all-init-atms all-init-atms-st-alt-def*)
done
subgoal by *simp*
done

lemma *deduplicate-binary-clauses-inv-wl-strengthen-def2*:
‹*deduplicate-binary-clauses-inv-wl S L* = (λ(*abort*, *i*, *a*, *T*). *deduplicate-binary-clauses-inv-wl S L* (*abort*, *i*, *a*, *T*) ∧
set-mset (*all-init-atms-st T*) = *set-mset* (*all-init-atms-st S*) ∧
set-mset ($\mathcal{L}_{all}$ (*all-init-atms-st T*)) = *set-mset* ($\mathcal{L}_{all}$ (*all-init-atms-st S*)))›
apply (*intro ext*, *clarsimp simp only*:)
apply (*subst deduplicate-binary-clauses-inv-wl-strengthen-def*)
apply *auto*
using $\mathcal{L}_{all}$-*cong* **apply** *blast+*
done

definition *mop-watched-by-at-init* :: ‹*'v twl-st-wl* ⇒ *'v literal* ⇒ *nat* ⇒ *'v watcher nres*› **where**
‹*mop-watched-by-at-init* = (λ*S L w*. *do* {
  ASSERT(*L* ∈# *all-init-lits-of-wl S*);
  ASSERT(*w* < *length* (*watched-by S L*));
  RETURN (*watched-by S L* ! *w*)
})›
lemma *mop-watched-by-app-heur-mop-watched-by-at-init-ana*:
  ‹(*uncurry2 mop-watched-by-app-heur*, *uncurry2 mop-watched-by-at-init*) ∈
   twl-st-heur-restart-ana u ×*f nat-lit-lit-rel* ×*f nat-rel* →*f* ⟨*Id*⟩*nres-rel*›
 unfolding *mop-watched-by-app-heur-def mop-watched-by-at-init-def uncurry-def all-lits-def*[*symmetric*]
   all-lits-alt-def[*symmetric*] *twl-st-heur-restart-ana-def twl-st-heur-restart-alt-def2 Let-def*
 by (*intro frefI nres-relI*, *refine-rcg*)
 (*simp-all add*: *map-fun-rel-def* $\mathcal{L}_{all}$*-all-init-atms*(*2*) *watched-by-alt-def*)

lemma *deduplicate-binary-clauses-wl-alt-def*:
‹*deduplicate-binary-clauses-wl L S* = *do* {
   ASSERT (*deduplicate-binary-clauses-pre-wl L S*);
   ASSERT (*L* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st S*));
   let CS = (λ-::*nat literal*. *None*);
   let l = *length* (*watched-by S L*);
   let val = *polarity* (*get-trail-wl S*) *L*;
   (-, -, -, *S*) ← *WHILE*$_T$*deduplicate-binary-clauses-inv-wl S L* (λ(*abort*, *i*, *CS*, *S*). ¬*abort* ∧ *i* < *l* ∧ *get-conflict-wl S* = *None*)
    (λ(*abort*, *i*, *CS*, *S*).
    do {
      (*C*,*L'*, *b*) ← *mop-watched-by-at-init S L i*;
      ASSERT (*L'* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st S*));
      let st = *C* ∈# *dom-m* (*get-clauses-wl S*);
      if ¬*st* ∨ ¬*b then*
        RETURN (*abort*, *i*+*1*, *CS*, *S*)
      else do {
        let - = *polarity* (*get-trail-wl S*) *L'*;
        if defined-lit (*get-trail-wl S*) *L' then do* {
          U ← *simplify-clause-with-unit-st-wl C S*;
          ASSERT (*set-mset* (*all-init-atms-st U*) = *set-mset* (*all-init-atms-st S*));
          ASSERT (*L* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st U*));
          let - = *polarity* (*get-trail-wl U*) *L*;
          RETURN (*defined-lit* (*get-trail-wl U*) *L*, *i*+*1*, *CS*, *U*)
        }
        else do {
          let c = *CS L'*;
          let - = *CS L'*;
          if c ≠ *None then do* {
            let C' = (*if* ¬*snd* (*the c*) ⟶ ¬*irred* (*get-clauses-wl S*) *C then C else fst* (*the c*));
             let CS = (*if* ¬*snd* (*the c*) ⟶ ¬*irred* (*get-clauses-wl S*) *C then CS else CS* (*L'* := *Some* (*C*, *irred* (*get-clauses-wl S*) *C*)));
            ASSERT (*C'* ∈# *dom-m* (*get-clauses-wl S*));
            S ← *clause-remove-duplicate-clause-wl C' S*;
            RETURN (*abort*, *i*+*1*, *CS*, *S*)
          } *else do* {
            let c = *CS* (−*L'*);
            if CS (−*L'*) ≠ *None then do* {
              S ← *binary-clause-subres-wl C L* (−*L'*) *S*;
              RETURN (*True*, *i*+*1*, *CS*, *S*)
            } *else do* {

```
            let CS' = CS (L' := Some (C, irred (get-clauses-wl S) C));
            RETURN (abort, i+1, CS', S)
          }
        }
      }
    }
   })
   (defined-lit (get-trail-wl S) L, 0, CS, S);
  let CS = (λ-::nat literal. None);
  RETURN S
}› (is ‹?A = ?B›)
proof −
  have H: ‹a = b ⟹ (a, b) ∈ Id› ‹x =y ⟹ x ≤ ⇓Id y› for a b x y
    by auto
  have ‹?A ≤ ⇓ Id ?B›
    supply [[goals-limit=1]]
    unfolding Let-def deduplicate-binary-clauses-wl-def bind-to-let-conv mop-watched-by-at-init-def
      nres-monad3
    by (refine-vcg H(1); (rule H refl)?; simp)
  moreover have ‹?B ≤ ⇓Id ?A›
    unfolding Let-def deduplicate-binary-clauses-wl-def bind-to-let-conv mop-watched-by-at-init-def
      nres-monad3
    apply (refine-vcg H(1); (rule H)?)
    subgoal
      unfolding deduplicate-binary-clauses-pre-wl-def deduplicate-binary-clauses-pre-def
      apply normalize-goal+
      by (simp add: 𝓛all-all-init-atms-all-init-lits all-init-lits-of-wl-def all-init-lits-def
        IsaSAT-Setup.get-unit-init-clss-wl-alt-def ac-simps 𝓛all-all-init-atms)
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal for x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i
      apply (subst (asm) deduplicate-binary-clauses-inv-wl-def)
      unfolding  deduplicate-binary-clauses-inv-alt-def case-prod-beta
      apply normalize-goal+
      apply simp
      subgoal for xa xb xc xd
      apply − unfolding mem-Collect-eq prod.simps
      apply normalize-goal+
      using rtranclp-cdcl-twl-inprocessing-l-all-init-lits-of-l[of xa xb]
        rtranclp-cdcl-twl-inprocessing-l-all-learned-lits-of-l[of xa xb]
      by (simp add: 𝓛all-all-init-atms(2) literals-are-𝓛in'-literals-are-𝓛in-iff(4))
      done
    subgoal for x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i
      apply (subst (asm) deduplicate-binary-clauses-inv-wl-def)
      unfolding  deduplicate-binary-clauses-inv-alt-def case-prod-beta
      apply normalize-goal+
      apply simp
      subgoal for xa xb xc xd
      apply − unfolding mem-Collect-eq prod.simps
      apply normalize-goal+
      using rtranclp-cdcl-twl-inprocessing-l-all-init-lits-of-l[of xa xb]
        rtranclp-cdcl-twl-inprocessing-l-all-learned-lits-of-l[of xa xb]
      by (auto simp add: literals-are-𝓛in'-def  blits-in-𝓛in'-def watched-by-alt-def
        𝓛all-all-init-atms dest!: multi-member-split nth-mem)
      done
```

subgoal for *x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i*
apply (*subst* (*asm*) *deduplicate-binary-clauses-inv-wl-def*)
unfolding *deduplicate-binary-clauses-inv-alt-def case-prod-beta*
apply *normalize-goal+*
apply *simp*
subgoal for *xa xb xc xd*
apply − **unfolding** *mem-Collect-eq prod.simps*
apply *normalize-goal+*
using *rtranclp-cdcl-twl-inprocessing-l-all-init-lits-of-l*[*of xa xb*]
rtranclp-cdcl-twl-inprocessing-l-all-learned-lits-of-l[*of xa xb*]
by (*auto simp add*: *literals-are-*$\mathcal{L}_{in}$*'-def blits-in-*$\mathcal{L}_{in}$*'-def watched-by-alt-def*
$\mathcal{L}_{all}$*-all-init-atms dest*!: *multi-member-split nth-mem*)
done
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal
apply (*subst* (*asm*) *deduplicate-binary-clauses-inv-wl-strengthen-def2*)
apply (*clarsimp dest*!:)
apply (*drule* $\mathcal{L}_{all}$*-cong*)
by *presburger*
subgoal by *auto*
subgoal by *auto*
subgoal
unfolding *case-prod-beta deduplicate-binary-clauses-inv-wl-def*
deduplicate-binary-clauses-inv-def deduplicate-binary-clauses-correctly-marked-def
by *normalize-goal+*
(*auto split*: *if-splits*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
ultimately show *?thesis* **unfolding** *Down-id-eq* **by** (*rule antisym*)
qed

lemma *deduplicate-binary-clauses-pre-wl-in-all-atmsD*:
‹*deduplicate-binary-clauses-pre-wl L S* ⟹ *L* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st S*)›
‹*deduplicate-binary-clauses-pre-wl L S* ⟹ *L* ∈# $\mathcal{L}_{all}$ (*all-atms-st S*)›
proof −
assume ‹*deduplicate-binary-clauses-pre-wl L S*›
then show ‹*L* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st S*)›
unfolding *deduplicate-binary-clauses-pre-wl-def deduplicate-binary-clauses-pre-def* **apply** −
apply *normalize-goal+*
by (*simp add*: $\mathcal{L}_{all}$*-all-init-atms-all-init-lits all-init-lits-of-wl-def all-init-lits-def*
IsaSAT-Setup.get-unit-init-clss-wl-alt-def ac-simps $\mathcal{L}_{all}$*-all-init-atms*)
then show ‹*L* ∈# $\mathcal{L}_{all}$ (*all-atms-st S*)›
using $\mathcal{L}_{all}$*-init-all* **by** *blast*
qed

lemma *isa-deduplicate-binary-clauses-mark-duplicated-binary-clauses-as-garbage-wl*:
assumes ‹(*S*, *S'*) ∈ *twl-st-heur-restart-ana''* *r u ns lw*› ‹(*L*,*L'*)∈ *nat-lit-lit-rel*› **and**
‹(*CS*, *Map.empty*) ∈ {((*c*, *m*), *c'*). *c* = *c'* ∧ *m* = (*length* (*get-watched-wl-heur S*))}› (**is** ‹- ∈ *?CS*›)
shows ‹*isa-deduplicate-binary-clauses-wl L CS S* ≤
⇓{((*CS*, *T*), *T'*). (*T*,*T'*) ∈ *twl-st-heur-restart-ana''* *r u ns lw* ∧ (*CS*, *Map.empty*) ∈ {((*c*, *m*), *c'*). *c* = *c'* ∧ *m* = (*length* (*get-watched-wl-heur S*))}}
(*deduplicate-binary-clauses-wl L' S'*)›
proof −
have [*simp*]: ‹*L'* = *L*›
using *assms* **by** *auto*
have [*simp*]: ‹*all-init-atms* (*get-clauses-wl S'*)
(*IsaSAT-Setup.get-unkept-unit-init-clss-wl S'* + *IsaSAT-Setup.get-kept-unit-init-clss-wl S'* + *get-subsumed-init-clauses-wl S'* +
get-init-clauses0-wl S') = *all-init-atms-st S'*›
by (*auto simp*: *all-init-atms-st-def IsaSAT-Setup.get-unit-init-clss-wl-alt-def*)

have [*refine0*]: ‹(*CS*, *Map.empty*) ∈*?CS* ⟹
(*val*, *polarity* (*get-trail-wl S'*) *L'*) ∈ ⟨*bool-rel*⟩*option-rel* ⟹
deduplicate-binary-clauses-inv-wl S' L' (*defined-lit* (*get-trail-wl S'*) *L'*, *0*, *Map.empty*, *S'*) ⟹
((*val* ≠ *UNSET*, *0*, *CS*, *S*), *defined-lit* (*get-trail-wl S'*) *L'*, *0*, *Map.empty*, *S'*) ∈ *bool-rel* ×$_r$ *nat-rel* ×$_r$ *?CS* ×$_r$
({(*a*, *b*). (*a*,*b*)∈ *twl-st-heur-restart-ana''* *r u ns lw* ∧ *learned-clss-count a* ≤ *learned-clss-count S*})›
(**is** ‹- ⟹ - ⟹ -⟹ - ∈ *?loop*›)
for *CS val*
using *assms* **by** (*auto simp*: *polarity-def*)
have [*refine0*]: ‹*isa-simplify-clause-with-unit-st2 C S*
≤ ⇓ {(*a*, *b*). (*a*, *b*) ∈ *twl-st-heur-restart* ∧ *get-avdom a* = *get-avdom S* ∧
get-vdom a = *get-vdom S* ∧
get-ivdom a = *get-ivdom S* ∧
length (*get-clauses-wl-heur a*) = *r* ∧ *learned-clss-count a* ≤ *u* ∧ *learned-clss-count a* ≤ *learned-clss-count S* ∧ *get-vmtf-heur a* = *get-vmtf-heur S* ∧
length (*get-watched-wl-heur a*) = *lw*}
(*simplify-clause-with-unit-st-wl C' T*)›
if ‹(*S*,*T*) ∈ {(*a*, *b*).
(*a*, *b*) ∈ *twl-st-heur-restart* ∧
get-avdom a = *get-avdom S* ∧
get-vdom a = *get-vdom S* ∧ *get-ivdom a* = *get-ivdom S* ∧ *length* (*get-clauses-wl-heur a*) = *r*
∧ *learned-clss-count a* ≤ *u* ∧ *get-vmtf-heur a* = *get-vmtf-heur S* ∧
length (*get-watched-wl-heur a*) = *lw*}›
‹(*C*,*C'*) ∈ *Id*›
for *S T C C'*
apply (*rule isa-simplify-clause-with-unit-st2-simplify-clause-with-unit-st2*[*THEN order-trans*])
apply (*rule that*)
apply (*rule that*)
apply (*rule ref-two-step''*)
defer
apply (*rule simplify-clause-with-unit-st2-simplify-clause-with-unit-st*[*THEN order-trans*, *of* - *C' T T*])
apply *auto*
done
have [*simp*]: ‹(*Sa*, *U*) ∈ *twl-st-heur-restart-ana* (*length* (*get-clauses-wl-heur Sa*)) ⟷ (*Sa*, *U*) ∈ *twl-st-heur-restart*› **for** *Sa U*
by (*auto simp*: *twl-st-heur-restart-ana-def*)
have *KK*: ‹*set-mset* (*$\mathcal{L}_{all}$* (*all-init-atms-st T*)) = *set-mset* (*$\mathcal{L}_{all}$* (*all-init-atms-st S'*)) ⟷

set-mset ((*all-init-atms-st* *T*)) = *set-mset* ((*all-init-atms-st* *S′*))› **for** *S′ T*
apply (*auto simp*: $\mathcal{L}_{all}$-*all-init-atms*(*2*))
apply (*metis* $\mathcal{L}_{all}$-*all-init-atms*(*2*) *atms-of-*$\mathcal{L}_{all}$-$\mathcal{A}_{in}$ *atms-of-cong-set-mset*)
apply (*metis* $\mathcal{L}_{all}$-*all-init-atms*(*2*) *atms-of-*$\mathcal{L}_{all}$-$\mathcal{A}_{in}$ *atms-of-cong-set-mset*)
apply (*metis* $\mathcal{L}_{all}$-*all-init-atms*(*2*) $\mathcal{L}_{all}$-*cong*)+
done

have *get-watched-wl-heur*: ‹*mop-watched-by-app-heur x2e L x1d* ≤ ⇓
{(*a*,*b*). *a* = *b* ∧ *a* = *get-watched-wl-heur x2e* ! *nat-of-lit L* ! *x1d* ∧ *b* = *watched-by x2b L′* ! *x1a* ∧
fst a ∈ *set* (*get-vdom x2e*)} (*mop-watched-by-at-init x2b L′ x1a*)›
(**is** ‹- ≤⇓ *?watched* -›)
if
‹(*S*, *S′*) ∈ *twl-st-heur-restart-ana′′ r u ns lw*› **and**
‹(*L*, *L′*) ∈ *nat-lit-lit-rel*› **and**
‹*deduplicate-binary-clauses-pre-wl L′ S′*› **and**
‹*L′* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st S′*)› **and**
‹(*CS*, *Map.empty*) ∈ {((*c*, *m*), *c′*). *c* = *c′* ∧ *m* = *length* (*get-watched-wl-heur S*)}› **and**
‹*polarity-pol-pre* (*get-trail-wl-heur S*) *L*› **and**
‹*inres* (*RETURN* (*polarity-pol* (*get-trail-wl-heur S*) *L*)) *val*› **and**
‹(*val*, *polarity* (*get-trail-wl S′*) *L′*) ∈ ⟨*bool-rel*⟩*option-rel*› **and**
‹(*x*, *x′*) ∈ *?loop*› **and**
‹*case x of* (*abort*, *i*, *CS*, *Sa*) ⇒ ¬ *abort* ∧ *i* < *l* ∧ *get-conflict-wl-is-None-heur Sa*› **and**
‹*case x′ of* (*abort*, *i*, *CS*, *S*) ⇒ ¬ *abort* ∧ *i* < *length* (*watched-by S′ L′*) ∧ *get-conflict-wl S* = *None*›
and
‹*case x′ of*
(*abort*, *i*, *a*, *T*) ⇒
deduplicate-binary-clauses-inv-wl S′ L′ (*abort*, *i*, *a*, *T*) ∧
set-mset (*all-init-atms-st T*) = *set-mset* (*all-init-atms-st S′*) ∧
set-mset ($\mathcal{L}_{all}$ (*all-init-atms-st T*)) = *set-mset* ($\mathcal{L}_{all}$ (*all-init-atms-st S′*))› **and**
‹*x2a* = (*x1b*, *x2b*)› **and**
‹*x2* = (*x1a*, *x2a*)› **and**
‹*x′* = (*x1*, *x2*)› **and**
‹*x2d* = (*x1e*, *x2e*)› **and**
‹*x2c* = (*x1d*, *x2d*)› **and**
‹*x* = (*x1c*, *x2c*)› **and**
‹(*l*, *length* (*watched-by S′ L′*)) ∈ {(*l*, *l′*). (*l*, *l′*) ∈ *nat-rel* ∧ *l′* = *length* (*watched-by S′ L′*)}›
for *CS val x x′ x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e l*
proof −
show *?thesis*
apply (*rule order-trans*)
apply (*rule mop-watched-by-app-heur-mop-watched-by-at-init-ana*[*of r*, *THEN fref-to-Down-curry2*,
of - - - *x2b L′ x1a*])
subgoal by *fast*
subgoal using *that* **by** *auto*
unfolding *Down-id-eq mop-watched-by-at-init-def*
apply (*refine-rcg*)
using *that twl-st-heur-restart-ana-watchlist-in-vdom*[**where** *L*=*L* **and** *x2e*=*x2e* **and** *x2f*=*x2b* **and**
x1d = *x1d*
and *a*=‹*fst* (*get-watched-wl-heur x2e* ! *nat-of-lit L* ! *x1d*)› **and** *b*=‹*snd*(*get-watched-wl-heur x2e* !
nat-of-lit L ! *x1d*)›
and *r*=*r*]
by (*auto simp*: *twl-st-heur-restart-ana-state-simp watched-by-alt-def*
deduplicate-binary-clauses-inv-wl-def mop-watched-by-at-init-def)
qed
have *watched-in-vdom*:

‹x1h ∈ set (get-vdom x2e)› ‹(x1h, x1f) ∈ nat-rel›
if ‹(xa, x'a) ∈ ?watched x1d x1a x2b x2e›
‹x'a = (x1f, x2f)›
‹x2f = (x3f, x3f')›
‹xa = (x1h, x2h)›
‹x2h = (x3h, x3h')›
for x2e xa x'a x1h x2h x1f x2f x1d x1a x2b x3f x3f' x3h x3h'
using *that*
by *auto*
have *irred-status*: ‹¬ (x1f ∉# dom-m (get-clauses-wl x2b) ∨ ¬ x2g) ⟹
(xb, x1f ∈# dom-m (get-clauses-wl x2b))
∈ {(a, b). (a ≠ DELETED) = b ∧
(a = IRRED) = (irred (get-clauses-wl x2b) x1f ∧ b) ∧ (a = LEARNED) = (¬ irred (get-clauses-wl x2b) x1f ∧ b)} ⟹
(xb, irred (get-clauses-wl x2b) x1f) ∈ {(a,b). a ≠ DELETED ∧ ((a=IRRED) ⟷ b) ∧ ((a=LEARNED) ⟷ ¬b)}›
for xb x2b x1f x2g
by (*cases* xb) *auto*
have *twl-st-heur-restart-ana-stateD*: ‹valid-arena (get-clauses-wl-heur x2e) (get-clauses-wl x2b) (set (get-vdom x2e))›
if ‹(x2e, x2b) ∈ twl-st-heur-restart-ana r›
for x2e x2b
using *that* **unfolding** *twl-st-heur-restart-ana-def twl-st-heur-restart-def*
by *simp*
have *is-markedI*: ‹(x1e, x1e') ∈ ?CS ⟹ (x1i, x1i') ∈ nat-lit-lit-rel ⟹ x1i' ∈# $\mathcal{L}_{all}$ (all-init-atms-st S') ⟹
is-marked x1e x1i ≤ SPEC (λc. (c, x1e' x1i') ∈ {(a,b). a ⟷ b≠None})›
‹(x1e, x1e') ∈ ?CS ⟹ (x1i, x1i') ∈ nat-lit-lit-rel ⟹ (m, x1e' x1i') ∈ {(a,b). a ⟷ b≠None} ⟹x1i' ∈# $\mathcal{L}_{all}$ (all-init-atms-st S') ⟹
(if m then get-marked x1e x1i else RETURN (1, True))
≤ SPEC
(λc. (c, x1e' x1i') ∈ {(a,b). b ≠ None ⟶ a = the b})›
‹(x1e, x1e') ∈ ?CS ⟹ (x1i, x1i') ∈ nat-lit-lit-rel ⟹ x1i' ∈# $\mathcal{L}_{all}$ (all-init-atms-st S') ⟹ (x, x') ∈ Id ⟹
(m, x1e' x1i') ∈ {(a,b). a ⟷ b≠None} ⟹ ¬m ⟹
set-marked x1e x1i x ≤ SPEC (λc. (c, x1e'(x1i' ↦ x')) ∈ ?CS)›
for x1e x1e' x1i x1i' m x x'
using *assms*(1)
unfolding *is-marked-def get-marked-def set-marked-def*
by (*auto intro*!: *ASSERT-leI simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def map-fun-rel-def*)
have *length-watchlist*:
‹(S, S') ∈ twl-st-heur-restart-ana'' r u ns lw ⟹
(L, L') ∈ nat-lit-lit-rel ⟹
L' ∈# $\mathcal{L}_{all}$ (all-init-atms-st S') ⟹
mop-length-watched-by-int S L ≤ SPEC (λc. (c, length (watched-by S' L')) ∈ {(l,l'). (l,l') ∈ nat-rel ∧ l' = length (watched-by S' L')})›
by (*auto simp*: *mop-length-watched-by-int-def twl-st-heur-restart-ana-def twl-st-heur-restart-def map-fun-rel-def watched-by-alt-def intro*!: *ASSERT-leI*)
have [*refine0*]: ‹(CS, a) ∈ ?CS ⟹ empty CS ≤ SPEC (λu. (u, Map.empty) ∈ ?CS)› **for** a CS
by (*auto simp*: *empty-def*)

have *update-marked*: ‹(if ¬ snd n ⟶ st = LEARNED then RETURN x1e else update-marked x1e x1i (x1h, st = IRRED))
≤ SPEC
(λc. (c, if ¬ snd (the (x1b x1g)) ⟶ ¬ irred (get-clauses-wl x2b) x1f then x1b

else x1b(*x1g* ↦ (*x1f*, *irred* (*get-clauses-wl x2b*) *x1f*)))
∈ *?CS*)›
if
loop: ‹(*x*, *x′*) ∈ *?loop*› **and**
‹*case x′ of*
(*abort*, *i*, *a*, *T*) ⇒
deduplicate-binary-clauses-inv-wl S′ L′ (*abort*, *i*, *a*, *T*) ∧
set-mset (*all-init-atms-st T*) = *set-mset* (*all-init-atms-st S′*) ∧
set-mset (*L*$_{all}$ (*all-init-atms-st T*)) = *set-mset* (*L*$_{all}$ (*all-init-atms-st S′*))› **and**
st: ‹*x2a* = (*x1b*, *x2b*)›
‹*x2* = (*x1a*, *x2a*)›
‹*x′* = (*x1*, *x2*)›
‹*x2d* = (*x1e*, *x2e*)›
‹*x2c* = (*x1d*, *x2d*)›
‹*x* = (*x1c*, *x2c*)›
‹*x2f* = (*x1g*, *x2g*)›
‹*x′a* = (*x1f*, *x2f*)›
‹*x2h* = (*x1i*, *x2i*)›
‹*xa* = (*x1h*, *x2h*)› **and**
‹*x1d* < *l*› **and**
‹(*xa*, *x′a*)
∈ {(*a*, *b*).
a = *b* ∧
a = *get-watched-wl-heur x2e* ! *nat-of-lit L* ! *x1d* ∧
b = *watched-by x2b L′* ! *x1a* ∧ *fst a* ∈ *set* (*get-vdom x2e*)}› **and**
‹*x1g* ∈# *L*$_{all}$ (*all-init-atms-st x2b*)› **and**
‹*0* < *x1h* ∧ *x1h* < *length* (*get-clauses-wl-heur x2e*)› **and**
‹(*st*, *x1f* ∈# *dom-m* (*get-clauses-wl x2b*))
∈ {(*a*, *b*).
(*a* ≠ *DELETED*) = *b* ∧
(*a* = *IRRED*) = (*irred* (*get-clauses-wl x2b*) *x1f* ∧ *b*) ∧
(*a* = *LEARNED*) = (¬ *irred* (*get-clauses-wl x2b*) *x1f* ∧ *b*)}› **and**
‹¬ (*st* = *DELETED* ∨ ¬ *x2i*)› **and**
‹¬ (*x1f* ∉# *dom-m* (*get-clauses-wl x2b*) ∨ ¬ *x2g*)› **and**
m: ‹(*m*, *x1b x1g*) ∈ {*a*. *case a of* (*a*, *b*) ⇒ *a* = (*b* ≠ *None*)}› *m* **and**
‹(*n*, *x1b x1g*) ∈ {*a*. *case a of* (*a*, *b*) ⇒ *b* ≠ *None* ⟶ *a* = *the b*}› **and**
‹*x1b x1g* ≠ *None*›
for *l val x x′ x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e xa x′a x1f x2f x1g x2g x1h x2h x1i x2i st*
vala m n
proof −
have ‹(*get-watched-wl-heur S*, *get-watched-wl S′*) ∈ ⟨*Id*⟩*map-fun-rel* (*D*$_0$ (*all-init-atms-st S′*))›
using *assms*
by (*auto intro*!: *ASSERT-leI simp*: *st twl-st-heur-restart-def twl-st-heur-restart-ana-def*)
then show *?thesis*
using *that*
unfolding *update-marked-def*
by (*auto intro*!: *ASSERT-leI simp*: *st map-fun-rel-def*)
qed
show *?thesis*
supply [[*goals-limit=1*]]
using *assms*
unfolding *isa-deduplicate-binary-clauses-wl-def deduplicate-binary-clauses-wl-alt-def mop-polarity-pol-def nres-monad3* **apply** −
apply (*subst deduplicate-binary-clauses-wl-alt-def*)
apply (*subst deduplicate-binary-clauses-inv-wl-strengthen-def2*)

apply (*refine-rcg polarity-pol-polarity*[*of* ‹*all-init-atms-st S'*›, *THEN fref-to-Down-unRET-uncurry*]
mop-arena-status-vdom isa-clause-remove-duplicate-clause-wl-clause-remove-duplicate-clause-wl[*of r* ‹*learned-clss-count S*› *ns lw* **for** *S*,
THEN fref-to-Down-curry, *of - - - S S* **for** *S*]
isa-binary-clause-subres-isa-binary-clause-subres-wl[*of r* ‹*learned-clss-count S*› *ns lw* **for** *S*, *THEN fref-to-Down-curry3*, *of - - - S - - - - S* **for** *S*]
length-watchlist)
subgoal
using *length-watched-le-ana*[*of S' S* ‹*length* (*get-clauses-wl-heur S*)› *L*]
by (*auto simp add*: *deduplicate-binary-clauses-pre-wl-def watched-by-alt-def*
deduplicate-binary-clauses-pre-wl-in-all-atmsD $\mathcal{L}_{all}$*-all-init-atms*(*2*)
twl-st-heur-restart-ana-state-simp twl-st-heur-restart-ana-def)
subgoal
by (*rule polarity-pol-pre*)
(*use assms* **in** ‹*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*›)[*2*]
subgoal
by *auto*
subgoal
by (*use assms* **in** ‹*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*›)
subgoal by *auto*
subgoal for *CS val*
by (*auto simp*: *watched-by-alt-def deduplicate-binary-clauses-pre-wl-in-all-atmsD get-conflict-wl-is-None-def*
twl-st-heur-restart-ana-state-simp get-conflict-wl-is-None-heur-get-conflict-wl-is-None-ana[*THEN fref-to-Down-unRET-Id*])
subgoal by *auto*
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
subgoal using *assms* **by** (*auto simp*: *twl-st-heur-restart-ana-def*)
apply (*rule get-watched-wl-heur*; *assumption*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*)
(*metis* (*no-types*, *lifting*) *arena-dom-status-iff*(*3*) *bot-nat-0.extremum gr0I le-antisym numeral-le-iff semiring-norm*(*69*))+

subgoal by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*
intro!: *valid-arena-in-vdom-le-arena*)
apply (*solves auto*)
subgoal for *CS val x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e*
by *auto*
subgoal
by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*)
subgoal by *auto*
subgoal
by (*simp add*: *deduplicate-binary-clauses-pre-wl-in-all-atmsD* $\mathcal{L}_{all}$*-all-init-atms*(*2*))
subgoal
apply (*rule polarity-pol-pre*)
apply (*use assms* **in** ‹*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-alt-def2 Let-def*›)[]
apply (*clarsimp simp add*: *watched-by-alt-def twl-st-heur-restart-ana-state-simp*)
done
subgoal by *auto*
subgoal
unfolding *prod-rel-iff*
apply (*intro conjI impI*)
subgoal
unfolding *twl-st-heur-restart-alt-def2 twl-st-heur-restart-ana-def Let-def KK prod.simps*
apply (*simp only*: *in-pair-collect-simp prod-rel-iff prod.simps*)
apply *normalize-goal+*
apply (*rule trail-pol-cong*, *assumption*, *assumption*)

done
subgoal
by (*clarsimp simp*: *watched-by-alt-def twl-st-heur-restart-ana-state-simp dest*: *trail-pol-cong*)
done
subgoal
by (*auto simp*: *polarity-def*)
subgoal
by (*auto simp*: *twl-st-heur-restart-ana-def*)
subgoal by (*clarsimp simp add*: *watched-by-alt-def twl-st-heur-restart-ana-state-simp*)
subgoal
apply (*rule polarity-pol-pre*)
apply (*use assms* **in** ‹*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-alt-def2 Let-def*›)[]
apply (*clarsimp simp add*: *watched-by-alt-def twl-st-heur-restart-ana-state-simp*)
done
subgoal by (*clarsimp simp add*: *twl-st-heur-restart-ana-def*)
subgoal
unfolding *twl-st-heur-restart-alt-def2 twl-st-heur-restart-ana-def Let-def KK prod.simps*
apply (*simp only*: *in-pair-collect-simp prod-rel-iff prod.simps*)
apply *normalize-goal+*
by (*metis* (*no-types*, *lifting*) *trail-pol-cong*)
subgoal
by (*auto simp*: *twl-st-heur-restart-ana-state-simp polarity-def*)
apply (*rule is-markedI*)
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
apply (*rule is-markedI*)
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *auto*
apply (*rule update-marked*; *assumption*)
subgoal by (*auto split*: *if-splits*)
subgoal by *simp*
subgoal by *simp*
apply (*rule is-markedI*)
subgoal by *simp*
subgoal by *simp*
subgoal by (*simp add*: *uminus-$\mathcal{A}_{in}$-iff*)
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
subgoal by *simp*
apply (*rule is-markedI*)
subgoal by *simp*
subgoal by *simp*
subgoal by (*simp add*: *uminus-$\mathcal{A}_{in}$-iff*)
subgoal by *simp*
apply *assumption*
subgoal by *auto*
apply (*solves auto*)
apply (*solves auto*)
subgoal by *auto*
done
qed

lemma *lambda-split-second*: ‹(λ(*a*, *x*). *f a x*) = (λ(*a*,*b*,*c*:: *isasat*). *f a* (*b*,*c*))›
by (*auto intro*!: *ext*)

lemma *isa-mark-duplicated-binary-clauses-as-garbage-wl-alt-def*:
‹*isa-mark-duplicated-binary-clauses-as-garbage-wl S*$_0$ = *do* {
ASSERT (*mark-duplicated-binary-clauses-as-garbage-pre-wl-heur S*$_0$);
let skip = *should-eliminate-pure-st S*$_0$;
CS ← *create* (*length* (*get-watched-wl-heur S*$_0$));
(*CS*, *S*) ← *iterate-over-VMTFC*
(λ*A* (*CS*, *S*). *do* {*ASSERT* (*get-vmtf-heur-array S*$_0$ = (*get-vmtf-heur-array S*));
skip-lit ← *mop-is-marked-added-heur-st S A*;
if ¬*skip then RETURN* (*CS*, *S*)
else do {
ASSERT (*length* (*get-clauses-wl-heur S*) ≤ *length* (*get-clauses-wl-heur S*$_0$) ∧ *learned-clss-count S* ≤ *learned-clss-count S*$_0$);
(*CS*, *S*) ← *isa-deduplicate-binary-clauses-wl* (*Pos A*) *CS S*;
ASSERT (*length* (*get-clauses-wl-heur S*) ≤ *length* (*get-clauses-wl-heur S*$_0$) ∧ *learned-clss-count S* ≤ *learned-clss-count S*$_0$);
(*CS*, *S*) ← *isa-deduplicate-binary-clauses-wl* (*Neg A*) *CS S*;
ASSERT (*get-vmtf-heur-array S*$_0$ = (*get-vmtf-heur-array S*));
RETURN (*CS*, *S*)
}})
(λ(*CS*, *S*). *get-vmtf-heur-array S*$_0$ = (*get-vmtf-heur-array S*))
(λ(*CS*, *S*). *get-conflict-wl-is-None-heur S*)
(*get-vmtf-heur-array S*$_0$, *Some* (*get-vmtf-heur-fst S*$_0$)) (*CS*, *S*$_0$);
RETURN S
}›
unfolding *iterate-over-VMTFC-def prod.simps nres-monad3 Let-def*
apply (*rewrite at* ‹*WHILE*$_T$⁻ - ⌑› *lambda-split-second*)
unfolding *isa-mark-duplicated-binary-clauses-as-garbage-wl-def*
apply (*rewrite at* ‹*WHILE*$_T$⁻ - ⌑ -› *lambda-split-second*)
apply (*auto intro*!: *bind-cong simp*: *Let-def*)
done

definition *mark-duplicated-binary-clauses-as-garbage-wl2* :: ‹- ⇒ *'v twl-st-wl nres*› **where**
‹*mark-duplicated-binary-clauses-as-garbage-wl2 S* = *do* {
ASSERT (*mark-duplicated-binary-clauses-as-garbage-pre-wl S*);
Ls ← *SPEC* (λ*Ls*:: *'v multiset*. *set-mset Ls* = *set-mset* (*atm-of* ‘# *all-init-lits-of-wl S*) ∧ *distinct-mset Ls*);
(-, *S*) ← *WHILE*$_T$λ(*L*, *T*). *mark-duplicated-binary-clauses-as-garbage-wl-inv Ls S* (*T*, *L*)(λ(*Ls*, *S*). *Ls* ≠ {#} ∧ *get-conflict-wl S* = *None*)
(λ(*Ls*, *S*). *do* {
ASSERT (*Ls* ≠ {#});
L ← *SPEC* (λ*L*. *L* ∈# *Ls*);
ASSERT (*L* ∈# *atm-of* ‘# *all-init-lits-of-wl S*);
skip ← *RES* (*UNIV* :: *bool set*);
if skip then RETURN (*remove1-mset L Ls*, *S*)
else do {
S ← *deduplicate-binary-clauses-wl* (*Pos L*) *S*;
S ← *deduplicate-binary-clauses-wl* (*Neg L*) *S*;
RETURN (*remove1-mset L Ls*, *S*)
}
})
(*Ls*, *S*);

```
    RETURN S
  }›

lemma mark-duplicated-binary-clauses-as-garbage-wl2-alt-def:
  ‹mark-duplicated-binary-clauses-as-garbage-wl2 S = do {
     ASSERT (mark-duplicated-binary-clauses-as-garbage-pre-wl S);
    Ls ← SPEC (λLs:: 'v multiset. set-mset Ls = set-mset (atm-of ‘# all-init-lits-of-wl S) ∧ distinct-mset
Ls);
    (-, S) ← WHILE_T^λ(L, T). mark-duplicated-binary-clauses-as-garbage-wl-inv Ls S (T, L)(λ(Ls, S). Ls ≠
{#} ∧ get-conflict-wl S = None)
     (λ(Ls, S). do {
       ASSERT (Ls ≠ {#});
       L ← SPEC (λL. L ∈# Ls);
       S ← do {
         ASSERT (L ∈# atm-of ‘# all-init-lits-of-wl S);
         skip ← RES (UNIV :: bool set);
         if skip then RETURN (S)
         else do {
           S ← deduplicate-binary-clauses-wl (Pos L) S;
           S ← deduplicate-binary-clauses-wl (Neg L) S;
           RETURN (S)
         }
       };
       RETURN (remove1-mset L Ls, S)
     })
     (Ls, S);
    RETURN S
  }›
  unfolding nres-monad-laws mark-duplicated-binary-clauses-as-garbage-wl2-def bind-to-let-conv Let-def
  apply (auto intro!: bind-cong[OF refl] simp: bind-to-let-conv)
  apply (subst bind-to-let-conv Let-def)+
  apply (auto simp: Let-def nres-monad-laws intro!: bind-cong)
  apply (subst nres-monad-laws)+
  apply auto
  done

lemma mark-duplicated-binary-clauses-as-garbage-wl2-ge-\mathcal{L}_{all}:
  ‹⇓ Id (mark-duplicated-binary-clauses-as-garbage-wl2 S) ≥ do {
     ASSERT (mark-duplicated-binary-clauses-as-garbage-pre-wl S);
   iterate-over-\mathcal{L}_{all} C
  (λL S. do {
         ASSERT (L ∈# atm-of ‘# all-init-lits-of-wl S);
         skip ← RES (UNIV :: bool set);
         if skip then RETURN (S)
         else do {
           S ← deduplicate-binary-clauses-wl (Pos L) S;
           S ← deduplicate-binary-clauses-wl (Neg L) S;
           RETURN (S)
         }
       })
       (atm-of ‘# all-init-lits-of-wl S)
       (λA T. mark-duplicated-binary-clauses-as-garbage-wl-inv (all-init-atms-st S) S (T, A))
       (λS. get-conflict-wl S = None) S}›
proof −
  have H: ‹a=b ⟹ (a,b) ∈ Id› for a b
    by auto
```

```
  have H': ‹a=b ⟹ a ≤⇓Id b› for a b
    by auto
  have HH: ‹mark-duplicated-binary-clauses-as-garbage-wl-inv Ls S (x2, x1) ⟹
    set-mset Ls = set-mset (all-init-atms-st S) ⟹
   distinct-mset Ls ⟹ mark-duplicated-binary-clauses-as-garbage-wl-inv (all-init-atms-st S) S (x2, x1)›
    for Ls x2 x1
    unfolding mark-duplicated-binary-clauses-as-garbage-wl-inv-def
      mark-duplicated-binary-clauses-as-garbage-inv-def prod.simps
    apply normalize-goal+
    apply (rule-tac x=x in exI, rule-tac x=xa in exI)
    apply simp
    by (metis Duplicate-Free-Multiset.distinct-mset-mono distinct-subseteq-iff)

  show ?thesis
    unfolding iterate-over-\mathcal{L}_{all}C-def mark-duplicated-binary-clauses-as-garbage-wl2-alt-def
    apply refine-vcg
    apply (rule H)
    subgoal by auto
    subgoal by (auto simp flip: all-init-atms-st-alt-def intro: HH)
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    apply (rule H)
    subgoal by auto
    apply (rule H')
    subgoal by auto
    apply (rule H')
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    done
qed

lemma mark-duplicated-binary-clauses-as-garbage-wl2-mark-duplicated-binary-clauses-as-garbage-wl:
  ‹mark-duplicated-binary-clauses-as-garbage-wl2 S ≤ ⇓Id (mark-duplicated-binary-clauses-as-garbage-wl
S)›
proof −
  have H: ‹fst a = snd b ∧ snd a = fst b ⟹ (a,b) ∈ {((s,t), (u, v)). (s=v) ∧ (t=u)}› for a b
    by (cases a; cases b) simp
  have H': ‹a = b ⟹ a ≤ ⇓Id b› for a b
    by auto
  show ?thesis
    unfolding mark-duplicated-binary-clauses-as-garbage-wl2-def
      mark-duplicated-binary-clauses-as-garbage-wl-def
    apply (refine-vcg)
    subgoal by auto
    apply (rule H)
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
```

subgoal by *auto*
subgoal by *auto*
apply (*rule H'*)
subgoal by *auto*
apply (*rule H'*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

lemma *isa-mark-duplicated-binary-clauses-as-garbage-wl-mark-duplicated-binary-clauses-as-garbage-wl*:
assumes ‹(*S*, *S'*) ∈ *twl-st-heur-restart-ana' r u*›
shows ‹*isa-mark-duplicated-binary-clauses-as-garbage-wl S* ≤
⇓(*twl-st-heur-restart-ana' r u*) (*mark-duplicated-binary-clauses-as-garbage-wl S'*)›
proof −
have *1*: ‹*get-vmtf-heur S* ∈ *bump-heur* (*atm-of* '# *all-init-lits-of-wl S'*) (*get-trail-wl S'*)› **and**
2: ‹*isasat-input-nempty* (*all-init-atms-st S'*)› **and**
3: ‹*isasat-input-bounded* (*all-init-atms-st S'*)›
using *assms* **unfolding** *twl-st-heur-restart-ana-def twl-st-heur-restart-alt-def2 Let-def*
by (*simp-all add*: *all-init-atms-st-alt-def*)
have [*refine0*]: ‹*RETURN False* ≤ ⇓ {(*a*,*b*). *a* = *b* ∧ ¬*b*} (*RES UNIV*)›
by (*auto intro*!: *RETURN-RES-refine*)
have *create*: ‹*create* (*length* (*get-watched-wl-heur S*)) ≤ *SPEC* (λ*c*. (*c*, *Map.empty*) ∈ {((*c* :: *nat literal* ⇒ (*nat* × *bool*) *option*, *m*::*nat*), *c'*). *c* = *c'* ∧ *m* = (*length* (*get-watched-wl-heur S*))})› (**is** ‹- -≤ *SPEC*(λ-. - ∈ *?CS*)›)
by (*auto simp*: *create-def*)
have *init*: ‹(*x2a*, *x2*) ∈ ⟨*nat-rel*⟩*option-rel* ⟹ (*CS*, *Map.empty*) ∈ *?CS* ⟹
((*x2a*, *CS*, *S*), *x2*, *S'*) ∈ {((*a*,*CS*,*T*), (*b*,*T'*)). ((*a*,*T*), *b*,*T'*) ∈ ⟨*nat-rel*⟩*option-rel* ×*r* {(*a*,*b*). (*a*,*b*)∈ *twl-st-heur-restart-ana''* (*length* (*get-clauses-wl-heur S*))
(*learned-clss-count S*) (*get-vmtf-heur S*) (*length* (*get-watched-wl-heur S*))} ∧
(*CS*, *Map.empty*) ∈ {((*c* :: *nat literal* ⇒ (*nat* × *bool*) *option*, *m*::*nat*), *c'*). *c* = *c'* ∧ *m* = (*length* (*get-watched-wl-heur T*))}}›
(**is** ‹- ⟹ - ⟹ - ∈ *?loop*›)
for *x2a x2 CS*
using *assms*
by (*auto simp*: *get-vmtf-heur-array-def twl-st-heur-restart-ana-def*)
have *rel*: ‹(*xa*, *Sa*)
∈ {((*CS*, *T*), *T'*).
(*T*, *T'*)
∈ *twl-st-heur-restart-ana''* (*length* (*get-clauses-wl-heur S*)) (*learned-clss-count S*) (*get-vmtf-heur S*)
(*length* (*get-watched-wl-heur S*)) ∧
(*CS*, *Map.empty*) ∈ {((*c*, *m*), *c'*). *c* = *c'* ∧ *m* = *length* (*get-watched-wl-heur x2d*)}} ⟹
xa = (*x1e*, *x2e*) ⟹
length (*get-clauses-wl-heur x2e*) ≤ *length* (*get-clauses-wl-heur S*) ∧
learned-clss-count x2e ≤ *learned-clss-count S* ⟹
(*xb*, *x'a*)
∈ {((*CS*, *T*), *T'*).
(*T*, *T'*)
∈ *twl-st-heur-restart-ana''* (*length* (*get-clauses-wl-heur S*)) (*learned-clss-count S*) (*get-vmtf-heur S*)
(*length* (*get-watched-wl-heur S*)) ∧
(*CS*, *Map.empty*) ∈ {((*c*, *m*), *c'*::*nat literal* ⇒ (*nat* × *bool*) *option*). *c* = *c'* ∧ *m* = *length* (*get-watched-wl-heur x2e*)}} ⟹
xb = (*a*, *b*) ⟹
get-vmtf-heur-array S = *get-vmtf-heur-array b* ⟹
((*a*, *b*), *x'a*) ∈ {((*CS*, *T*), *T'*). (*T*,*T'*)∈*twl-st-heur-restart-ana''* (*length* (*get-clauses-wl-heur S*))

(*learned-clss-count S*) (*get-vmtf-heur S*)
(*length* (*get-watched-wl-heur S*)) ∧
(*CS*, *Map.empty*) ∈ {((*c*, *m*), *c'*::*nat literal* ⇒ (*nat* × *bool*) *option*). *c* = *c'* ∧ *m* = *length* (*get-watched-wl-heur S*)}› **for** *A Sa x1d x2d skip xa x1e x2e xb x'a a b*
by *auto*
have *rel2*:
‹(*x*, *x'*)
∈ {((*a*, *CS*, *T*), *b*, *T'*).
(*a*, *T*), *b*, *T'*)
∈ ⟨*nat-rel*⟩*option-rel* ×*f*
{(*a*, *b*).
(*a*, *b*)
∈ *twl-st-heur-restart-ana''* (*length* (*get-clauses-wl-heur S*)) (*learned-clss-count S*)
(*get-vmtf-heur S*) (*length* (*get-watched-wl-heur S*))} ∧
(*CS*, *Map.empty*) ∈ {((*c*, *m*), *c'*). *c* = *c'* ∧ *m* = *length* (*get-watched-wl-heur T*)}} ⟹
x' = (*x1b*, *x2b*) ⟹ *x* = (*x1c*, *x2c*) ⟹ (*x2c*, *x2b*) ∈ {((*CS*, *T*), *T'*).
(*T*), *T'*)
∈
{(*a*, *b*).
(*a*, *b*)
∈ *twl-st-heur-restart-ana''* (*length* (*get-clauses-wl-heur S*)) (*learned-clss-count S*)
(*get-vmtf-heur S*) (*length* (*get-watched-wl-heur S*))} ∧
(*CS*, *Map.empty*) ∈ {((*c*, *m*), *c'*). *c* = *c'* ∧ *m* = *length* (*get-watched-wl-heur T*)}}› **for** *x' x1b x2b x1c x2c x*
by *auto*
have *rel0*: ‹((*x1d*, *x2d*), *x2b*) ∈ {((*CS*, *T*), *T'*). (*T*,*T'*)∈*twl-st-heur-restart-ana''* (*length* (*get-clauses-wl-heur S*)) (*learned-clss-count S*) (*get-vmtf-heur S*)
(*length* (*get-watched-wl-heur S*)) ∧
(*CS*, *Map.empty*) ∈ {((*c*, *m*), *c'*::*nat literal* ⇒ (*nat* × *bool*) *option*). *c* = *c'* ∧ *m* = *length* (*get-watched-wl-heur S*)}›
if
‹*mark-duplicated-binary-clauses-as-garbage-pre-wl S'*› **and**
‹*mark-duplicated-binary-clauses-as-garbage-pre-wl-heur S*› **and**
‹*inres* (*create* (*length* (*get-watched-wl-heur S*))) *CS*› **and**
‹(*CS*, *Map.empty*) ∈ {((*c*, *m*), *c'*). *c* = *c'* ∧ *m* = *length* (*get-watched-wl-heur S*)}› **and**
‹(*get-vmtf-heur-array S*, *Some* (*get-vmtf-heur-fst S*)) = (*x1a*, *x2a*)› **and**
‹(*x*, *x'*)
∈ {((*a*, *CS*, *T*), *b*, *T'*).
(*a*, *T*), *b*, *T'*)
∈ ⟨*nat-rel*⟩*option-rel* ×*f*
{(*a*, *b*).
(*a*, *b*)
∈ *twl-st-heur-restart-ana''* (*length* (*get-clauses-wl-heur S*)) (*learned-clss-count S*)
(*get-vmtf-heur S*) (*length* (*get-watched-wl-heur S*))} ∧
(*CS*, *Map.empty*) ∈ {((*c*, *m*), *c'*). *c* = *c'* ∧ *m* = *length* (*get-watched-wl-heur T*)}}› **and**
‹*case x of* (*n*, *x*) ⇒ ∀ *x1 x2*. *x* = (*x1*, *x2*) ⟶ *n* ≠ *None* ∧ *get-conflict-wl-is-None-heur x2*› **and**
‹*case x' of* (*n*, *x*) ⇒ *n* ≠ *None* ∧ *get-conflict-wl x* = *None*› **and**
‹*case x of* (*n*, *CS*, *Sa*) ⇒ *get-vmtf-heur-array S* = *get-vmtf-heur-array Sa*› **and**
‹*case x' of*
(*n*, *x*) ⇒
∃*B'*. *mark-duplicated-binary-clauses-as-garbage-wl-inv* (*all-init-atms-st S'*) *S'*
(*x*, *B'*)› **and**
‹*x'* = (*x1b*, *x2b*)› **and**
‹*x* = (*x1c*, *x2c*)› **and**
‹*x1b* ≠ *None*› **and**
‹*x1c* ≠ *None*› **and**

‹the x1b < length x1› **and**
‹the x1b ≤ unat32-max div 2› **and**
‹the x1c < length x1a› **and**
‹the x1c ≤ unat32-max div 2› **and**
‹x2c = (x1d, x2d)› **and**
‹get-vmtf-heur-array S = get-vmtf-heur-array x2d› **and**
‹the x1b ∈# atm-of ‘# all-init-lits-of-wl x2b›
for *skip CS x1 x2 x1a x2a x x′ x1b x2b x1c x2c x1d x2d skipa*
using *that* **by** *auto*
have *binary-deduplicate-required*: ‹(*should-eliminate-pure-st S*, *True*) ∈ *UNIV*›
for *S skip v*
by (*auto simp*: *should-eliminate-pure-st-def*)
have *GC-required-skip*: ‹*mop-is-marked-added-heur-st x2d* (*the x1c*)
≤ ⇓ {(*a*,*b*). (¬*skip*,*b*)∈*bool-rel*}
(*RES UNIV*)›
if
‹*mark-duplicated-binary-clauses-as-garbage-pre-wl S′*› **and**
‹*mark-duplicated-binary-clauses-as-garbage-pre-wl-heur S*› **and**
‹*inres* (*create* (*length* (*get-watched-wl-heur S*))) *CS*› **and**
‹(*CS*, *Map.empty*)
∈ {((*c*, *m*), *c′*). *c* = *c′* ∧ *m* = *length* (*get-watched-wl-heur S*)}› **and**
‹(*get-vmtf-heur-array S*, *Some* (*get-vmtf-heur-fst S*)) =
(*x1a*, *x2a*)› **and**
‹(*x*, *x′*)
∈ {((*a*, *CS*, *T*), *b*, *T′*).
((*a*, *T*), *b*, *T′*)
∈ ⟨*nat-rel*⟩*option-rel* ×*f*
{(*a*, *b*).
(*a*, *b*)
∈ *twl-st-heur-restart-ana′′* (*length* (*get-clauses-wl-heur S*))
(*learned-clss-count S*) (*get-vmtf-heur S*)
(*length* (*get-watched-wl-heur S*))} ∧
(*CS*, *Map.empty*)
∈ {((*c*, *m*), *c′*).
c = *c′* ∧ *m* = *length* (*get-watched-wl-heur T*)}}› **and**
‹*case x of*
(*n*, *x*) ⇒
∀ *x1 x2*.
x = (*x1*, *x2*) ⟶ *n* ≠ *None* ∧ *get-conflict-wl-is-None-heur x2*› **and**
‹*case x′ of* (*n*, *x*) ⇒ *n* ≠ *None* ∧ *get-conflict-wl x* = *None*› **and**
‹*case x of*
(*n*, *CS*, *Sa*) ⇒ *get-vmtf-heur-array S* = *get-vmtf-heur-array Sa*› **and**
‹*case x′ of*
(*n*, *x*) ⇒
∃ℬ′. *mark-duplicated-binary-clauses-as-garbage-wl-inv*
(*all-init-atms-st S′*) *S′* (*x*, ℬ′)› **and**
‹*x′* = (*x1b*, *x2b*)› **and**
‹*x* = (*x1c*, *x2c*)› **and**
‹*x1b* ≠ *None*› **and**
‹*x1c* ≠ *None*› **and**
‹*the x1b < length x1*› **and**
‹*the x1b ≤ unat32-max div 2*› **and**
‹*the x1c < length x1a*› **and**
‹*the x1c ≤ unat32-max div 2*› **and**
‹*x2c* = (*x1d*, *x2d*)› **and**
‹*get-vmtf-heur-array S* = *get-vmtf-heur-array x2d*› **and**

```
    ‹the x1b ∈# atm-of ‘# all-init-lits-of-wl x2b›
   for skip CS x1 x2 x1a x2a x x′ x1b x2b x1c x2c x1d x2d
  proof −
   term ?thesis
   have heur: ‹heuristic-rel (all-init-atms-st x2b) (get-heur x2d)›
    using that
    by (auto simp: twl-st-heur-restart-ana-def twl-st-heur-restart-def
      all-init-atms-st-def get-unit-init-clss-wl-alt-def)
   moreover have ‹the x1c ∈# all-init-atms-st x2b›
    using that by (auto simp: all-init-atms-st-alt-def)
   ultimately have ‹is-marked-added-heur-pre (get-heur x2d) (the x1c)›
    unfolding is-marked-added-heur-pre-def
    by (auto simp: heuristic-rel-def is-marked-added-heur-pre-stats-def
      heuristic-rel-stats-def phase-saving-def atms-of-\mathcal{L}_{all}-\mathcal{A}_{in})
   then show ?thesis
    unfolding mop-is-marked-added-heur-st-def mop-is-marked-added-heur-def
    by (auto intro!: RETURN-RES-refine)
  qed
  have skip: ‹(skip-lit, skip)
   ∈ {(a, b). (¬should-eliminate-pure-st S, b) ∈ bool-rel} ⟹
   skip ∈ UNIV ⟹ (¬ should-eliminate-pure-st S) = skip› for skip-lit skip
   by auto
  have last-step: ‹do {
  - ← ASSERT (mark-duplicated-binary-clauses-as-garbage-pre-wl-heur S);
    let skip = should-eliminate-pure-st S;
   (CS::(nat literal ⇒ (nat × bool) option)× nat) ← create (length (get-watched-wl-heur S));
   (CS, S) ← iterate-over-VMTFC
   (λA (CS, Sa). do {
     - ← ASSERT (get-vmtf-heur-array S = get-vmtf-heur-array Sa);
     - ← mop-is-marked-added-heur-st Sa A;
     if ¬skip then RETURN (CS, Sa)
     else do {
       ASSERT (length (get-clauses-wl-heur Sa) ≤ length (get-clauses-wl-heur S) ∧ learned-clss-count
Sa ≤ learned-clss-count S);
      (CS, Sa) ← isa-deduplicate-binary-clauses-wl (Pos A) CS Sa;
       ASSERT (length (get-clauses-wl-heur Sa) ≤ length (get-clauses-wl-heur S) ∧ learned-clss-count
Sa ≤ learned-clss-count S);
      (CS, Sa) ← isa-deduplicate-binary-clauses-wl (Neg A) CS Sa;
      - ← ASSERT (get-vmtf-heur-array S = get-vmtf-heur-array Sa);
      RETURN (CS, Sa)
       }
    })
   (λ(CS, Sa). get-vmtf-heur-array S = get-vmtf-heur-array Sa)
   (λ(CS, S). get-conflict-wl-is-None-heur S)
   (get-vmtf-heur-array S, Some (get-vmtf-heur-fst S)) (CS, S);
   RETURN S
   } ≤ ⇓ (twl-st-heur-restart-ana′ r u)
    (do {
     x ← ASSERT (mark-duplicated-binary-clauses-as-garbage-pre-wl S′);
     let - = True;
     let - = (λ-::nat literal. None :: (nat × bool) option);
     x ← iterate-over-VMTFC
      (λL (S). do {
        - ← ASSERT (L ∈# atm-of ‘# all-init-lits-of-wl S);
        skip ← RES UNIV;
        if skip then RETURN (S)
```

```
        else do {
         (S) ← deduplicate-binary-clauses-wl (Pos L) S;
         (S) ← deduplicate-binary-clauses-wl (Neg L) S;
         RETURN (S)
        }
       })
      (λ(x). ∃B'. mark-duplicated-binary-clauses-as-garbage-wl-inv
          (all-init-atms-st S') S' (x, B'))
      (λ(x). get-conflict-wl x = None) (get-vmtf-heur-array S, Some (get-vmtf-heur-fst S)) (S');
     RETURN x
         })› for CS
   supply [[goals-limit=1]]
   unfolding iterate-over-VMTFC-def
   apply (refine-vcg
   isa-deduplicate-binary-clauses-mark-duplicated-binary-clauses-as-garbage-wl[where r=‹length (get-clauses-wl-heur
S)› and u=‹learned-clss-count S› and
     ns = ‹get-vmtf-heur S› and lw=‹length (get-watched-wl-heur S)›])
   subgoal using assms unfolding mark-duplicated-binary-clauses-as-garbage-pre-wl-heur-def
     by fast
   apply (rule create)
   apply (rule init)
   subgoal by (use  in ‹auto simp: get-vmtf-heur-fst-def›)
   subgoal by auto
   subgoal by (auto simp: get-vmtf-heur-array-def)
   subgoal
     apply auto
    apply (subst (asm) get-conflict-wl-is-None-heur-get-conflict-wl-is-None-ana[THEN fref-to-Down-unRET-Id])
     apply (auto simp: get-conflict-wl-is-None-def)
    apply (subst get-conflict-wl-is-None-heur-get-conflict-wl-is-None-ana[THEN fref-to-Down-unRET-Id])
     apply (auto simp: get-conflict-wl-is-None-def)
     done
   subgoal by auto
   subgoal by auto
   subgoal by auto
   subgoal by auto
   apply (rule GC-required-skip; assumption?)
   apply (rule skip; assumption)
   apply (rule rel0; assumption)
   subgoal by (auto simp add: twl-st-heur-restart-ana-def)
   subgoal by (auto simp add: twl-st-heur-restart-ana-def)
   subgoal by simp
   subgoal by simp
   subgoal by auto
   subgoal by (auto simp add: twl-st-heur-restart-ana-def)
   subgoal by auto
   subgoal by auto
   subgoal by auto
   subgoal by simp
   subgoal premises p
     using p(25−) unfolding get-vmtf-heur-array-def by simp
   apply (rule rel; assumption?)
   subgoal by auto
   apply (rule rel2; assumption)
   subgoal using assms by (auto simp: twl-st-heur-restart-ana-def)
   done
```

let *?vm* = ‹*get-vmtf-heur S*›
obtain *M′* **where** *vmtf′*: ‹(*get-vmtf-heur-array S*, *fst* (*snd* (*bump-get-heuristics ?vm*)),
get-vmtf-heur-fst S, *fst* (*snd* (*snd* (*snd* (*bump-get-heuristics ?vm*)))),
snd (*snd* (*snd* (*snd* (*bump-get-heuristics ?vm*))))) ∈ *vmtf* (*atm-of* ‘# *all-init-lits-of-wl S′*) *M′*›
using *1* **unfolding** *bump-heur-def get-vmtf-heur-array-def bump-get-heuristics-def get-vmtf-heur-fst-def*
by (*cases* ‹*bump-get-heuristics ?vm*›) (*auto simp*: *bump-get-heuristics-def bumped-vmtf-array-fst-def*
split: *if-splits*)

show *?thesis*
unfolding *isa-mark-duplicated-binary-clauses-as-garbage-wl-alt-def*
apply (*rule order-trans*)
defer
apply (*rule ref-two-step′*)
apply (*rule subset-refl*)
apply (*rule mark-duplicated-binary-clauses-as-garbage-wl2-mark-duplicated-binary-clauses-as-garbage-wl*[*unfolded Down-id-eq*])
apply (*rule order-trans*)
defer
apply (*rule ref-two-step′*)
apply (*rule subset-refl*)
apply (*rule mark-duplicated-binary-clauses-as-garbage-wl2-ge-*$\mathcal{L}_{all}$[*unfolded Down-id-eq*])
defer
apply (*rule order-trans*)
defer
apply (*rule ref-two-step′*)
apply (*rule subset-refl*)
apply (*rule bind-refine*[*of - - - - Id*, *unfolded Down-id-eq*])
apply (*rule Id-refine*)
apply (*rule iterate-over-VMTFC-iterate-over-*$\mathcal{L}_{all}$*C*[*unfolded Down-id-eq*,
where *I* = ‹$\lambda x.\ \exists \mathcal{B}'.$ *mark-duplicated-binary-clauses-as-garbage-wl-inv* (*all-init-atms-st S′*) *S′* (*x*, $\mathcal{B}'$)› **and**
P = ‹$\lambda x.$ *get-conflict-wl x* = *None*› **for** $\mathcal{B}$])
apply (*rule vmtf′*)
apply (*solves* ‹*use 2 in* ‹*auto simp*: *all-init-atms-st-alt-def*››)
apply (*solves* ‹*use 3 in* ‹*auto simp*: *all-init-atms-st-alt-def*››)
apply (*solves auto*)
apply (*solves auto*)
apply (*rule last-step*[*THEN order-trans*])
by *auto*
qed

lemma *isa-mark-duplicated-binary-clauses-as-garbage-wl2-isa-mark-duplicated-binary-clauses-as-garbage-wl*:
‹*isa-mark-duplicated-binary-clauses-as-garbage-wl2 S* ≤
⇓*Id* (*isa-mark-duplicated-binary-clauses-as-garbage-wl S*)›
proof −
have *H*: ‹*a*=*b*⟹ (*a*,*b*)∈*Id*› ‹*c*=*d* ⟹ *c* ≤⇓*Id d*› **for** *a b c d*
by *auto*
have *K*: ‹(*Sb*, *Sc*) ∈ *Id* ⟹(*CSb*, *CSc*) ∈ *Id* ⟹
get-vmtf-heur-array S = *get-vmtf-heur-array Sb* ⟹
((*CSb*, *Sb*), (*CSc*, *Sc*)) ∈ {((*CSa*, *a*), (*CSb*, *b*)). (*CSa*, *CSb*)∈*Id* ∧ (*a*,*b*)∈*Id* ∧ *get-vmtf-heur-array S* = *get-vmtf-heur-array a*}› **for** *Sb Sc CSb CSc*
by *auto*
show *?thesis*
unfolding *isa-mark-duplicated-binary-clauses-as-garbage-wl2-def*
isa-mark-duplicated-binary-clauses-as-garbage-wl-def nres-monad3 Let-def

apply *refine-rcg*
apply (*rule H*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
apply (*rule H*)
subgoal by *auto*
subgoal by *auto*
apply (*rule K*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
apply (*rule H*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
apply (*rule H*)
subgoal by *auto*
subgoal by *auto*
apply (*rule K*; *assumption?*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

lemma *isa-mark-duplicated-binary-clauses-as-garbage-wl-mark-duplicated-binary-clauses-as-garbage-wl2*:
assumes ‹(*S*, *S'*) ∈ *twl-st-heur-restart-ana' r u*›
shows ‹*isa-mark-duplicated-binary-clauses-as-garbage-wl2 S* ≤
⇓(*twl-st-heur-restart-ana' r u*) (*mark-duplicated-binary-clauses-as-garbage-wl S'*)›
apply (*rule order-trans*)
apply (*rule isa-mark-duplicated-binary-clauses-as-garbage-wl2-isa-mark-duplicated-binary-clauses-as-garbage-wl*)
unfolding *Down-id-eq*
apply (*rule isa-mark-duplicated-binary-clauses-as-garbage-wl-mark-duplicated-binary-clauses-as-garbage-wl*)
apply (*rule assms*)+
done

end
theory *IsaSAT-Simplify-Binaries-LLVM*
imports
IsaSAT-Simplify-Clause-Units-LLVM
IsaSAT-Simplify-Binaries-Defs
IsaSAT-Proofs-LLVM
begin

abbreviation *ahm-assn* :: ‹-› **where**
‹*ahm-assn* ≡ *larray64-assn* (*sint-assn' TYPE*(*64*)) ×$_a$ *al-assn' TYPE*(*64*) (*snat-assn' TYPE*(*64*))›

sepref-def *ahm-create-code*

is ‹*ahm-create*›
:: ‹$(snat\text{-}assn'\ TYPE(64))^k \to_a ahm\text{-}assn$›
unfolding *ahm-create-def larray-fold-custom-replicate al-fold-custom-empty*[**where** *'l=64*]
apply (*annot-sint-const* ‹*TYPE*(*64*)›)
by *sepref*

definition *encoded-irred-indices* **where**
‹*encoded-irred-indices* = {(*a*, *b*::*nat* × *bool*). *a* ≤ *int snat64-max* ∧ −*a* ≤ *int snat64-max* ∧ (*snd b* ⟷ *a* > *0*) ∧ *fst b* = (*if a* < *0 then nat* (−*a*) *else nat a*) ∧ *fst b* ≠ *0*}›

sepref-def *ahm-is-marked-code*
is ‹*uncurry ahm-is-marked*›
:: ‹$(ahm\text{-}assn)^k *_a\ unat\text{-}lit\text{-}assn^k \to_a bool1\text{-}assn$›
unfolding *ahm-is-marked-def*
apply (*annot-sint-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *ahm-get-marked-code*
is ‹*uncurry ahm-get-marked*›
:: ‹$(ahm\text{-}assn)^k *_a\ unat\text{-}lit\text{-}assn^k \to_a sint\text{-}assn'\ TYPE(64)$›
unfolding *ahm-get-marked-def*
by *sepref*

sepref-def *ahm-empty-code*
is ‹*ahm-empty*›
:: ‹$(ahm\text{-}assn)^d \to_a ahm\text{-}assn$›
unfolding *ahm-empty-def*
apply (*annot-sint-const* ‹*TYPE*(*64*)›)
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

definition *encoded-irred-index-irred* **where**
‹*encoded-irred-index-irred a* = *snd a*›

definition *encoded-irred-index-irred-int* **where**
‹*encoded-irred-index-irred-int a* = (*a* > *0*)›

lemma *encoded-irred-index-irred*:
‹(*encoded-irred-index-irred-int*, *encoded-irred-index-irred*) ∈ *encoded-irred-indices* → *bool-rel*›
by (*auto simp*: *encoded-irred-indices-def encoded-irred-index-irred-int-def*
encoded-irred-index-irred-def)

definition *encoded-irred-index-get* **where**
‹*encoded-irred-index-get a* = *fst a*›

definition *encoded-irred-index-get-int* **where**
‹*encoded-irred-index-get-int a* = *do* {*ASSERT* (*a* ≤ *int snat64-max* ∧ −*a* ≤ *int snat64-max*); *RETURN* (*if a* > *0 then nat a else nat* (−*a*))}›

lemma *encoded-irred-index-get*:
‹(*encoded-irred-index-get-int*, *RETURN o encoded-irred-index-get*) ∈ *encoded-irred-indices* → ⟨*nat-rel*⟩*nres-rel*›
by (*auto simp*: *encoded-irred-indices-def encoded-irred-index-get-int-def*
encoded-irred-index-get-def intro!: *nres-relI*)

sepref-def *encoded-irred-index-irred-int-impl*
is ‹*RETURN o encoded-irred-index-irred-int*›
:: ‹$(sint\text{-}assn'\ TYPE(64))^k \to_a bool1\text{-}assn$›
unfolding *encoded-irred-index-irred-int-def*
apply (*annot-sint-const* ‹*TYPE*(64)›)
by *sepref*

lemma *nat-sint-snat*: ‹$0 \le sint\ xi \Longrightarrow (nat\ (sint\ xi) = snat\ xi)$›
by (*auto simp*: *snat-def*)

lemma [*sepref-fr-rules*]:
‹$(Mreturn, RETURN\ o\ nat) \in [\lambda a.\ a \ge 0]_a\ (sint\text{-}assn'\ TYPE(64))^k \to sint64\text{-}nat\text{-}assn$›
apply *sepref-to-hoare*
apply *vcg*
apply (*auto simp*: *sint-rel-def ENTAILS-def snat-rel-def snat.rel-def br-def sint.rel-def*
pure-true-conv Exists-eq-simp snat-invar-def word-msb-sint nat-sint-snat)
done
lemma [*sepref-fr-rules*]:
‹$(Mreturn\ o\ (\lambda x.\ -x), RETURN\ o\ uminus) \in [\lambda a.\ a \le int\ snat64\text{-}max \wedge -a \le int\ snat64\text{-}max]_a$
$(sint\text{-}assn'\ TYPE(64))^k \to (sint\text{-}assn'\ TYPE(64))$›
apply *sepref-to-hoare*
apply *vcg*
subgoal for *x xi asf s*
using *sdiv-word-min'*[*of xi 1*] *sdiv-word-max'*[*of xi 1*]
apply (*auto simp*: *sint-rel-def ENTAILS-def snat-rel-def snat.rel-def br-def sint.rel-def*
pure-true-conv Exists-eq-simp snat-invar-def word-msb-sint nat-sint-snat
signed-arith-ineq-checks-to-eq word-size snat64-max-def word-size)
apply (*subst signed-arith-ineq-checks-to-eq*[*symmetric*])
apply (*auto simp*: *word-size pure-true-conv*)
done
done

lemma *encoded-irred-index-get-int-alt-def*:
‹*encoded-irred-index-get-int a = do* {*ASSERT* ($a \le int\ snat64\text{-}max \wedge -a \le int\ snat64\text{-}max$); *RETURN*
(*if a > 0 then nat a else nat* $(0-a)$)}›
unfolding *encoded-irred-index-get-int-def* **by** *auto*
sepref-def *encoded-irred-index-irred-get-impl*
is ‹*encoded-irred-index-get-int*›
:: ‹$(sint\text{-}assn'\ TYPE(64))^k \to_a sint64\text{-}nat\text{-}assn$›
unfolding *encoded-irred-index-get-int-alt-def*
apply (*annot-sint-const* ‹*TYPE*(64)›)
by *sepref*

lemmas [*sepref-fr-rules*] =
encoded-irred-index-irred-get-impl.refine[*FCOMP encoded-irred-index-get*]
encoded-irred-index-irred-int-impl.refine[*FCOMP encoded-irred-index-irred*]

definition *encoded-irred-index-set* **where**
‹*encoded-irred-index-set a b* = (*a*::*nat*, *b*::*bool*)›

definition *encoded-irred-index-set-int* **where**
‹*encoded-irred-index-set-int a b = do* { (*if b then RETURN* (*int a*) *else RETURN* (− *int a*))}›

lemma *encoded-irred-index-set*:
‹(*uncurry encoded-irred-index-set-int*, *uncurry* (*RETURN oo encoded-irred-index-set*)) ∈ [λ(*a*,*b*). *a* ≠

$0 \wedge a \leq$ *snat64-max*$]_f$ *nat-rel* $\times_r$ *bool-rel* $\to \langle$*encoded-irred-indices*$\rangle$*nres-rel*›
by (*clarsimp simp*: *encoded-irred-indices-def encoded-irred-index-set-int-def encoded-irred-index-set-def intro*!: *nres-relI frefI*)

lemma *int-snat-sint*: ‹¬ *sint xi* < *0* ⟹ *int* (*snat* (*xi*::*64 word*)) = *sint xi*›
by (*auto simp*: *snat-def*)

lemma [*sepref-fr-rules*]:
‹(*Mreturn*, *RETURN o int*) ∈ (*snat-assn*′ *TYPE*(*64*))^k →$_a$ (*sint-assn*′ *TYPE*(*64*))›
apply *sepref-to-hoare*
apply *vcg*
apply (*auto simp*: *sint-rel-def ENTAILS-def snat-rel-def snat.rel-def br-def sint.rel-def pure-true-conv Exists-eq-simp snat-invar-def word-msb-sint nat-sint-snat int-snat-sint*)
done

sepref-register *uminus* :: *int* ⇒ *int int*
lemma *encoded-irred-index-set-int-alt-def*:
‹*encoded-irred-index-set-int a b* = *do* { (*if b then RETURN* (*int a*) *else RETURN* (*0* − *int a*))}›
by (*auto simp*: *encoded-irred-index-set-int-def*)

sepref-def *encoded-irred-index-set-int-impl*
is ‹*uncurry encoded-irred-index-set-int*›
:: ‹*sint64-nat-assn*^{k} *∗*$_a$ *bool1-assn*^{k} →$_a$ (*sint-assn*′ *TYPE*(*64*))›
unfolding *encoded-irred-index-set-int-alt-def*
apply (*annot-sint-const* ‹*TYPE*(*64*)›)
by *sepref*

lemmas [*sepref-fr-rules*] =
encoded-irred-index-set-int-impl.refine[*FCOMP encoded-irred-index-set*]

sepref-register *is-marked set-marked update-marked*

sepref-def *ahm-set-marked-code*
is ‹*uncurry2 ahm-set-marked*›
:: ‹*ahm-assn*^{d} *∗*$_a$ *unat-lit-assn*^{k} *∗*$_a$ (*sint-assn*′ *TYPE*(*64*))^k →$_a$ *ahm-assn*›
unfolding *ahm-set-marked-def*
by *sepref*

sepref-def *ahm-update-marked-code*
is ‹*uncurry2 ahm-update-marked*›
:: ‹*ahm-assn*^{d} *∗*$_a$ *unat-lit-assn*^{k} *∗*$_a$ (*sint-assn*′ *TYPE*(*64*))^k →$_a$ *ahm-assn*›
unfolding *ahm-update-marked-def*
by *sepref*

definition *ahm-full-assn* :: ‹-› **where**
‹*ahm-full-assn* = *hr-comp* (*larray64-assn* (*sint-assn*′ *TYPE*(*64*)) ×$_a$ *Size-Ordering-it.arr-assn*)
(*array-hash-map-rel encoded-irred-indices*)›

schematic-goal *ahm-full-assn-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE ahm-full-assn ?a*›
unfolding *ahm-full-assn-def* **by** *synthesize-free*

lemma *ahm-set-marked-set-marked*:
‹(*uncurry2 ahm-set-marked*, *uncurry2 set-marked*)
∈ (*array-hash-map-rel encoded-irred-indices*) ×$_f$ *nat-lit-lit-rel* ×$_f$ *encoded-irred-indices* → ⟨*array-hash-map-rel encoded-irred-indices*⟩*nres-rel*›

proof −
  have *H*: ‹(*0*, *a*) ∉ *encoded-irred-indices*› **for** *a*
    by (*auto simp*: *encoded-irred-indices-def*)
  show *?thesis*
    unfolding *fref-param1*
    apply (*rule ahm-set-marked-set-marked*)
    apply (*rule H*)
    done
qed

lemma *ahm-update-marked-update-marked*:
‹(*uncurry2 ahm-update-marked*, *uncurry2 update-marked*)
    ∈ (*array-hash-map-rel encoded-irred-indices*) ×$_f$ *nat-lit-lit-rel* ×$_f$ *encoded-irred-indices* → ⟨*array-hash-map-rel encoded-irred-indices*⟩*nres-rel*›
proof −
  have *H*: ‹(*0*, *a*) ∉ *encoded-irred-indices*› **for** *a*
    by (*auto simp*: *encoded-irred-indices-def*)
  show *?thesis*
    unfolding *fref-param1*
    apply (*rule ahm-update-marked-update-marked*)
    apply (*rule H*)
    done
qed

thm
  ahm-create-code.refine[*FCOMP ahm-create-create*[**where** *R*= *encoded-irred-indices*]]

lemmas [*unfolded ahm-full-assn-def*[*symmetric*], *sepref-fr-rules*] =
  ahm-create-code.refine[*FCOMP ahm-create-create*[**where** *R*= *encoded-irred-indices*]]
  ahm-empty-code.refine[*FCOMP ahm-empty-empty*[**where** *R* = *encoded-irred-indices*]]
  ahm-is-marked-code.refine[*FCOMP ahm-is-marked-is-marked*[**where** *R* = *encoded-irred-indices*]]
  ahm-get-marked-code.refine[*FCOMP ahm-get-marked-get-marked*[**where** *R* = *encoded-irred-indices*]]
  ahm-empty-code.refine[*FCOMP ahm-empty-empty*, **where** *R19* = *encoded-irred-indices*]
  ahm-set-marked-code.refine[*FCOMP ahm-set-marked-set-marked*]
  ahm-update-marked-code.refine[*FCOMP ahm-update-marked-update-marked*]

sepref-register *create encoded-irred-index-set encoded-irred-index-get*
sepref-register *uminus-lit*: *uminus* :: *nat literal* ⇒ -

lemma *isa-clause-remove-duplicate-clause-wl-alt-def*:
  ‹*isa-clause-remove-duplicate-clause-wl C S* = (*do*{
    - ← *log-del-clause-heur S C*;
    let (*N'*, *S*) = *extract-arena-wl-heur S*;
    st ← *mop-arena-status N' C*;
    let st = *st* = *IRRED*;
    ASSERT (*mark-garbage-pre* (*N'*, *C*) ∧ *arena-is-valid-clause-vdom* (*N'*) *C*);
    let N' = *extra-information-mark-to-delete* (*N'*) *C*;
    let (*lcount*, *S*) = *extract-lcount-wl-heur S*;
    ASSERT(¬*st* ⟶ *clss-size-lcount lcount* ≥ *1*);
    let lcount = (*if st then lcount else* (*clss-size-decr-lcount lcount*));
    let (*stats*, *S*) = *extract-stats-wl-heur S*;
    let stats = *incr-binary-red-removed* (*if st then decr-irred-clss stats else stats*);
    let S = *update-arena-wl-heur N' S*;
    let S = *update-lcount-wl-heur lcount S*;
    let S = *update-stats-wl-heur stats S*;

RETURN S
})›
by (*auto simp*: *isa-clause-remove-duplicate-clause-wl-def*
state-extractors split: *isasat-int-splits*)

sepref-def *isa-clause-remove-duplicate-clause-wl-impl*
is ‹*uncurry isa-clause-remove-duplicate-clause-wl*›
:: ‹[λ(*L*, *S*). *length* (*get-clauses-wl-heur S*) ≤ *snat64-max* ∧ *learned-clss-count S* ≤ *unat64-max*]$_a$
sint64-nat-assn^k *$*_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *isa-clause-remove-duplicate-clause-wl-alt-def*
by *sepref*

sepref-register *isa-binary-clause-subres-wl*

sepref-register *incr-binary-unit-derived*
lemma *isa-binary-clause-subres-wl-alt-def*:
‹*isa-binary-clause-subres-wl C L L' S*$_0$ = *do* {
ASSERT (*isa-binary-clause-subres-lits-wl-pre C L L' S*$_0$);
let (*M*, *S*) = *extract-trail-wl-heur S*$_0$;
M ← *cons-trail-Propagated-tr L 0 M*;
let (*lcount*, *S*) = *extract-lcount-wl-heur S*;
ASSERT (*lcount* = *get-learned-count S*$_0$);
let (*N'*, *S*) = *extract-arena-wl-heur S*;
st ← *mop-arena-status N' C*;
let st = *st* = *IRRED*;
ASSERT (*mark-garbage-pre* (*N'*, *C*) ∧ *arena-is-valid-clause-vdom* (*N'*) *C*);
let N' = *extra-information-mark-to-delete* (*N'*) *C*;
ASSERT(¬*st* ⟶ (*clss-size-lcount lcount* ≥ 1 ∧ *clss-size-lcountUEk* (*clss-size-decr-lcount lcount*)
< *learned-clss-count S*$_0$));
let lcount = (*if st then lcount else* (*clss-size-incr-lcountUEk* (*clss-size-decr-lcount lcount*)));
let (*stats*, *S*) = *extract-stats-wl-heur S*;
let stats = *incr-binary-unit-derived* (*if st then decr-irred-clss stats else stats*);
let stats = *incr-units-since-last-GC* (*incr-uset stats*);
let S = *update-trail-wl-heur M S*;
let S = *update-arena-wl-heur N' S*;
let S = *update-lcount-wl-heur lcount S*;
let S = *update-stats-wl-heur stats S*;
let - = *log-unit-clause L*;
RETURN S
}›
by (*subst Let-def*[*of* ‹*log-unit-clause L*›])
(*simp add*: *isa-binary-clause-subres-wl-def learned-clss-count-def*
state-extractors split: *isasat-int-splits*)

sepref-def *isa-binary-clause-subres-wl-impl*
is ‹*uncurry3 isa-binary-clause-subres-wl*›
:: ‹[λ(((*C*,*L*), *L'*), *S*). *length* (*get-clauses-wl-heur S*) ≤ *snat64-max* ∧ *learned-clss-count S* ≤ *unat64-max*]$_a$
sint64-nat-assn^k *$*_a$ *unat-lit-assn*^{k} *$*_a$ *unat-lit-assn*^{k} *$*_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *isa-binary-clause-subres-wl-alt-def*[*abs-def*]
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-register *should-eliminate-pure-st*

sepref-def *should-eliminate-pure-st-impl*
is ‹*RETURN o should-eliminate-pure-st*›
:: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *bool1-assn*›
unfolding *should-eliminate-pure-st-def*
by *sepref*

sepref-def *isa-deduplicate-binary-clauses-wl-code*
is ‹*uncurry2 isa-deduplicate-binary-clauses-wl*›
:: ‹[λ((*L*, *CS*), *S*). *length* (*get-clauses-wl-heur S*) ≤ *snat64-max* ∧ *learned-clss-count S* ≤ *unat64-max*]$_a$
unat-lit-assn^k $*_a$ *ahm-full-assn*^{d} $*_a$ *isasat-bounded-assn*^{d} →
ahm-full-assn $\times_a$ *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *isa-deduplicate-binary-clauses-wl-def*
mop-polarity-st-heur-def[*symmetric*]
mop-arena-status-st-def[*symmetric*]
tri-bool-eq-def[*symmetric*]
encoded-irred-index-set-def[*symmetric*]
encoded-irred-index-irred-def[*symmetric*]
encoded-irred-index-get-def[*symmetric*]
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-register *get-bump-heur-array-nth get-vmtf-heur-fst*
isa-deduplicate-binary-clauses-wl

lemma *Massign-split*: ‹*do*{ *x* ← (*M* :: - *nres*); *f x*} = *do*{(*a*,*b*) ← *M*; *f* (*a*,*b*)}›
by *auto*

lemma *isa-mark-duplicated-binary-clauses-as-garbage-wl2-alt-def*:
‹*isa-mark-duplicated-binary-clauses-as-garbage-wl2* S_0 = (*do* {
let ns = *get-vmtf-heur-array* S_0;
ASSERT (*mark-duplicated-binary-clauses-as-garbage-pre-wl-heur* S_0);
let skip = *should-eliminate-pure-st* S_0;
CS ← *create* (*length* (*get-watched-wl-heur* S_0));
(-, *CS*, *S*) ← *WHILE*$_T$ λ(*n*,*CS*, *S*). *get-vmtf-heur-array* S_0 = (*get-vmtf-heur-array S*)(λ(*n*, *CS*, *S*). *n* ≠ *None* ∧ *get-conflict-wl-is-None-heur S*)
(λ(*n*, *CS*, *S*). *do* {
ASSERT (*n* ≠ *None*);
let A = *the n*;
ASSERT (*A* < *length* (*get-vmtf-heur-array S*));
ASSERT (*A* ≤ *unat32-max div 2*);
added ← *mop-is-marked-added-heur-st S A*;
if ¬*skip then RETURN* (*get-next* (*get-vmtf-heur-array S* ! *A*), *CS*, *S*)
else do {
ASSERT (*length* (*get-clauses-wl-heur S*) ≤ *length* (*get-clauses-wl-heur* S_0) ∧ *learned-clss-count S* ≤ *learned-clss-count* S_0);
(*CS*, *S*) ← *isa-deduplicate-binary-clauses-wl* (*Pos A*) *CS S*;
ASSERT (*length* (*get-clauses-wl-heur S*) ≤ *length* (*get-clauses-wl-heur* S_0) ∧ *learned-clss-count S* ≤ *learned-clss-count* S_0);
(*CS*, *S*) ← *isa-deduplicate-binary-clauses-wl* (*Neg A*) *CS S*;
ASSERT (*ns* = *get-vmtf-heur-array S*);
RETURN (*get-next* (*get-vmtf-heur-array S* ! *A*), *CS*, *S*)
}
})

```
    (Some (get-vmtf-heur-fst S₀), CS, S₀);
    RETURN S
  })›
    unfolding isa-mark-duplicated-binary-clauses-as-garbage-wl2-def bind-to-let-conv
      nres-monad3
   apply (simp add: case-prod-beta cong: if-cong)
   unfolding bind-to-let-conv Let-def prod.simps
   apply (subst Massign-split[of ‹isa-deduplicate-binary-clauses-wl (Pos -) - -›])
   unfolding prod.simps nres-monad3
   apply (subst (2) Massign-split[of ‹- :: (- × isasat) nres›])
   unfolding prod.simps nres-monad3
   apply (auto intro!: bind-cong[OF refl] cong: if-cong)
   done

sepref-def isa-deduplicate-binary-clauses-code
  is isa-mark-duplicated-binary-clauses-as-garbage-wl2
  :: ‹[λS. length (get-clauses-wl-heur S) ≤ snat64-max ∧ learned-clss-count S ≤ unat64-max]ₐ
     isasat-bounded-assnᵈ → isasat-bounded-assn›
  unfolding isa-mark-duplicated-binary-clauses-as-garbage-wl2-alt-def
    get-bump-heur-array-nth-def[symmetric] atom.fold-option nres-monad3
    length-watchlist-def[unfolded length-ll-def, symmetric]
    length-watchlist-raw-def[symmetric]
  apply (rewrite at ‹let - = get-vmtf-heur-array - in -› Let-def)
  unfolding if-False nres-monad3
  supply [[goals-limit=1]]
  by sepref

end
theory IsaSAT-Simplify-Pure-Literals-Defs
  imports IsaSAT-Setup
    IsaSAT-Restart-Defs
begin
  definition isa-pure-literal-count-occs-clause-wl-invs :: ‹- ⇒› where
  ‹isa-pure-literal-count-occs-clause-wl-invs C S occs =
  (λ(i, occs2, remaining). ∃S' r u occs' occs2'. (S, S') ∈ twl-st-heur-restart-ana' r u ∧
    (occs, occs') ∈ ⟨bool-rel⟩map-fun-rel (D₀ (all-init-atms-st S')) ∧
    (occs2, occs2') ∈ ⟨bool-rel⟩map-fun-rel (D₀ (all-init-atms-st S')) ∧
    pure-literal-count-occs-clause-wl-invs C S' occs' (i, occs2'))›

definition isa-pure-literal-count-occs-clause-wl-pre :: ‹- ⇒› where
  ‹isa-pure-literal-count-occs-clause-wl-pre C S occs =
  (∃S' r u occs'. (S, S') ∈ twl-st-heur-restart-ana' r u ∧
    (occs, occs') ∈ ⟨bool-rel⟩map-fun-rel (D₀ (all-init-atms-st S')) ∧
    pure-literal-count-occs-clause-wl-pre C S' occs')›

definition isa-pure-literal-count-occs-clause-wl :: ‹nat ⇒ isasat ⇒ - ⇒ 64 word ⇒ -› where
  ‹isa-pure-literal-count-occs-clause-wl C S occs remaining = do {
    ASSERT (isa-pure-literal-count-occs-clause-wl-pre C S occs);
    m ← mop-arena-length-st S C;
    (i, occs, -) ← WHILE_T^isa-pure-literal-count-occs-clause-wl-invs C S occs (λ(i, occs, remaining). i < m)
      (λ(i, occs, remaining). do {
        ASSERT (i < m);
        L ← mop-access-lit-in-clauses-heur S C i;
        ASSERT (nat-of-lit L < length occs);
        ASSERT (nat-of-lit (−L) < length occs);
      let remaining = (if ¬occs!(nat-of-lit L) ∧ occs ! (nat-of-lit (−L)) then remaining−1 else remaining);
```

```
    let occs = occs [nat-of-lit L := True];
    RETURN (i+1, occs, remaining)
   })
   (0, occs, remaining);
  RETURN (occs, remaining)
 }›

definition isa-pure-literal-count-occs-wl :: ‹isasat ⇒ -› where
  ‹isa-pure-literal-count-occs-wl S = do {
  let xs = get-avdom S @ get-ivdom S;
  let m = length (xs);
  let remaining = ((of-nat (length (get-watched-wl-heur S)) :: 64 word) >> 1) − units-since-beginning-st
S;
  let abort = (remaining ≤ 0);
  let occs = replicate (length (get-watched-wl-heur S)) False;
  ASSERT (m ≤ length (get-clauses-wl-heur S) − 2);
  (-, occs, abort) ← WHILE_T(λ(i, occs, remaining). i < m ∧ remaining > 0)
    (λ(i, occs, remaining). do {
      ASSERT (i ≤ length (get-clauses-wl-heur S) − 2);
      ASSERT ((i < length (get-avdom S) ⟶ access-avdom-at-pre S i) ∧
        (i ≥ length (get-avdom S) ⟶ access-ivdom-at-pre S (i − length-avdom S)));
      let C = (get-avdom S @ get-ivdom S) ! i;
      E ← mop-arena-status (get-clauses-wl-heur S) C;
      if (E = IRRED) then do {
        (occs, remaining) ← isa-pure-literal-count-occs-clause-wl C S occs remaining;
        let abort = (remaining ≤ 0);
        RETURN (i+1, occs, remaining)
      } else RETURN (i+1, occs, remaining)
    })
    (0, occs, remaining);
  RETURN (abort ≤ 0, occs)
  }›


definition isa-pure-literal-elimination-round-wl-pre where
  ‹isa-pure-literal-elimination-round-wl-pre S ⟷
  (∃ T r u. (S, T) ∈ twl-st-heur-restart-ana′ r u ∧ pure-literal-elimination-round-wl-pre T)›

definition isa-pure-literal-deletion-wl-pre :: ‹-› where
  ‹isa-pure-literal-deletion-wl-pre S ⟷
  (∃ T r u. (S, T) ∈ twl-st-heur-restart-ana′ r u ∧ pure-literal-deletion-wl-pre T)›


definition isa-propagate-pure-bt-wl
  :: ‹nat literal ⇒ isasat ⇒ isasat nres›
  where
    ‹isa-propagate-pure-bt-wl = (λL S. do {
      let M = get-trail-wl-heur S;
      let stats = get-stats-heur S;
      ASSERT(0 ≠ DECISION-REASON);
      ASSERT(cons-trail-Propagated-tr-pre ((L, 0::nat), M));
      M ← cons-trail-Propagated-tr (L) 0 M;
      let stats = incr-units-since-last-GC (incr-uset (incr-purelit-elim stats));
      let S = set-stats-wl-heur stats S;
      let S = set-trail-wl-heur M S;
      RETURN S})
```

›

definition *isa-pure-literal-deletion-wl-raw* :: ‹*bool list* ⇒ *isasat* ⇒ (*64 word* × *isasat*) *nres*› **where**
‹*isa-pure-literal-deletion-wl-raw occs* S_0 = (*do* {
ASSERT (*isa-pure-literal-deletion-wl-pre* S_0);
(*eliminated*, *S*) ← *iterate-over-VMTF*
(λ*A* (*eliminated*, *T*). *do* {
ASSERT (*get-vmtf-heur-array* S_0 = *get-vmtf-heur-array T*);
ASSERT (*nat-of-lit* (*Pos A*) < *length occs*);
ASSERT (*nat-of-lit* (*Neg A*) < *length occs*);
let L = (*if occs* ! (*nat-of-lit* (*Pos A*)) ∧ ¬ *occs* ! (*nat-of-lit* (*Neg A*))
then Pos A else Neg A);
ASSERT (*nat-of-lit* (−*L*) < *length occs*);
val ← *mop-polarity-pol* (*get-trail-wl-heur T*) *L*;
if ¬*occs* ! (*nat-of-lit* (−*L*)) ∧ *val* = *None*
then do {*S* ← *isa-propagate-pure-bt-wl L T*;
ASSERT (*get-vmtf-heur-array* S_0 = *get-vmtf-heur-array S*);
RETURN (*eliminated* + *1*, *S*)}
else RETURN (*eliminated*, *T*)
})
(λ(-, *S*). *get-vmtf-heur-array* S_0 = (*get-vmtf-heur-array S*))
(*get-vmtf-heur-array* S_0, *Some* (*get-vmtf-heur-fst* S_0)) (*0* :: *64 word*, S_0);
RETURN (*eliminated*, *S*)
})›

definition *isa-pure-literal-deletion-wl* :: ‹*bool list* ⇒ *isasat* ⇒ (*64 word* × *isasat*) *nres*› **where**
‹*isa-pure-literal-deletion-wl occs* S_0 = (*do* {
ASSERT (*isa-pure-literal-deletion-wl-pre* S_0);
(-, *eliminated*, *S*) ← $WHILE_T^{\lambda(n,\text{-},S).\ \textit{get-vmtf-heur-array}\ S_0 = \textit{get-vmtf-heur-array}\ S}$ (λ(*n*, *x*). *n* ≠ *None*)
(λ(*n*, *eliminated*, *T*). *do* {
ASSERT (*n* ≠ *None*);
let A = *the n*;
ASSERT (*A* < *length* (*get-vmtf-heur-array* S_0));
ASSERT (*A* ≤ *unat32-max div 2*);
ASSERT (*get-vmtf-heur-array* S_0 = *get-vmtf-heur-array T*);
ASSERT (*nat-of-lit* (*Pos A*) < *length occs*);
ASSERT (*nat-of-lit* (*Neg A*) < *length occs*);
let L = (*if occs* ! *nat-of-lit* (*Pos A*) ∧ ¬ *occs* ! *nat-of-lit* (*Neg A*) *then Pos A else Neg A*);
ASSERT (*nat-of-lit* (− *L*) < *length occs*);
val ← *mop-polarity-pol* (*get-trail-wl-heur T*) *L*;
if ¬ *occs* ! *nat-of-lit* (− *L*) ∧ *val* = *None then do* {
S ← *isa-propagate-pure-bt-wl L T*;
ASSERT (*get-vmtf-heur-array* S_0 = *get-vmtf-heur-array S*);
RETURN (*get-next* (*get-vmtf-heur-array S* ! *A*),*eliminated* + *1*, *S*)
}
else RETURN (*get-next* (*get-vmtf-heur-array T* ! *A*),*eliminated*, *T*)
})
(*Some* (*get-vmtf-heur-fst* S_0), *0*, S_0);
RETURN (*eliminated*, *S*)
})›

end

theory *IsaSAT-Simplify-Pure-Literals*
imports *IsaSAT-Simplify-Pure-Literals-Defs*
Watched-Literals.Watched-Literals-Watch-List-Inprocessing
More-Refinement-Libs.WB-More-Refinement-Loops
IsaSAT-Restart
begin

lemma *isa-pure-literal-count-occs-clause-wl-pure-literal-count-occs-clause-wl*:
assumes ‹$(S, S') \in$ *twl-st-heur-restart-ana'* r u›
‹$(occs, occs') \in \langle$*bool-rel*$\rangle$*map-fun-rel* ($D_0$ (*all-init-atms-st* S'))›
‹$(C,C') \in$*nat-rel*›
shows ‹*isa-pure-literal-count-occs-clause-wl* C S *occs remaining* $\leq \Downarrow\{((occs, remaining), occs').$
$(occs, occs') \in \langle$*bool-rel*$\rangle$*map-fun-rel* ($D_0$ (*all-init-atms-st* S'))$\}$
(*pure-literal-count-occs-clause-wl* C' S' $occs'$)›
proof −
have *pure-literal-count-occs-clause-wl-alt-def*:
‹*pure-literal-count-occs-clause-wl* C S *occs* = *do* {
ASSERT (*pure-literal-count-occs-clause-wl-pre* C S *occs*);
let m = *length* (*get-clauses-wl* $S \propto C$);
(i, *occs*) ← $WHILE_T$*pure-literal-count-occs-clause-wl-invs* C S *occs* ($\lambda(i, occs).\ i < m$)
($\lambda(i, occs).$ *do* {
let L = *get-clauses-wl* $S \propto C$! i;
let occs = *occs* (L := *True*);
RETURN (i+*1*, *occs*)
})
(*0*, *occs*);
RETURN occs
}› **for** C S *occs*
by (*auto simp*: *pure-literal-count-occs-clause-wl-def*)
have [*refine0*]: ‹ ((*0*, *occs*, *remaining*), *0*, $occs'$) ∈ {((*n*, *occs*, *remaining*), n', $occs'$).
(n,n')∈ *nat-rel* ∧ ($occs$, $occs'$) ∈ $\langle$*bool-rel*$\rangle$*map-fun-rel* (D_0 (*all-init-atms-st* S'))}›
using *assms* **by** *auto*
show *?thesis*
supply *RETURN-as-SPEC-refine*[*refine2 del*]
unfolding *isa-pure-literal-count-occs-clause-wl-def pure-literal-count-occs-clause-wl-alt-def*
mop-arena-length-st-def mop-access-lit-in-clauses-heur-def
apply (*refine-vcg mop-arena-length*[*THEN fref-to-Down-curry*, *unfolded comp-def*,
of ‹*get-clauses-wl* S'› C' ‹*get-clauses-wl-heur* S› C ‹*set* (*get-vdom* S)›]
mop-arena-lit(*2*)[*THEN RETURN-as-SPEC-refine*, *of* - - ‹*set* (*get-vdom* S)›])
subgoal using *assms* **unfolding** *isa-pure-literal-count-occs-clause-wl-pre-def* **by** *fast*
subgoal
unfolding *pure-literal-count-occs-clause-wl-pre-def*
pure-literal-count-occs-l-clause-pre-def
by *normalize-goal+ auto*
subgoal using *assms* **by** (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)
subgoal using *assms* **unfolding** *isa-pure-literal-count-occs-clause-wl-invs-def* **by** *fast*
subgoal by *auto*
subgoal by *auto*
subgoal using *assms* **by** (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*)
subgoal
unfolding *pure-literal-count-occs-clause-wl-pre-def*
pure-literal-count-occs-l-clause-pre-def
by *normalize-goal+ auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal by *auto*

subgoal
unfolding *pure-literal-count-occs-clause-wl-pre-def*
pure-literal-count-occs-l-clause-pre-def
by *normalize-goal+*
(*auto simp add*: *map-fun-rel-def ran-m-def all-init-atms-alt-def*
$\mathcal{L}_{all}$*-all-init-atms*(*2*) *all-init-lits-of-wl-def all-lits-of-mm-add-mset*
dest!: *multi-member-split*
dest: *nth-mem-mset*[*THEN in-clause-in-all-lits-of-m*])
subgoal
unfolding *pure-literal-count-occs-clause-wl-pre-def*
pure-literal-count-occs-l-clause-pre-def
apply *normalize-goal+*
apply (*auto 4 3 simp add*: *map-fun-rel-def ran-m-def all-init-atms-alt-def*
$\mathcal{L}_{all}$*-all-init-atms*(*2*) *all-init-lits-of-wl-def all-lits-of-mm-add-mset*
dest!: *multi-member-split*
dest: *nth-mem-mset*[*THEN in-clause-in-all-lits-of-m*] *in-all-lits-of-mm-uminusD*)
by (*metis Un-iff all-lits-of-mm-add-mset in-all-lits-of-mm-uminusD in-clause-in-all-lits-of-m*
nth-mem-mset set-mset-union)
subgoal by (*auto simp add*: *map-fun-rel-def*)
subgoal by *auto*
done
qed

lemma *distinct-mset-add-subset-iff*: ‹*distinct-mset* (*A*+*B*) $\Longrightarrow$ *A* + *B* $\subseteq\#$ *C* $\longleftrightarrow$ *A* $\subseteq\#$ *C* $\wedge$ *B* $\subseteq\#$ *C*›
by (*induction A*)
(*auto simp add*: *insert-subset-eq-iff subset-remove1-mset-notin*)

lemma *isa-pure-literal-count-occs-wl-pure-literal-count-occs-wl*:
assumes ‹(*S*, *S'*) $\in$ *twl-st-heur-restart-ana'* *r* *u*›
shows ‹*isa-pure-literal-count-occs-wl S* $\leq$
$\Downarrow$ (*bool-rel* $\times_f$ ⟨*bool-rel*⟩*map-fun-rel* (*D*$_0$ (*all-init-atms-st S'*)))
(*pure-literal-count-occs-wl S'*)›
proof −
have *pure-literal-count-occs-wl-alt-def*:
‹*pure-literal-count-occs-wl S* = *do* {
ASSERT (*pure-literal-count-occs-wl-pre S*);
xs ← *SPEC* (λ*xs*. *distinct-mset xs* ∧ (∀ *C*∈#*dom-m* (*get-clauses-wl S*). *irred* (*get-clauses-wl S*) *C* ⟶ *C* ∈# *xs*));
abort ← *RES* (*UNIV* :: *bool set*);
let occs = (λ-. *False*);
(-, *occs*, *abort*) ← *WHILE*$_T$(λ(*A*, *occs*, *abort*). *A* ≠ {#} ∧ ¬*abort*)
(λ(*A*, *occs*, *abort*). *do* {
ASSERT (*A* ≠ {#});
C ← *SPEC* (λ*C*. *C* ∈# *A*);
b ← *RETURN* (*C* ∈# *dom-m* (*get-clauses-wl S*));
if (*b* ∧ *irred* (*get-clauses-wl S*) *C*) *then do* {
occs ← *pure-literal-count-occs-clause-wl C S occs*;
abort ← *RES* (*UNIV* :: *bool set*);
RETURN (*remove1-mset C A*, *occs*, *abort*)
} *else RETURN* (*remove1-mset C A*, *occs*, *abort*)
})
(*xs*, *occs*, *abort*);
RETURN (*abort*, *occs*)

```
  }› for S
  by (auto simp: pure-literal-count-occs-wl-def)

  have dist: ‹distinct-mset A ⟹ distinct-mset B ⟹ set-mset A ∩ set-mset B = {} ⟹ distinct-mset
(A + B)› for A B
    by (metis distinct-mset-add set-mset-eq-empty-iff set-mset-inter)

  have [refine0]: ‹pure-literal-count-occs-wl-pre S' ⟹
    RETURN (get-avdom S @ get-ivdom S)
    ≤ ⇓ (list-mset-rel)
    (SPEC
      (λxs. distinct-mset xs ∧
      (∀ C∈#dom-m (get-clauses-wl S').
    irred (get-clauses-wl S') C ⟶ C ∈# xs)))›
    apply (rule RETURN-SPEC-refine)
    apply (rule exI[of - ‹mset (get-avdom S @ get-ivdom S)›])
    using assms unfolding twl-st-heur-restart-def twl-st-heur-restart-ana-def in-pair-collect-simp
    apply −
    apply normalize-goal+
    by (auto simp: list-mset-rel-def br-def aivdom-inv-dec-alt-def
      dest: distinct-mset-mono
      intro!: dist)
  have conj-eqI: ‹a=a' ⟹ b=b' ⟹ (a&b) = (a'&b')› for a a' b b'
    by auto
  have [refine0]: ‹(get-avdom S @ get-ivdom S, xs) ∈ list-mset-rel ⟹
    (c ≤ 0, abort) ∈ bool-rel ⟹
    ((0, replicate (length (get-watched-wl-heur S)) False, c),
  xs, λ-. False, abort)
 ∈ {(i,xs). xs = mset (drop i (get-avdom S @ get-ivdom S))} ×r ⟨bool-rel⟩map-fun-rel (D0 (all-init-atms-st
S')) ×r {(a,b). b = (a ≤ 0)}› for xs abort c
    using assms unfolding twl-st-heur-restart-def twl-st-heur-restart-ana-def in-pair-collect-simp
    apply −
    apply normalize-goal+
    by (auto simp: list-mset-rel-def br-def map-fun-rel-def all-init-atms-alt-def)
  have [refine0]: ‹RETURN (0 ≥ a) ≤ ⇓ bool-rel (RES UNIV)› for a
    by auto
 have K: ‹(a',b)∈ nat-rel ⟹ (a, a'∈# dom-m (get-clauses-wl S')) ∈ A⟹ (a,b∈# dom-m (get-clauses-wl
S'))∈A› for a b f A a'
    by auto
  have aivdom: ‹aivdom-inv-dec (get-aivdom S) (dom-m (get-clauses-wl S'))› and
    valid: ‹valid-arena (get-clauses-wl-heur S) (get-clauses-wl S') (set (get-vdom S))›
    using assms
    by (auto simp: twl-st-heur-restart-def twl-st-heur-restart-ana-def
      aivdom-inv-dec-alt-def dest!: )
  have dist-vdom: ‹distinct (get-vdom S)› and
      valid: ‹valid-arena (get-clauses-wl-heur S) (get-clauses-wl S') (set (get-vdom S))›
   using assms by (auto simp: twl-st-heur-restart-def twl-st-heur-restart-ana-def aivdom-inv-dec-alt-def)


  have vdom-incl: ‹set (get-vdom S) ⊆ {MIN-HEADER-SIZE..< length (get-clauses-wl-heur S)}›
    using valid-arena-in-vdom-le-arena[OF valid] arena-dom-status-iff[OF valid] by auto

  have dist: ‹distinct-mset A ⟹ distinct-mset B ⟹ set-mset A ∩ set-mset B = {} ⟹ distinct-mset
(A + B)› for A B
    by (metis distinct-mset-add set-mset-eq-empty-iff set-mset-inter)
  then have dist2: ‹distinct-mset (mset (get-avdom S @ get-ivdom S))›
```

using *aivdom* **unfolding** *aivdom-inv-dec-alt-def*
by (*auto intro*!: *dist dest*: *distinct-mset-mono*)

have ‹*mset* (*get-avdom S* @ *get-ivdom S*) ⊆# *mset* (*get-vdom S*)›
using *dist2 aivdom* **unfolding** *aivdom-inv-dec-alt-def*
by (*auto simp*: *distinct-mset-add-subset-iff dist2*)
then have ‹*length* (*get-avdom S* @ *get-ivdom S*) ≤ *length* (*get-vdom S*)›
using *size-mset-mono* **by** *fastforce*
also have *le-vdom-arena*: ‹*length* (*get-vdom S*) ≤ *length* (*get-clauses-wl-heur S*) − *2*›
by (*subst distinct-card*[*OF dist-vdom*, *symmetric*])
(*use card-mono*[*OF - vdom-incl*] **in** *auto*)
finally have *le*: ‹*length* (*get-avdom S* @ *get-ivdom S*) ≤ *length* (*get-clauses-wl-heur S*) − *2*› **.**
show *?thesis*
unfolding *isa-pure-literal-count-occs-wl-def*
pure-literal-count-occs-wl-alt-def
apply (*rewrite at* ‹*let - = length - in -*› *Let-def*)
apply (*rewrite at* ‹*let - = - − - in -*› *Let-def*)
apply (*refine-vcg mop-arena-status2*[**where** *vdom*=‹*set*(*get-vdom S*)› **and** *N*=‹*get-clauses-wl S'*›]
isa-pure-literal-count-occs-clause-wl-pure-literal-count-occs-clause-wl)
subgoal by (*rule le*)
subgoal by (*auto simp*: *word-greater-zero-iff*)
subgoal by (*auto simp*: *access-avdom-at-pre-def*)
subgoal by (*auto simp*: *access-avdom-at-pre-def*)
subgoal by (*auto simp*: *access-ivdom-at-pre-def length-avdom-def*)
subgoal by (*auto 6 4 intro*: *in-set-dropI simp*: *nth-append*)
apply (*assumption*)
subgoal
using *aivdom*
apply (*simp add*: *aivdom-inv-dec-alt-def*)
by (*metis* (*no-types*, *lifting*) *Un-iff length-append mset-subset-eqD nth-mem-mset*
set-mset-mset set-union-code)
subgoal by (*rule valid*)
apply (*rule K*;*assumption*)
subgoal by *auto*
apply (*rule assms*)
subgoal by *simp*
subgoal by (*auto simp*: *drop-Suc-nth simp del*: *drop-append*)
subgoal by (*auto simp*: *drop-Suc-nth simp del*: *drop-append*)
subgoal by *auto*
done
qed

lemma *lookup-clause-rel-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *L* ∈ *lookup-clause-rel* $\mathcal{A}$ ⟹ *L* ∈ *lookup-clause-rel* $\mathcal{B}$›
using $\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$] *atms-of-*$\mathcal{L}_{all}$*-cong*[*of* $\mathcal{A}$ $\mathcal{B}$]
unfolding *lookup-clause-rel-def*
by (*cases L*) (*auto*)

lemma *isa-propagate-pure-bt-wl-propagate-pure-bt-wl*:
assumes *S*$_0$ *T*: ‹(*S*$_0$, *T*) ∈ *twl-st-heur-restart-ana'* *r u*› **and**
‹(*L*,*L'*)∈*Id*›
shows ‹*isa-propagate-pure-bt-wl L S*$_0$ ≤⇓{(*U*, *V*). (*U*,*V*)∈*twl-st-heur-restart-ana'* *r u* ∧ *get-vmtf-heur U* = *get-vmtf-heur S*$_0$} (*propagate-pure-bt-wl L' T*)›
proof −

have *propagate-pure-bt-wl-alt-def*:
‹*propagate-pure-bt-wl* = (λ*L* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *WS*). *do* {
ASSERT(*propagate-pure-wl-pre L* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *WS*));
M ← *cons-trail-propagate-l L 0 M*;
RETURN (*M*, *N*, *D*, *NE*, *UE*, *add-mset* {#*L*#} *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *add-mset* (−*L*) *Q*, *WS*)})›
unfolding *propagate-pure-bt-wl-def* **by** *auto*
have [*refine*]: ‹*S*=*S*$_0$ ⟹ *M* = *get-trail-wl T* ⟹
mop-isa-length-trail (*get-trail-wl-heur S*) ≤ *SPEC* (λ*ca*. (*ca*, *length M*) ∈ *Id*)› **for** *S M*
apply (*subst twl-st-heur-restart-isa-length-trail-get-trail-wl*[*of* - *T r*])
using *S*$_0$ *T* **apply** *simp*
using *S*$_0$ *T* **apply** (*simp-all add*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*
all-init-atms-st-alt-def all-init-atms-alt-def)
done
have *trail*: ‹(*get-trail-wl-heur S*$_0$, *get-trail-wl T*) ∈ *trail-pol* (*all-init-atms-st T*)› **for** *x2a x2*
using *assms* **by** (*auto simp*: *twl-st-heur-restart-def twl-st-heur-restart-ana-def*
all-init-atms-alt-def)
have *H*: ‹*A* = *B* ⟹ *x* ∈ *A* ⟹ *x* ∈ *B*› **for** *A B x*
by *auto*
have *H'*: ‹*A* = *B* ⟹ *A x* ⟹ *B x*› **for** *A B x*
by *auto*

obtain *M N D NE UE NEk UEk NS US N0 U0 Q WS* **where**
T: ‹*T* = (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *WS*)›
by (*cases T*)
{
assume *pre*: ‹*propagate-pure-wl-pre L T*›
note *cong* = *trail-pol-cong heuristic-rel-cong*
option-lookup-clause-rel-cong
vdom-m-cong[*symmetric*, *THEN H*] *isasat-input-nempty-cong*[*THEN iffD1*]
isasat-input-bounded-cong[*THEN iffD1*]
cach-refinement-empty-cong[*THEN H'*]
phase-saving-cong[*THEN H'*]
$\mathcal{L}_{all}$-*cong*[*THEN H*]
D_0-*cong*[*THEN H*]
lookup-clause-rel-cong
map-fun-rel-D_0-*cong*
isa-vmtf-cong[*THEN isa-vmtf-consD*]
isasat-input-nempty-cong[*THEN iffD1*]
isasat-input-bounded-cong[*THEN iffD1*]
note *subst* = *empty-occs-list-cong*

let *?T* = ‹(*Propagated L 0* # *M*, *N*, *D*, *NE*, *UE*, *add-mset* {#*L'*#} *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *add-mset* (− *L'*) *Q*, *WS*)›
have ‹*L'* ∈# *all-lits-of-mm* ({#*mset* (*fst x*). *x* ∈# *init-clss-l N*#} + (*NE* + *NEk*) + *NS* + *N0*)› **and**
D: ‹*D* = *None*› **and**
undef: ‹*undefined-lit M L'*›
using *pre assms*(*2*) **unfolding** *propagate-pure-wl-pre-def propagate-pure-l-pre-def* **apply** −
by (*solves* ‹*normalize-goal*+; *auto simp*: *T twl-st-l-def all-init-lits-of-wl-def*›)+
then have *H*: ‹*set-mset* (*all-init-lits-of-wl ?T*) = *set-mset* (*all-init-lits-of-wl T*)›
unfolding *T* **by** (*auto simp*: *all-init-lits-of-wl-def*
all-lits-of-mm-add-mset all-lits-of-m-add-mset in-all-lits-of-mm-uminus-iff)
then have *H*: ‹*set-mset* (*all-init-atms-st ?T*) = *set-mset* (*all-init-atms-st T*)›
unfolding *all-init-atms-st-alt-def* **by** *auto*
note - = *D undef subst*[*OF H*] *cong*[*OF H*[*symmetric*]]

} **note** *cong* = *this*(*4*−) **and** *subst* = *this*(*3*) **and** *D* = *this*(*1*) **and** *undef* = *this*(*2*)

show *?thesis*
supply [*simp del*] = *isasat-input-nempty-def isasat-input-bounded-def*
unfolding *propagate-pure-bt-wl-alt-def isa-propagate-pure-bt-wl-def T*
apply (*rewrite at* ‹*let* - = *get-trail-wl-heur* - *in* -› *Let-def*)
apply (*rewrite at* ‹*let* - = *get-stats-heur* - *in* -› *Let-def*)
apply (*cases* S_0)
apply (*hypsubst*, *unfold prod.simps*)
apply (*refine-vcg*
cons-trail-Propagated-tr2[*of* - - - - - - ‹*all-init-atms-st T*›])
subgoal by (*auto simp*: *DECISION-REASON-def*)
subgoal by (*rule cons-trail-Propagated-tr-pre*[*of* - ‹*get-trail-wl T*› ‹*all-init-atms-st T*›])
(*use* ‹(*L*,*L*′)∈*Id*› *trail* **in** ‹*auto simp*: *propagate-pure-wl-pre-def propagate-pure-l-pre-def*
state-wl-l-def twl-st-l-def $\mathcal{L}_{all}$*-all-init-atms all-init-lits-of-wl-def all-init-lits-of-l-def*
get-init-clss-l-def T›)
subgoal by (*use trail assms* **in** ‹*auto simp*: *T*›)
subgoal by (*use* ‹(*L*,*L*′)∈*Id*› *trail* **in** ‹*auto simp*: *propagate-pure-wl-pre-def propagate-pure-l-pre-def*
state-wl-l-def twl-st-l-def $\mathcal{L}_{all}$*-all-init-atms all-init-lits-of-wl-def all-init-lits-of-l-def*
get-init-clss-l-def T›)
subgoal using *assms D undef* **unfolding** *twl-st-heur-restart-ana-def twl-st-heur-restart-def*
$\mathcal{L}_{all}$*-all-init-atms all-init-atms-st-alt-def all-init-atms-alt-def T subst*
unfolding *all-init-atms-st-alt-def*[*symmetric*] **apply** −
apply (*simp only*: *in-pair-collect-simp*)
apply (*intro conjI*)
subgoal
apply *simp*
using *cong T* **apply** *fast*
done
subgoal by *simp*
subgoal
apply *simp*
using *cong T* **apply** *fast*
done
subgoal
apply *auto*
done
subgoal
apply *simp*
done
subgoal
apply *simp*
using *cong*(*10*−) *T* **apply** *fast*
done
subgoal apply *simp* **using** *cong*(*10*−) *T* **by** *fast*
subgoal by *simp*
subgoal by *simp*
subgoal apply *simp* **using** *cong T* **by** *fast*
subgoal by *simp*
subgoal by *simp*
subgoal apply *simp* **using** *cong*(*4*) *T* **by** *fast*
subgoal by *simp*
subgoal apply *simp* **using** *cong T* **by** *fast*
subgoal by *simp* (*use cong T* **in** *fast*)
subgoal by *simp*
subgoal by *simp* (*use cong T* **in** *fast*)

```
    subgoal using subst by (simp add: T all-init-atms-st-def)
    subgoal by (simp add: learned-clss-count-def)
    subgoal by (simp add: learned-clss-count-def)
    subgoal by (simp add: subst)
    done
  done
qed


lemma isa-pure-literal-deletion-wl-pure-literal-deletion-wl:
 assumes S_0 T: ‹(S_0, T) ∈ twl-st-heur-restart-ana' r u› and
   occs: ‹(occs, occs') ∈ ⟨bool-rel⟩map-fun-rel (D_0 (all-init-atms-st T))›
 shows ‹isa-pure-literal-deletion-wl occs S_0 ≤⇓{((-, U), V). (U,V)∈twl-st-heur-restart-ana' r u} (pure-literal-deletion-wl
occs' T)›
proof −
 have B: ‹⇓Id(pure-literal-deletion-wl occs' T) ≥
 (do {
   ASSERT (pure-literal-deletion-wl-pre T);
   iterate-over-\mathcal{L}_{all}
     (λA (T). do {
       ASSERT (A ∈# all-init-atms-st T);
       let L = (if occs' (Pos A) ∧ ¬ occs' (Neg A)
             then Pos A else Neg A);
       let val = polarity (get-trail-wl T) L;
       if ¬occs' (−L) ∧ val = None
       then do {S ← propagate-pure-bt-wl L T;
        RETURN (S)}
       else RETURN (T)
         })
         (atm-of `# all-init-lits-of-wl T)
   (λxs S. pure-literal-deletion-wl-inv T (atm-of `# all-init-lits-of-wl T) (S, xs))
   (T)})› (is ‹- ≥ ?B› is ‹- ≥ do { ASSERT -; iterate-over-\mathcal{L}_{all} ?f - - -}›)
 proof −
   have [refine0]: ‹(x, xs) ∈ Id ⟹ set-mset x = set-mset (atm-of `# all-init-lits-of-wl T) ⟹
    ((x, T), T, xs) ∈ {((a,b), (c,d)). (a,d) ∈Id ∧ (b,c)∈Id}› for x xs
    by (auto simp: all-init-atms-st-alt-def)
   have K: ‹a=b ⟹ a ≤⇓Id b› for a b
    by auto
   have Lin: ‹pure-literal-deletion-wl-inv T (atm-of `# all-init-lits-of-wl T) (x1, x2) ⟹
     L ∈# x2 ⟹ L ∈# all-init-atms-st x1› for L x1 x2
    unfolding pure-literal-deletion-wl-inv-def prod.simps pure-literal-deletion-l-inv-def
    apply normalize-goal+
    apply (simp add: all-init-atms-st-alt-def)
    by (metis (no-types, lifting) all-init-lits-of-l-all-init-lits-of-wl mset-subset-eqD
     multiset.set-map rtranclp-cdcl-twl-pure-remove-l-same(7))

   show ?thesis
    unfolding iterate-over-VMTF-def pure-literal-deletion-wl-def
     pure-literal-deletion-candidates-wl-def iterate-over-\mathcal{L}_{all}-def
     iterate-over-\mathcal{L}_{all}C-def nres-monad3 nres-bind-let-law If-bind-distrib
    apply (rewrite at ‹let - = ⋃ - in -› Let-def)
    apply refine-vcg
    subgoal by (auto simp: all-init-atms-st-alt-def)
    subgoal by fast
    subgoal for \mathcal{A} xs x x' x1 x2
     unfolding all-init-atms-st-alt-def pure-literal-deletion-l-inv-def
```

pure-literal-deletion-wl-inv-def case-prod-beta
apply *normalize-goal+*
apply (*rename-tac ya yb yc*, *rule-tac x=ya* **in** *exI*[*of*])
apply (*rule-tac x=yb* **in** *exI*[*of*])
apply (*intro conjI*)
apply *simp*
apply *simp*
apply (*rule-tac x=yc* **in** *exI*[*of*])
apply *simp-all*
by (*metis distinct-subseteq-iff set-image-mset subset-mset.dual-order.trans subset-mset.order-refl*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*simp add*: *Lin*)
subgoal by (*auto simp*: *polarity-def*)
apply (*rule K*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

have *vmtf*: ‹*get-vmtf-heur* S_0 ∈ *bump-heur* (*all-init-atms-st T*) (*get-trail-wl T*)› **and**
nempty: ‹*isasat-input-nempty* (*all-init-atms-st T*)› **and**
bounded: ‹*isasat-input-bounded* (*all-init-atms-st T*)›
using *assms* **unfolding** *twl-st-heur-restart-ana-def twl-st-heur-restart-def*
by (*simp-all add*: *all-init-atms-alt-def del*: *isasat-input-nempty-def*)
let *?vm* = ‹*get-vmtf-heur* S_0›
obtain *M* **where** *vmtf'*: ‹(*get-vmtf-heur-array* S_0, *fst* (*snd* (*bump-get-heuristics ?vm*)),
get-vmtf-heur-fst S_0, *fst* (*snd* (*snd* (*snd* (*bump-get-heuristics ?vm*)))), *snd* (*snd* (*snd* (*snd* (*bump-get-heuristics ?vm*))))) ∈ *vmtf* (*all-init-atms-st T*) *M*› **and**
M: ‹*M* = (*get-trail-wl T*) ∨ *M* = (*get-unit-trail* (*get-trail-wl T*))›
using *vmtf* **unfolding** *bump-heur-def get-vmtf-heur-array-def bump-get-heuristics-def get-vmtf-heur-fst-def*
by (*cases* ‹*bump-get-heuristics ns*›) (*auto simp*: *bump-get-heuristics-def bumped-vmtf-array-fst-def split*: *if-splits*)
have *C*: ‹⇓*Id ?B* ≥ (*do* {
ASSERT (*pure-literal-deletion-wl-pre T*);
(*S*) ← *iterate-over-VMTF ?f*
(λ*S*. ∃ *xs*. *pure-literal-deletion-wl-inv T* (*atm-of* ‘# *all-init-lits-of-wl T*) (*S*, *xs*))
(*get-vmtf-heur-array* S_0, *Some* (*get-vmtf-heur-fst* S_0)) (*T*);
RETURN (*S*)
})› (**is** ‹- ≥ *?C*›)
apply (*refine-vcg iterate-over-VMTF-iterate-over-*$\mathcal{L}_{all}$)
unfolding *nres-monad2 all-init-atms-st-alt-def*[*symmetric*]
apply (*rule iterate-over-VMTF-iterate-over-*$\mathcal{L}_{all}$[*OF vmtf'*])
subgoal using *nempty* **by** *auto*
subgoal using *bounded* **by** *auto*
subgoal by *blast*
done

have *D*: ‹⇓(*twl-st-heur-restart-ana'* *r u*) *?C* ≥
(*do* {
ASSERT (*pure-literal-deletion-wl-pre T*);
S ← *iterate-over-VMTF*
(λ*A* (*T*). *do* {

```
        ASSERT (nat-of-lit (Pos A) < length occs);
        ASSERT (nat-of-lit (Neg A) < length occs);
        ASSERT (occs ! (nat-of-lit(Pos A)) = occs' (Pos A));
        ASSERT (occs ! (nat-of-lit(Neg A)) = occs' (Neg A));
        let L = (if occs ! (nat-of-lit(Pos A)) ∧ ¬ occs ! (nat-of-lit (Neg A))
              then Pos A else Neg A);
        val ← mop-polarity-pol (get-trail-wl-heur T) L;
        ASSERT (nat-of-lit (−L) < length occs);
        if ¬occs ! nat-of-lit (−L) ∧ val = None
        then do {S ← isa-propagate-pure-bt-wl L T;
          ASSERT (get-vmtf-heur-array S₀ = get-vmtf-heur-array S);
         RETURN (S)}
       else RETURN (T)
         })
     (λS. get-vmtf-heur-array S₀ = get-vmtf-heur-array S)
    (get-vmtf-heur-array S₀, Some (get-vmtf-heur-fst S₀))
    (S₀);
   RETURN S})› (is ‹- ≥ ?D›)
  proof −
    have [refine]: ‹ ((Some (get-vmtf-heur-fst S₀), S₀), Some (get-vmtf-heur-fst S₀), T)
    ∈ Id ×r {(U, V). (U, V)∈twl-st-heur-restart-ana' r u ∧ get-vmtf-heur-array S₀ = get-vmtf-heur-array
U}›
      using assms by auto
    have H: ‹P ∈# \mathcal{L}_{all} (atm-of `# all-init-lits-of-wl A) ⟷ atm-of P ∈# all-init-atms-st A› for P A
      using \mathcal{L}_{all}-all-init-atms(2) in-\mathcal{L}_{all}-atm-of-\mathcal{A}_{in} by (metis all-init-atms-st-alt-def)
    have K: ‹a=b ⟹(a,b)∈Id› for a b
      by auto
    have [simp, intro!]: ‹(x2a, x2) ∈ twl-st-heur-restart-ana r ⟹
   (get-trail-wl-heur x2a, get-trail-wl x2) ∈ trail-pol (atm-of `# all-init-lits-of-wl x2)› for x2a x2
      by (auto simp: twl-st-heur-restart-ana-def twl-st-heur-restart-def
        all-init-atms-alt-def all-init-atms-st-alt-def)
    have occs-st: ‹occs ! (nat-of-lit (Pos (x1))) = occs' (Pos (x1))›
      ‹occs ! (nat-of-lit (Neg (x1))) = occs' (Neg (x1))›
      ‹nat-of-lit (Pos x1) < length occs›
      ‹nat-of-lit (Neg x1) < length occs›
      if inv: ‹∃xs. pure-literal-deletion-wl-inv T (atm-of `# all-init-lits-of-wl T) (x2, xs)›
        ‹x1 ∈# all-init-atms-st x2› for x1 x2
    proof −
      obtain xs where inv: ‹pure-literal-deletion-wl-inv T (atm-of `# all-init-lits-of-wl T) (x2, xs)›
        using inv by auto
      have ‹x1 ∈# all-init-atms-st T›
        using inv unfolding pure-literal-deletion-wl-inv-def prod.simps
          pure-literal-deletion-l-inv-def apply −
        by normalize-goal+
          (metis (no-types, opaque-lifting) all-init-atms-st-alt-def
        all-init-lits-of-l-all-init-lits-of-wl rtranclp-cdcl-twl-pure-remove-l-same(7) set-image-mset that(2))
      then show ‹occs ! (nat-of-lit (Pos (x1))) = occs' (Pos (x1))›
        ‹occs ! (nat-of-lit (Neg (x1))) = occs' (Neg (x1))›
      ‹nat-of-lit (Pos x1) < length occs›
      ‹nat-of-lit (Neg x1) < length occs›
        using occs by (auto simp: map-fun-rel-def \mathcal{L}_{all}-add-mset dest!: multi-member-split)
    qed
    show ?thesis
      unfolding iterate-over-VMTF-def nres-monad3 nres-bind-let-law prod.simps If-bind-distrib
        mop-polarity-pol-def nres-monad2 nres-monad1[of - ‹λx. RETURN (-,x)›]
      apply (refine-vcg isa-propagate-pure-bt-wl-propagate-pure-bt-wl[where r=r and u=u])
```

subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal for *x x′ x1 x2 x1a x2a*
unfolding *case-prod-beta*
by (*rule occs-st*) (*assumption*, *simp*)
subgoal for *x x′ x1 x2 x1a x2a*
unfolding *case-prod-beta*
by (*rule occs-st*) (*assumption*, *simp*)
subgoal for *x x′ x1 x2 x1a x2a*
unfolding *case-prod-beta*
by (*rule occs-st*) (*assumption*, *simp*)
subgoal for *x x′ x1 x2 x1a x2a*
unfolding *case-prod-beta*
by (*rule occs-st*) (*assumption*, *simp*)
subgoal for *x x′ x1 x2 x1a x2a*
by (*rule polarity-pol-pre*[*of* - - ‹(*all-init-atms-st x2*)›])
(*auto simp add*: *H* $\mathcal{L}_{all}$*-all-init-atms all-init-atms-st-alt-def all-init-atms-alt-def*)
apply (*rule K*)
subgoal for *x x′ x1 x2 x1a x2a*
using *assms*
by (*auto intro*!: *polarity-pol-polarity*[*of* ‹(*all-init-atms-st x2*)›, *unfolded option-rel-id-simp*, *THEN fref-to-Down-unRET-uncurry-Id*]
simp: *H all-init-atms-st-alt-def*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *get-vmtf-heur-array-def*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

have *E*: ‹⇓{((-, *U*), *V*). (*U*,*V*)∈*Id*} *?D* ≥ *isa-pure-literal-deletion-wl-raw occs* S_0› (**is** ‹- ≥ *?E*›)
proof −
have *K*: ‹*a*=*b* ⟹ *a*≤⇓*Id b*› **for** *a b*
by *auto*
have *H1*: ‹(*S*, *x′a*) ∈ *Id* ⟹ ((*x1b* + *1*, *S*), *x′a*) ∈ {((*a*,*b*),*c*). (*b*,*c*)∈*Id*}› **for** *S x′a x1b*
by *auto*
have *H2*: ‹⋀*x x′ x1 x2 x1a x2a*.
pure-literal-deletion-wl-pre T ⟹
isa-pure-literal-deletion-wl-pre S_0 ⟹
(*x*, *x′*) ∈ {((*a*, *b*, *c*), *x*, *y*). (*a*, *x*) ∈ *Id* ∧ (*c*, *y*) ∈ *Id*} ⟹ *x′* = (*x1*, *x2*) ⟹ *x* = (*x1a*, *x2a*) ⟹ (*x2a*, *x2*) ∈ {((*a*,*b*),*c*). (*b*,*c*)∈*Id*}›
by *auto*
have [*refine*]: ‹((*Some* (*get-vmtf-heur-fst* S_0), *0*, S_0), *Some* (*get-vmtf-heur-fst* S_0), S_0) ∈
{((*a*,*b*,*c*), (*x*,*y*)). (*a*,*x*)∈ *Id* ∧ (*c*,*y*)∈*Id*}› **by** *auto*
show *?thesis*
unfolding *isa-pure-literal-deletion-wl-raw-def iterate-over-VMTF-def prod.simps Let-def*
apply *refine-vcg*
subgoal using *assms* **unfolding** *isa-pure-literal-deletion-wl-pre-def* **by** *fast*
subgoal by *auto*

subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
apply (*rule K*)
subgoal by *auto*
subgoal by *auto*
apply (*rule K*)
subgoal by *auto*
subgoal by *auto*
apply (*rule H1*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
apply (*rule H2*; *assumption*)
subgoal by *auto*
done
qed
have *F*: ‹⇓*Id ?E* ≥ *isa-pure-literal-deletion-wl occs* S_0›
proof −
have [*refine*]: ‹((*Some* (*get-vmtf-heur-fst* S_0), *0*, S_0), *snd* (*get-vmtf-heur-array* S_0, *Some* (*get-vmtf-heur-fst* S_0)), *0*, S_0) ∈ *Id* ×$_r$ *Id* ×$_r$ *Id*›
by *auto*
have *K*: ‹*a*=*b* ⟹ *a*≤⇓*Id b*› **for** *a b*
by *auto*
show *?thesis*
unfolding *isa-pure-literal-deletion-wl-def isa-pure-literal-deletion-wl-raw-def iterate-over-VMTF-def nres-monad3 nres-monad2 If-bind-distrib case-prod-beta nres-bind-let-law*
apply *refine-vcg*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
apply (*rule K*)
subgoal by *auto*
subgoal by *auto*
apply (*rule K*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed
show *?thesis*
apply (*rule order-trans*[*OF F*])
apply (*rule order-trans*)
apply (*rule ref-two-step'*[*OF E*])

apply (*subst conc-fun-chain*)
apply (*rule order-trans*)
apply (*rule ref-two-step'*[*OF D*])
apply (*subst conc-fun-chain*)
apply (*rule order-trans*)
apply (*rule ref-two-step'*[*OF C*])
apply (*subst conc-fun-chain*)
apply (*rule order-trans*)
apply (*rule ref-two-step'*[*OF B*])
apply (*rule ref-two-step''*)
by *auto*
qed

end
theory *IsaSAT-Simplify-Pure-Literals-LLVM*
imports *IsaSAT-Restart-Defs*
IsaSAT-Simplify-Pure-Literals-Defs
IsaSAT-Setup-LLVM IsaSAT-Trail-LLVM
IsaSAT-Proofs-LLVM
begin

sepref-register *mop-arena-status-st isa-pure-literal-count-occs-clause-wl*
sepref-def *isa-pure-literal-count-occs-clause-wl-code*
is ‹*uncurry3 isa-pure-literal-count-occs-clause-wl*›
:: ‹[λ(((*C*, *S*), -), -). *length* (*get-clauses-wl-heur S*) ≤ *snat64-max* ∧ *learned-clss-count S* ≤ *unat64-max*]$_a$
sint64-nat-assn^k *$*_a$ *isasat-bounded-assn*^{k} *$*_a$ (*larray-assn'* *TYPE*(*64*) *bool1-assn*)^d *$*_a$ *word64-assn*^{d}
→ *larray-assn'* *TYPE*(*64*) *bool1-assn* ×$_a$ *word64-assn*›
unfolding *isa-pure-literal-count-occs-clause-wl-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-register *isa-pure-literal-count-occs-wl*
sepref-def *isa-pure-literal-count-occs-wl-code*
is *isa-pure-literal-count-occs-wl*
:: ‹[λ*S*. *length* (*get-clauses-wl-heur S*) ≤ *snat64-max* ∧ *learned-clss-count S* ≤ *unat64-max*]$_a$
isasat-bounded-assn^k → *bool1-assn* ×$_a$ *larray-assn'* *TYPE*(*64*) *bool1-assn*›
unfolding *isa-pure-literal-count-occs-wl-def nth-append Let-def*
larray-fold-custom-replicate mop-arena-status-st-def[*symmetric*]
access-ivdom-at-def[*symmetric*] *length-avdom-def*[*symmetric*] *length-ivdom-def*[*symmetric*]
access-avdom-at-def[*symmetric*] *length-watchlist-raw-def*[*symmetric*] *length-append*
supply *of-nat-snat*[*sepref-import-param*]
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *isa-propagate-pure-bt-wl-alt-def*:
‹*isa-propagate-pure-bt-wl* = (λ*L S*. *do* {
let (*M*, *S*) = *extract-trail-wl-heur S*;
let (*stats*, *S*) = *extract-stats-wl-heur S*;
ASSERT(*0* ≠ *DECISION-REASON*);
ASSERT(*cons-trail-Propagated-tr-pre* ((*L*, *0*::*nat*), *M*));
M ← *cons-trail-Propagated-tr* (*L*) *0 M*;
let stats = *incr-units-since-last-GC* (*incr-uset* (*incr-purelit-elim stats*));
let S = *update-stats-wl-heur stats S*;
let S = *update-trail-wl-heur M S*;
let - = *log-unit-clause L*;
RETURN S})›

unfolding *isa-propagate-pure-bt-wl-def log-unit-clause-def*
by (*auto simp*: *empty-US-heur-def state-extractors Let-def intro*!: *ext split*: *isasat-int-splits*)

sepref-register *isa-propagate-pure-bt-wl cons-trail-Propagated-tr*
sepref-def *isa-propagate-pure-bt-wl-code*
is ‹*uncurry isa-propagate-pure-bt-wl*›
:: ‹*unat-lit-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
unfolding *isa-propagate-pure-bt-wl-alt-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *isa-pure-literal-deletion-wl-alt-def*:
‹*isa-pure-literal-deletion-wl occs* S_0 = (*do* {
ASSERT (*isa-pure-literal-deletion-wl-pre* S_0);
(-, *eliminated*, *S*) ← *WHILE*$_T$$^{\lambda(n,\text{-},S).\ \textit{get-vmtf-heur-array}\ S_0 = \textit{get-vmtf-heur-array}\ S}$ ($\lambda(n,\ x).\ n \neq$ *None*)
(λ(*n*, *eliminated*, *T*). *do* {
ASSERT (*n* ≠ *None*);
let A = *the n*;
ASSERT (*A* < *length* (*get-vmtf-heur-array* S_0));
ASSERT (*A* ≤ *unat32-max div 2*);
ASSERT (*get-vmtf-heur-array* S_0 = *get-vmtf-heur-array T*);
ASSERT (*nat-of-lit* (*Pos A*) < *length occs*);
ASSERT (*nat-of-lit* (*Neg A*) < *length occs*);
let L = (*if occs* ! *nat-of-lit* (*Pos A*) ∧ ¬ *occs* ! *nat-of-lit* (*Neg A*) *then Pos A else Neg A*);
ASSERT (*nat-of-lit* (− *L*) < *length occs*);
val ← *mop-polarity-pol* (*get-trail-wl-heur T*) *L*;
if ¬ *occs* ! *nat-of-lit* (− *L*) ∧ *val* = *None then do* {
S ← *isa-propagate-pure-bt-wl L T*;
ASSERT (*get-vmtf-heur-array* S_0 = *get-vmtf-heur-array S*);
RETURN (*get-next* (*get-vmtf-heur-array S* ! *A*),*eliminated* + *1*, *S*)
}
else RETURN (*get-next* (*get-vmtf-heur-array T* ! *A*),*eliminated*, *T*)
})
(*Some* (*get-vmtf-heur-fst* S_0), *0*, S_0);
mop-free occs;
RETURN (*eliminated*, *S*)
})›
unfolding *isa-pure-literal-deletion-wl-def mop-free-def*
by *auto*

sepref-def *isa-pure-literal-deletion-wl-code*
is ‹*uncurry isa-pure-literal-deletion-wl*›
:: ‹[λ(-, *S*). *length* (*get-clauses-wl-heur S*) ≤ *snat64-max* ∧ *learned-clss-count S* ≤ *unat64-max*]$_a$
(*larray-assn*$'$ *TYPE*(*64*) *bool1-assn*)^d $*_a$ *isasat-bounded-assn*^{d} → *word64-assn* $\times_a$ *isasat-bounded-assn*›
unfolding *isa-pure-literal-deletion-wl-alt-def nres-monad3 nres-monad2*
get-bump-heur-array-nth-def[*symmetric*] *UNSET-def*[*symmetric*] *atom.fold-option*
mop-polarity-st-heur-def[*symmetric*] *tri-bool-eq-def*[*symmetric*]
get-bump-heur-array-nth-def[*symmetric*] *prod.simps*
get-vmtf-heur-array-def[*symmetric*]
by *sepref*

end
theory *IsaSAT-Simplify-Forward-Subsumption-Defs*
imports *IsaSAT-Setup*
IsaSAT-Occurence-List

IsaSAT-Restart
IsaSAT-LBD
begin

22.2 Forward subsumption

22.2.1 Algorithm

We first refine the algorithm to use occurence lists, while keeping as many things as possible abstract (like the candidate selection or the selection of the literal with the least number of occurrences). We also include the marking structure (at least abstractly, because why not)

For simplicity, we keep the occurrence list outside of the state (unlike the current solver where this is part of the state.)

definition *valid-occs* **where** ‹*valid-occs occs vdom* ⟷ *cocc-content-set occs* ⊆ *set* (*get-vdom-aivdom vdom*) ∧
distinct-mset (*cocc-content occs*)›

This version is equivalent to *twl-st-heur-restart*, without any information on the occurrence list.

definition *twl-st-heur-restart-occs* :: ‹(*isasat* × *nat twl-st-wl*) *set*› **where**
[*unfolded Let-def*]: ‹*twl-st-heur-restart-occs* =
{(*S*,*T*).
let M′ = *get-trail-wl-heur S*; *N′* = *get-clauses-wl-heur S*; *D′* = *get-conflict-wl-heur S*;
W′ = *get-watched-wl-heur S*; *j* = *literals-to-update-wl-heur S*; *outl* = *get-outlearned-heur S*;
cach = *get-conflict-cach S*; *clvls* = *get-count-max-lvls-heur S*;
vm = *get-vmtf-heur S*;
vdom = *get-aivdom S*; *heur* = *get-heur S*; *old-arena* = *get-old-arena S*;
lcount = *get-learned-count S*; *occs* = *get-occs S in*
let M = *get-trail-wl T*; *N* = *get-clauses-wl T*; *D* = *get-conflict-wl T*;
Q = *literals-to-update-wl T*;
W = *get-watched-wl T*; *N0* = *get-init-clauses0-wl T*; *U0* = *get-learned-clauses0-wl T*;
NS = *get-subsumed-init-clauses-wl T*; *US* = *get-subsumed-learned-clauses-wl T*;
NEk = *get-kept-unit-init-clss-wl T*; *UEk* = *get-kept-unit-learned-clss-wl T*;
NE = *get-unkept-unit-init-clss-wl T*; *UE* = *get-unkept-unit-learned-clss-wl T in*
(*M′*, *M*) ∈ *trail-pol* (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) ∧
valid-arena N′ N (*set* (*get-vdom-aivdom vdom*)) ∧
(*D′*, *D*) ∈ *option-lookup-clause-rel* (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) ∧
(*D* = *None* ⟶ *j* ≤ *length M*) ∧
Q = *uminus* ‘# *lit-of* ‘# *mset* (*drop j* (*rev M*)) ∧
(*W′*, *W*) ∈ ⟨*Id*⟩*map-fun-rel* (*D*$_0$ (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*))) ∧
vm ∈ *bump-heur* (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) *M* ∧
no-dup M ∧
clvls ∈ *counts-maximum-level M D* ∧
cach-refinement-empty (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) *cach* ∧
out-learned M D outl ∧
clss-size-corr-restart N NE {#} *NEk UEk NS* {#} *N0* {#} *lcount* ∧
vdom-m (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) *W N* ⊆ *set* (*get-vdom-aivdom vdom*) ∧
aivdom-inv-dec vdom (*dom-m N*) ∧
isasat-input-bounded (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) ∧
isasat-input-nempty (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) ∧
old-arena = [] ∧
heuristic-rel (*all-init-atms N* (*NE*+*NEk*+*NS*+*N0*)) *heur* ∧
valid-occs occs vdom
}›

abbreviation *twl-st-heur-restart-occs'* :: ‹-› **where**
‹*twl-st-heur-restart-occs' r u* ≡
{(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-restart-occs* ∧ *length* (*get-clauses-wl-heur S*) = *r* ∧ *learned-clss-count S* ≤ *u*}›

definition *subsume-clauses-match2-pre* :: ‹*nat* ⇒ *nat* ⇒ - ⇒ *clause-hash* ⇒ *bool*› **where**
‹*subsume-clauses-match2-pre C C' N D* ⟷
subsume-clauses-match-pre C C' (*get-clauses-wl N*) ∧
snd D = *mset* (*get-clauses-wl N* ∝ *C'*)›

definition *subsume-clauses-match2* :: ‹*nat* ⇒ *nat* ⇒ - ⇒ *clause-hash* ⇒ *nat subsumption nres*› **where**
‹*subsume-clauses-match2 C C' N D* = *do* {
ASSERT (*subsume-clauses-match2-pre C C' N D*);
let n = *length* (*get-clauses-wl N* ∝ *C*);
(*i*, *st*) ← *WHILE*$_T$ λ(*i*,*s*). *try-to-subsume C' C* ((*get-clauses-wl N*)(*C* ↪ *take i* (*get-clauses-wl N* ∝ *C*))) *s* (λ(*i*, *st*). *i* < *n*∧ *st* ≠ *NONE*)
(λ(*i*, *st*). *do* {
L ← *mop-clauses-at* (*get-clauses-wl N*) *C i*;
lin ← *mop-ch-in L D*;
if lin
then RETURN (*i*+*1*, *st*)
else do {
lin ← *mop-ch-in* (−*L*) *D*;
if lin
then if is-subsumed st
then RETURN (*i*+*1*, *STRENGTHENED-BY L C*)
else RETURN (*i*+*1*, *NONE*)
else RETURN (*i*+*1*, *NONE*)
}})
(*0*, *SUBSUMED-BY C*);
RETURN st
}›

definition *push-to-occs-list2-pre* :: ‹-› **where**
‹*push-to-occs-list2-pre C S occs* ⟷
(*C* ∈# *dom-m* (*get-clauses-wl S*) ∧ *length* (*get-clauses-wl S* ∝ *C*) ≥ *2* ∧ *fst occs* = *set-mset* (*all-init-atms-st S*) ∧
atm-of ‘ *set* (*get-clauses-wl S* ∝ *C*) ⊆ *set-mset* (*all-init-atms-st S*) ∧
C ∉# *all-occurrences* (*mset-set* (*fst occs*)) *occs*)›

definition *push-to-occs-list2* **where**
‹*push-to-occs-list2 C S occs* = *do* {
ASSERT (*push-to-occs-list2-pre C S occs*);
L ← *SPEC* (λ*L*. *L* ∈# *mset* (*get-clauses-wl S* ∝ *C*));
mop-occ-list-append C occs L
}›

definition *maybe-push-to-occs-list2* **where**
‹*maybe-push-to-occs-list2 C S occs* = *do* {
ASSERT (*push-to-occs-list2-pre C S occs*);
b ← *SPEC* (λ-. *True*);
if b then do {
L ← *SPEC* (λ*L*. *L* ∈# *mset* (*get-clauses-wl S* ∝ *C*));
mop-occ-list-append C occs L

```
    } else RETURN occs
  }›

definition isa-is-candidate-forward-subsumption where
  ‹isa-is-candidate-forward-subsumption S C = do {
    ASSERT(arena-act-pre (get-clauses-wl-heur S) C);
    lbd ← mop-arena-lbd (get-clauses-wl-heur S) C;
    sze ← mop-arena-length (get-clauses-wl-heur S) C;
    status ← mop-arena-status (get-clauses-wl-heur S) C;
    ASSERT (sze ≤ length (get-clauses-wl-heur S));
    (-, added) ← WHILE_T (λ(i, added). i < sze ∧ added ≤ 2)
      (λ(i, added). do {
          ASSERT (i < sze);
          L ← mop-arena-lit2 (get-clauses-wl-heur S) C i;
          is-added ← mop-is-marked-added-heur (get-heur S) (atm-of L);
          RETURN (i+1, added + (if is-added then 1 else 0))
    }) (0, 0 :: 64 word);
    let (lbd-limit, size-limit) = get-lsize-limit-stats-st S;
    let can-del =
      sze ≠ 2 ∧ (status = LEARNED ⟶ lbd ≤ lbd-limit ∧ sze ≤ size-limit) ∧ (added ≥ 2);
    RETURN can-del
  }›

definition find-best-subsumption-candidate where
  ‹find-best-subsumption-candidate C S = do {
    L ← mop-arena-lit2 (get-clauses-wl-heur S) C 0;
    ASSERT (nat-of-lit L < length (get-occs S));
    score ← mop-cocc-list-length (get-occs S) L;
    n ← mop-arena-length-st S C;
   (i,score,L) ← WHILE_T (λ(i,score,L). i < n)
     (λ(i,score,L). do {
       ASSERT (Suc i ≤ unat32-max);
       new-L ← mop-arena-lit2 (get-clauses-wl-heur S) C i;
       ASSERT (nat-of-lit L < length (get-occs S));
       new-score ← mop-cocc-list-length (get-occs S) L;
       if new-score < score then RETURN (i+1, new-score, new-L) else RETURN (i+1, score, L)
     })
     (1, score, L);
    RETURN L
  }›

definition isa-push-to-occs-list-st where
  ‹isa-push-to-occs-list-st C S = do {
     L ← find-best-subsumption-candidate C S;
     ASSERT (length (get-occs S ! nat-of-lit L) < length (get-clauses-wl-heur S));
     occs ← mop-cocc-list-append C (get-occs S) L;
     RETURN (set-occs-wl-heur occs S)
  }›


definition find-best-subsumption-candidate-and-push where
  ‹find-best-subsumption-candidate-and-push C S = do {
    L ← mop-arena-lit2 (get-clauses-wl-heur S) C 0;
    ASSERT (nat-of-lit L < length (get-occs S));
    score ← mop-cocc-list-length (get-occs S) L;
    n ← mop-arena-length-st S C;
```

```
  (i,score,L,push) ← WHILE_T (λ(i,score,L,push). i < n ∧ push)
    (λ(i,score,L,push). do {
      ASSERT (Suc i ≤ unat32-max);
      new-L ← mop-arena-lit2 (get-clauses-wl-heur S) C i;
      ASSERT (nat-of-lit L < length (get-occs S));
      new-score ← mop-cocc-list-length (get-occs S) L;
      b ← mop-is-marked-added-heur (get-heur S) (atm-of L);
      if new-score < score then RETURN (i+1, new-score, new-L,b) else RETURN (i+1, score, L,b)
    })
    (1, score, L, True);
    RETURN (L, push)
  }›

definition isa-maybe-push-to-occs-list-st where
  ‹isa-maybe-push-to-occs-list-st C S = do {
    (L, push) ← find-best-subsumption-candidate-and-push C S;
    if push then do {
      let L = L;
      ASSERT (length (get-occs S ! nat-of-lit L) < length (get-clauses-wl-heur S));
      occs ← mop-cocc-list-append C (get-occs S) L;
      RETURN (set-occs-wl-heur occs S)
    } else RETURN S
  }›

definition forward-subsumption-one-wl2-pre :: ‹nat ⇒ nat multiset ⇒ nat literal ⇒ nat twl-st-wl ⇒
bool› where
  ‹forward-subsumption-one-wl2-pre = (λC cands L S.
  forward-subsumption-one-wl-pre C cands S ∧ L ∈# all-init-lits-of-wl S)›

definition isa-forward-subsumption-one-wl-pre :: ‹-› where
  ‹isa-forward-subsumption-one-wl-pre C L S ⟷
  (∃ T r u cands. (S,T)∈twl-st-heur-restart-occs' r u ∧ forward-subsumption-one-wl2-pre C cands L T)
›

definition forward-subsumption-one-wl2-inv :: ‹'v twl-st-wl ⇒ nat ⇒ nat multiset ⇒ nat list ⇒
  nat × 'v subsumption ⇒ bool› where
  ‹forward-subsumption-one-wl2-inv = (λS C cands ys (i, x). forward-subsumption-one-wl-inv S C (mset
ys) (mset (drop i ys), x))›

definition isa-forward-subsumption-one-wl2-inv :: ‹isasat ⇒ nat ⇒ nat literal ⇒
  nat × nat subsumption ⇒ bool› where
  ‹isa-forward-subsumption-one-wl2-inv = (λS C L (ix).
  (∃ T r u cands. (S,T)∈twl-st-heur-restart-occs' r u ∧
    forward-subsumption-one-wl2-inv T C cands (get-occs S ! nat-of-lit L) (ix)))›

definition isa-subsume-clauses-match2-pre :: ‹-› where
  ‹isa-subsume-clauses-match2-pre C C' S D ⟷ (
  ∃ T r u D'. (S,T)∈twl-st-heur-restart-occs' r u ∧ subsume-clauses-match2-pre C C' T D' ∧
    (D,D') ∈ clause-hash) ›

definition isa-subsume-clauses-match2 :: ‹nat ⇒ nat ⇒ isasat ⇒ bool list ⇒ nat subsumption nres›
where
  ‹isa-subsume-clauses-match2 C' C N D = do {
  ASSERT (isa-subsume-clauses-match2-pre C' C N D);
  n ← mop-arena-length-st N C';
  ASSERT (n ≤ length (get-clauses-wl-heur N));
```

```
  (i, st) ← WHILE_T^{λ(i,s). True} (λ(i, st). i < n∧ st ≠ NONE)
    (λ(i, st). do {
      ASSERT (i < n);
      L ← mop-arena-lit2 (get-clauses-wl-heur N) C' i;
      lin ← mop-cch-in L D;
      if lin
      then RETURN (i+1, st)
      else do {
      lin ← mop-cch-in (−L) D;
      if lin
      then if is-subsumed st
      then do {RETURN (i+1, STRENGTHENED-BY L C')}
      else do {RETURN (i+1, NONE)}
      else do {RETURN (i+1, NONE)}
    }})
     (0, SUBSUMED-BY C');
  RETURN st
  }›

definition isa-subsume-or-strengthen-wl-pre :: ‹-› where
  ‹isa-subsume-or-strengthen-wl-pre C s S ⟷
   (∃ T r u.  (S,T)∈twl-st-heur-restart-occs' r u ∧ subsume-or-strengthen-wl-pre C s T)›


definition remove-lit-from-clause where
  ‹remove-lit-from-clause N C L = do {
    n ← mop-arena-length N C;
   (i, j, N) ← WHILE_T (λ(i, j, N). j < n)
     (λ(i, j, N). do {
       ASSERT (i < n);
       ASSERT (j < n);
       K ← mop-arena-lit2 N C j;
       if K ≠ L then do {
         N ← mop-arena-swap C i j N;
         RETURN (i+1, j+1, N)}
       else RETURN (i, j+1, N)
     }) (0, 0, N);
   N ← mop-arena-shorten C i N;
   N ← update-lbd-shrunk-clause C N;
   RETURN N
  }›

definition remove-lit-from-clause-st :: ‹-› where
  ‹remove-lit-from-clause-st T C L = do {
    N ← remove-lit-from-clause (get-clauses-wl-heur T) C L;
    RETURN (set-clauses-wl-heur N T)
}›

definition mark-garbage-heur-as-subsumed :: ‹nat ⇒ isasat ⇒ isasat nres› where
  ‹mark-garbage-heur-as-subsumed C S = (do{
    let N' = get-clauses-wl-heur S;
    ASSERT (arena-is-valid-clause-vdom N' C);
    - ← log-del-clause-heur S C;
    let st = arena-status N' C = IRRED;
    ASSERT (mark-garbage-pre (N', C));
    let N' = extra-information-mark-to-delete (N') C;
```

```
    size ← mop-arena-length (get-clauses-wl-heur S) C;
    let lcount = get-learned-count S;
    ASSERT(¬st ⟶ clss-size-lcount lcount ≥ 1);
    let lcount = (if st then lcount else (clss-size-decr-lcount lcount));
    let stats = get-stats-heur S;
    let stats = (if st then decr-irred-clss stats else stats);
    let S = set-clauses-wl-heur N' S;
    let S = set-learned-count-wl-heur lcount S;
    let S = set-stats-wl-heur stats S;
    let S = incr-wasted-st (of-nat size) S;
    RETURN S
  })›

definition isa-strengthen-clause-wl2 where
  ‹isa-strengthen-clause-wl2 C C' L S = do {
    m ← mop-arena-length (get-clauses-wl-heur S) C;
    n ← mop-arena-length (get-clauses-wl-heur S) C';
    st1 ← mop-arena-status (get-clauses-wl-heur S) C;
    st2 ← mop-arena-status (get-clauses-wl-heur S) C';
    S ← remove-lit-from-clause-st S C (−L);
    - ← log-new-clause-heur S C;
    let - = mark-clause-for-unit-as-changed 0;
    if m = n
    then do {
      S ← RETURN S;
      S ← (if st1 = LEARNED ∧ st2 = IRRED then mop-arena-promote-st S C else RETURN S);
      S ← mark-garbage-heur-as-subsumed C' S;
       RETURN (set-stats-wl-heur (incr-forward-strengethening (get-stats-heur S)) S)
    }
    else
     RETURN (set-stats-wl-heur (incr-forward-strengethening (get-stats-heur S)) S)
  }›

definition incr-forward-subsumed-st :: ‹-› where
  ‹incr-forward-subsumed-st S = (set-stats-wl-heur (incr-forward-subsumed (get-stats-heur S)) S)›

definition incr-forward-tried-st :: ‹-› where
  ‹incr-forward-tried-st S = (set-stats-wl-heur (incr-forward-tried (get-stats-heur S)) S)›

definition incr-forward-rounds-st :: ‹-› where
  ‹incr-forward-rounds-st S = (set-stats-wl-heur (incr-forward-rounds (get-stats-heur S)) S)›

definition incr-forward-strengthened-st :: ‹-› where
  ‹incr-forward-strengthened-st S = (set-stats-wl-heur (incr-forward-strengethening (get-stats-heur S))
S)›

definition isa-subsume-or-strengthen-wl :: ‹nat ⇒ nat subsumption ⇒ isasat ⇒ isasat nres› where
  ‹isa-subsume-or-strengthen-wl = (λC s S. do {
   ASSERT(isa-subsume-or-strengthen-wl-pre C s S);
   (case s of
     NONE ⇒ RETURN S
  | SUBSUMED-BY C' ⇒ do {
     st1 ← mop-arena-status (get-clauses-wl-heur S) C;
     st2 ← mop-arena-status (get-clauses-wl-heur S) C';
     S ← mark-garbage-heur2 C S;
     let - = mark-clause-for-unit-as-changed 0;
```

```
    S ← (if st1 = IRRED ∧ st2 = LEARNED then mop-arena-promote-st S C′ else RETURN S);
    let S = (set-stats-wl-heur (incr-forward-subsumed (get-stats-heur S)) S);
    RETURN S
  }
  | STRENGTHENED-BY L C′ ⇒ isa-strengthen-clause-wl2 C C′ L S)
  })›

definition mop-cch-remove-one where
  ‹mop-cch-remove-one L D = do {
    ASSERT (nat-of-lit L < length D);
    RETURN (D[nat-of-lit L := False])
  } ›

definition mop-cch-remove-all-clauses where
  ‹mop-cch-remove-all-clauses S C D = do {
    n ← mop-arena-length (get-clauses-wl-heur S) C;
    (-, D) ← WHILE_T (λ(i, D). i < n)
    (λ(i, D). do {ASSERT (i < length (get-clauses-wl-heur S)); L ← mop-arena-lit2 (get-clauses-wl-heur
S) C i; D ← mop-cch-remove-one L D; RETURN (i+1, D)})
     (0, D);
    RETURN D
  } ›

definition isa-forward-subsumption-one-wl :: ‹nat ⇒ bool list ⇒ nat literal ⇒ isasat ⇒ (isasat × nat
subsumption × bool list) nres› where
  ‹isa-forward-subsumption-one-wl = (λC D L S. do {
  ASSERT (isa-forward-subsumption-one-wl-pre C L S);
  ASSERT (nat-of-lit L < length (get-occs S));
  n ← mop-cocc-list-length (get-occs S) L;
  (-, s) ←
    WHILE_T isa-forward-subsumption-one-wl2-inv S C L  (λ(i, s). i < n ∧ s = NONE)
    (λ(i, s). do {
      ASSERT (i < n);
      C′ ← mop-cocc-list-at (get-occs S) L i;
      status ← mop-arena-status (get-clauses-wl-heur S) C′;
      if status = DELETED
      then RETURN (i+1, s)
      else do {
        s ← isa-subsume-clauses-match2 C′ C S D;
       RETURN (i+1, s)
      }
    })
       (0, NONE);
  D ← (if s ≠ NONE then mop-cch-remove-all-clauses S C D else RETURN D);
  S ← (if is-strengthened s then isa-maybe-push-to-occs-list-st C S else RETURN S);
  S ← isa-subsume-or-strengthen-wl C s S;
  RETURN (S, s, D)
  })›

definition try-to-forward-subsume-wl2-pre :: ‹-› where
  ‹try-to-forward-subsume-wl2-pre C cands shrunken S ⟷
    distinct-mset cands ∧
    try-to-forward-subsume-wl-pre C cands S›

definition isa-try-to-forward-subsume-wl-pre :: ‹-› where
  ‹isa-try-to-forward-subsume-wl-pre C shrunken S ⟷
```

(∃ T r u cands occs′. (S,T)∈twl-st-heur-restart-occs′ r u ∧ (get-occs S, occs′) ∈ occurrence-list-ref ∧
try-to-forward-subsume-wl2-pre C cands (mset shrunken) T)›

definition try-to-forward-subsume-wl2-inv :: ‹-› **where**
‹try-to-forward-subsume-wl2-inv S cands C = (λ(i, changed, break, occs, D, T).
try-to-forward-subsume-wl-inv S cands C (i, break, T) ∧
(¬changed ⟶ (D, mset (get-clauses-wl T ∝ C)) ∈ clause-hash-ref (all-init-atms-st T)) ∧
(changed ⟶ (D, {#}) ∈ clause-hash-ref (all-init-atms-st T)))›

definition isa-try-to-forward-subsume-wl-inv :: ‹isasat ⇒ nat ⇒ nat × nat subsumption × bool × bool list × isasat ⇒ bool› **where**
‹isa-try-to-forward-subsume-wl-inv S C = (λ(i, changed, break, D, T).
(∃ S′ T′ cands occs′ D′. (S,S′)∈twl-st-heur-restart-occs′ (length (get-clauses-wl-heur S)) (learned-clss-count S) ∧
(T,T′)∈twl-st-heur-restart-occs′ (length (get-clauses-wl-heur S)) (learned-clss-count S)∧
(get-occs T, occs′) ∈ occurrence-list-ref ∧
(D,D′) ∈clause-hash ∧
try-to-forward-subsume-wl2-inv S′ cands C (i, changed ≠ NONE, break, occs′, D′, T′)))›

definition isa-try-to-forward-subsume-wl2-break :: ‹isasat ⇒ bool nres› **where**
‹isa-try-to-forward-subsume-wl2-break S = RETURN False›

definition isa-try-to-forward-subsume-wl2 :: ‹nat ⇒ bool list ⇒ nat list ⇒ isasat ⇒ (bool list × nat list × isasat) nres› **where**
‹isa-try-to-forward-subsume-wl2 C D shrunken S = do {
ASSERT (isa-try-to-forward-subsume-wl-pre C shrunken S);
n ← mop-arena-length-st S C;
ASSERT (n ≤ Suc (unat32-max div 2));
let n = 2 * n;
ebreak ← isa-try-to-forward-subsume-wl2-break S;
(-, changed, -, D, S) ← WHILE$_T$ isa-try-to-forward-subsume-wl-inv S C
(λ(i, changed, break, D, S). ¬break ∧ i < n)
(λ(i, changed, break, D, S). do {
ASSERT (i < n);
L ← mop-arena-lit2 (get-clauses-wl-heur S) C (i div 2);
let L = (if i mod 2 = 0 then L else − L);
(S, subs, D) ← isa-forward-subsumption-one-wl C D L S;
ebreak ← isa-try-to-forward-subsume-wl2-break S;
RETURN (i+1, subs, (subs ≠ NONE) ∨ ebreak, D, S)
})
(0, NONE, ebreak, D, S);
D ← (if changed = NONE then mop-cch-remove-all-clauses S C D else RETURN D);
let - = (if changed = NONE then mark-clause-for-unit-as-unchanged 0 else ());
ASSERT (Suc (length shrunken) ≤ length (get-tvdom S));
let add-to-shunken = (is-strengthened changed);
let shrunken = (if add-to-shunken then shrunken @ [C] else shrunken);
RETURN (D, shrunken, S)
}›

definition isa-forward-subsumption-pre-all :: ‹-› **where**
‹isa-forward-subsumption-pre-all S ⟷
(∃ T r u. (S,T)∈twl-st-heur-restart-ana′ r u ∧ forward-subsumption-all-wl-pre T)›

definition correct-occurence-list **where**
‹correct-occurence-list S occs cands n ⟷

distinct-mset cands ∧
all-occurrences (*all-init-atms-st S*) *occs* ∩# *cands* = {#} ∧
(∀ *C*∈# *all-occurrences* (*all-init-atms-st S*) *occs*. *C* ∈# *dom-m* (*get-clauses-wl S*) ⟶ *length* (*get-clauses-wl S* ∝ *C*) ≤ *n*)∧
(∀ *C*∈# *all-occurrences* (*all-init-atms-st S*) *occs*. *C* ∈# *dom-m* (*get-clauses-wl S*) ⟶
(∀ *L*∈*set* (*get-clauses-wl S* ∝ *C*). *undefined-lit* (*get-trail-wl S*) *L*)) ∧
fst occs = *set-mset* (*all-init-atms-st S*)›

definition *populate-occs-inv* **where**
‹*populate-occs-inv S xs* = (λ(*i*, *occs*, *cands*).
all-occurrences (*all-init-atms-st S*) *occs* + *mset cands* ⊆# *mset* (*take i xs*) ∩# *dom-m* (*get-clauses-wl S*) ∧
distinct cands ∧ *fst occs* = *set-mset* (*all-init-atms-st S*) ∧
correct-occurence-list S occs (*mset* (*drop i xs*)) *2* ∧
all-occurrences (*all-init-atms-st S*) *occs* ∩# *mset cands* = {#} ∧
(∀ *C*∈# *all-occurrences* (*all-init-atms-st S*) *occs*. *C* ∈# *dom-m* (*get-clauses-wl S*) ∧
(∀ *L*∈*set* (*get-clauses-wl S* ∝ *C*). *undefined-lit* (*get-trail-wl S*) *L*) ∧ *length* (*get-clauses-wl S* ∝ *C*) = *2*) ∧
(∀ *C*∈*set cands*. *C* ∈# *dom-m* (*get-clauses-wl S*) ∧ *length* (*get-clauses-wl S* ∝ *C*) > *2* ∧ (∀ *L*∈*set* (*get-clauses-wl S* ∝ *C*). *undefined-lit* (*get-trail-wl S*) *L*)))›

definition *isa-populate-occs-inv* **where**
‹*isa-populate-occs-inv S xs* = (λ(*i*, *U*).
(∃ *T U′ occs*. (*S*,*T*)∈*twl-st-heur-restart-occs′* (*length* (*get-clauses-wl-heur S*)) (*learned-clss-count S*) ∧
(*get-occs U*, *occs*) ∈ *occurrence-list-ref* ∧
(*U*,*U′*)∈*twl-st-heur-restart-occs′* (*length* (*get-clauses-wl-heur S*)) (*learned-clss-count S*) ∧ *populate-occs-inv T xs* (*i*, *occs*, *get-tvdom U*)))›

definition *isa-all-lit-clause-unset-pre* :: ‹-› **where**
‹*isa-all-lit-clause-unset-pre C S* ⟷ (∃ *T r u*. (*S*,*T*)∈*twl-st-heur-restart-occs′ r u* ∧ *forward-subsumption-all-wl-pre T* ∧ *C*∈*set* (*get-vdom S*)) ›

definition *isa-all-lit-clause-unset* **where**
‹*isa-all-lit-clause-unset C S* = *do* {
ASSERT (*isa-all-lit-clause-unset-pre C S*);
not-garbage ← *mop-clause-not-marked-to-delete-heur S C*;
if ¬*not-garbage then RETURN False*
else do {
n ← *mop-arena-length-st S C*;
(*i*, *unset*, *added*) ← *WHILE*$_T$ (λ(*i*, *unset*, -). *unset* ∧ *i* < *n*)
(λ(*i*, *unset*, *added*). *do* {
ASSERT (*i*+*1* ≤ *Suc* (*unat32-max div 2*));
ASSERT(*Suc i* ≤ *unat32-max*);
L ← *mop-arena-lit2* (*get-clauses-wl-heur S*) *C i*;
val ← *mop-polarity-pol* (*get-trail-wl-heur S*) *L*;
is-added ← *mop-is-marked-added-heur-st S* (*atm-of L*);
RETURN (*i*+*1*, *val* = *None*, *if is-added then added* + *1 else added*)
}) (*0*, *True*, *0*::*64 word*);
let a = (*added* ≥ *2*);
RETURN (*unset* ∧ *a*)
}
}›

definition *forward-subsumption-all-wl2-inv* :: ‹*nat twl-st-wl* ⇒ *nat list* ⇒ *nat* × - × - × *nat twl-st-wl*

× *nat* × *nat multiset* ⇒ *bool*› **where**
‹*forward-subsumption-all-wl2-inv* = (λ*S xs* (*i*, *occs*, *D*, *s*, *n*, *shrunken*).
(*D*, {#}) ∈ *clause-hash-ref* (*all-init-atms-st s*) ∧ *shrunken* ⊆# *mset* (*take i xs*) ∧
forward-subsumption-all-wl-inv S (*mset xs*) (*mset* (*drop i xs*), *s*))
›

definition *sort-cands-by-length* **where**
‹*sort-cands-by-length S* = *do* {
let tvdom = *get-tvdom S*;
let avdom = *get-avdom S*;
let ivdom = *get-ivdom S*;
let vdom = *get-vdom S*;
ASSERT (∀ *i*∈*set tvdom*. *arena-is-valid-clause-idx* (*get-clauses-wl-heur S*) *i*);
tvdom ← *SPEC* (λ*cands'*. *mset cands'* = *mset tvdom* ∧
sorted-wrt (λ*a b*. *arena-length* (*get-clauses-wl-heur S*) *a* ≤ *arena-length* (*get-clauses-wl-heur S*) *b*) *cands'*);
RETURN (*set-aivdom-wl-heur* (*AIvdom* (*vdom*, *avdom*, *ivdom*, *tvdom*)) *S*)
}›

definition *push-to-tvdom-st* :: ‹*nat* ⇒ *isasat* ⇒ *isasat nres*› **where**
‹*push-to-tvdom-st C S* = *do* {
ASSERT (*length* (*get-vdom S*) ≤ *length* (*get-clauses-wl-heur S*));
ASSERT (*length* (*get-tvdom S*) < *length* (*get-clauses-wl-heur S*));
let av = *get-aivdom S*; *let av* = *push-to-tvdom C av*;
RETURN (*set-aivdom-wl-heur av S*)
}›

definition *empty-tvdom-st* :: ‹*isasat* ⇒ *isasat nres*› **where**
‹*empty-tvdom-st S* = *do* {
let aivdom = *get-aivdom S*;
let aivdom = *empty-tvdom aivdom*;
RETURN (*set-aivdom-wl-heur aivdom S*)
}›

Using *empty-tvdom-st* is mostly laziness: It should actually already be empty, but re-cleaning is not costing much anyhow.

definition *isa-populate-occs* :: ‹*isasat* ⇒ - *nres*› **where**
‹*isa-populate-occs S* = *do* {
ASSERT (*length* (*get-avdom-aivdom* (*get-aivdom S*) @ *get-ivdom-aivdom* (*get-aivdom S*)) ≤ *length* (*get-clauses-wl-heur S*));
let xs = *get-avdom-aivdom* (*get-aivdom S*) @ *get-ivdom-aivdom* (*get-aivdom S*);
let m = *size* (*get-avdom-aivdom* (*get-aivdom S*));
let n = *size xs*;
let occs = *get-occs S*;
ASSERT (*n* ≤ *length* (*get-clauses-wl-heur S*));
T ← *empty-tvdom-st S*;
(*xs*, *S*) ← *WHILE*$_T^{\textit{isa-populate-occs-inv } S\ xs}$ (λ(*i*, *S*). *i* < *n*)
(λ(*i*, *S*). *do* {
ASSERT (*i* < *n*);
ASSERT (*Suc i* ≤ *length* (*get-avdom-aivdom* (*get-aivdom S*) @ *get-ivdom-aivdom* (*get-aivdom S*)));
ASSERT (*i* < *m* ⟶ *access-avdom-at-pre S i*);
ASSERT (*i* ≥ *m* ⟶ *access-ivdom-at-pre S* (*i* − *m*));
let C = (*if i* < *m then get-avdom-aivdom* (*get-aivdom S*) ! *i else get-ivdom-aivdom* (*get-aivdom S*) ! (*i*−*m*));

```
    ASSERT (C ∈ set (get-vdom S));
    all-undef ← isa-all-lit-clause-unset C S;
    if ¬all-undef then
      RETURN (i+1, S)
    else do {
      n ← mop-arena-length-st S C;
      if n = 2 then do {
        S ← isa-push-to-occs-list-st C S;
        RETURN (i+1, S)
      }
      else  do {
        cand ← isa-is-candidate-forward-subsumption S C;
        if cand then do {S ← push-to-tvdom-st C S; RETURN (i+1, S)}
        else RETURN (i+1, S)
        }
      }
         }
    )
    (0, T);
    T ← sort-cands-by-length S;
    RETURN T
  }›

definition mop-cch-add-all-clause :: ‹-› where
  ‹mop-cch-add-all-clause S C D = do {
    n ← mop-arena-length-st S C;
    ASSERT (n ≤ length (get-clauses-wl-heur S));
    (-, D) ← WHILE_T (λ(i, D). i < n)
    (λ(i,D). do {
      ASSERT (i < n);
      L ← mop-arena-lit2 (get-clauses-wl-heur S) C i;
      let - = mark-literal-for-unit-deletion L;
      D ← mop-cch-add L D;
      RETURN (i+1, D)
      }) (0, D);
    RETURN D
}›

definition mop-ch-add-all-clause :: ‹-› where
  ‹mop-ch-add-all-clause S C D = do {
    ASSERT (C ∈# dom-m (get-clauses-wl S));
    let n = length (get-clauses-wl S ∝ C);
    (-, D) ← WHILE_T (λ(i, D). i < n)
    (λ(i,D). do {
      L ← mop-clauses-at (get-clauses-wl S) C i;
      D ← mop-ch-add L D;
      RETURN (i+1, D)
      }) (0, D);
    RETURN D
}›

definition empty-occs-st :: ‹isasat ⇒ isasat nres› where
  ‹empty-occs-st S = do {
    let D = get-occs S;
    let D = replicate (length D) [];
    RETURN (set-occs-wl-heur D S)
```

}›

definition *empty-occs2* **where**
‹*empty-occs2 occs*$_0$ = *do* {
let n = length occs$_0$;
(-, *occs*)← *WHILE*$_T$ (λ(*i*,*occs*). *i* < *n*)
(λ(*i*,*occs*). *do* {
ASSERT (*i* < *n* ∧ *length occs* = *length occs*$_0$);
RETURN (*i*+*1*, *occs*[*i* := *take 0* (*occs* ! *i*)])
}) (*0*, *occs*$_0$);
RETURN occs
}›

definition *isa-forward-reset-added-and-stats* **where**
‹*isa-forward-reset-added-and-stats S* =
(*let S* = (*set-stats-wl-heur* (*incr-forward-rounds* (*get-stats-heur S*)) *S*) *in*
set-heur-wl-heur (*reset-added-heur* (*get-heur S*)) *S*)›

definition *empty-occs2-st* :: ‹*isasat* ⇒ *isasat nres*› **where**
‹*empty-occs2-st S* = *do* {
let D = *get-occs S*;
D ← *empty-occs2 D*;
RETURN (*set-occs-wl-heur D S*)
}›

definition *forward-subsumption-finalize* :: ‹*nat list* ⇒ *isasat* ⇒ *isasat nres*› **where**
‹*forward-subsumption-finalize shrunken S* = *do* {
let S = *isa-forward-reset-added-and-stats* (*schedule-next-subsume-st* ((*1* + *stats-forward-rounds-st S*)
∗ *10000*) *S*);
- ← *isasat-current-progress 115 S*;
(-, *S*) ← *WHILE*$_T$(λ(*i*,*S*). *i* < *length shrunken*) (λ(*i*,*S*). *do* {
ASSERT (*i* < *length shrunken*);
let C = *shrunken* ! *i*;
not-garbage ← *mop-clause-not-marked-to-delete-heur S C*;
S ← (*if not-garbage then mark-added-clause-heur2 S C else RETURN S*);
RETURN (*i*+*1*, *S*)
}) (*0*, *S*);
empty-occs2-st S
}›

definition *isa-forward-subsumption-all-wl-inv* :: ‹-› **where**
‹*isa-forward-subsumption-all-wl-inv R*$_0$ *S* =
(λ(*i*, *D*, *shrunken*, *T*). ∃ *R*$_0$′ *S*′ *T*′ *D*′ *occs*′ *n*. (*R*$_0$, *R*$_0$′)∈*twl-st-heur-restart-occs*′ (*length* (*get-clauses-wl-heur*
R$_0$)) (*learned-clss-count R*$_0$) ∧
(*S*, *S*′)∈*twl-st-heur-restart-occs*′ (*length* (*get-clauses-wl-heur R*$_0$)) (*learned-clss-count R*$_0$) ∧
(*T*,*T*′)∈*twl-st-heur-restart-occs*′ (*length* (*get-clauses-wl-heur R*$_0$)) (*learned-clss-count R*$_0$) ∧ (*get-occs*
T, *occs*′) ∈ *occurrence-list-ref* ∧
(*D*, *D*′) ∈ *clause-hash* ∧
forward-subsumption-all-wl2-inv S′ (*get-tvdom S*) (*i*, *occs*′, *D*′, *T*′, *n*, *mset shrunken*)) ›

definition *isa-forward-subsumption-all* :: ‹- ⇒ - *nres*› **where**
‹*isa-forward-subsumption-all* = (λ*S*$_0$. *do* {
ASSERT (*isa-forward-subsumption-pre-all S*$_0$);
S ← *isa-populate-occs S*$_0$;
ASSERT (*isasat-fast-relaxed S*$_0$ ⟶ *isasat-fast-relaxed S*);
let m = *length* (*get-tvdom S*);

```
  D ← mop-cch-create (length (get-watched-wl-heur S));
  let shrunken = [];
  (-, D, shrunken, S) ←
   WHILE_T isa-forward-subsumption-all-wl-inv S_0 S (λ(i, D, shrunken, S). i < m ∧ get-conflict-wl-is-None-heur
S)
    (λ(i, D, shrunken, S). do {
      ASSERT (i < m);
      ASSERT (access-tvdom-at-pre S i);
      let C = get-tvdom S!i;
      D ← mop-cch-add-all-clause S C D;
      (D, shrunken, T) ← isa-try-to-forward-subsume-wl2 C D shrunken S;
      RETURN (i+1, D, shrunken, incr-forward-tried-st T)
    })
    (0, D, shrunken, S);
  ASSERT (∀ C∈set shrunken. C ∈ set (get-vdom S));
  forward-subsumption-finalize shrunken S
  }
)›

definition isa-forward-subsume :: ‹isasat ⇒ isasat nres› where
  ‹isa-forward-subsume S = do {
    let b = should-subsume-st S;
    if b then isa-forward-subsumption-all S else RETURN S
  }›

end
theory IsaSAT-Simplify-Forward-Subsumption
  imports IsaSAT-Setup
    Watched-Literals.Watched-Literals-Watch-List-Inprocessing
    More-Refinement-Libs.WB-More-Refinement-Loops
    IsaSAT-Restart
    IsaSAT-Simplify-Forward-Subsumption-Defs
begin


lemma subsume-clauses-match2-subsume-clauses-match:
  assumes
    ‹(C, E) ∈ nat-rel›
    ‹(C', F) ∈ nat-rel› and
    DG: ‹(D, G) ∈ clause-hash-ref (all-init-atms-st S)› and
    N: ‹N = get-clauses-wl S› and
    G: ‹G = mset (get-clauses-wl S ∝ C')› and
    lin: ‹literals-are-ℒ_in' S›
  shows ‹subsume-clauses-match2 C C' S D ≤ ⇓Id (subsume-clauses-match E F N)›
proof −
  have eq: ‹E = C› ‹F = C'›
    using assms by auto
  have [simp]: ‹set-mset (all-init-atms-st S) = set-mset (all-atms-st S)›
    using lin unfolding literals-are-ℒ_in'-def all-atms-st-alt-def
    by (metis Un-subset-iff ℒ_all-all-atms ℒ_all-all-init-atms(2) all-atms-st-alt-def-sym
      all-lits-st-init-learned atms-of-ℒ_all-𝒜_in atms-of-cong-set-mset dual-order.antisym subset-refl)
  have [intro!]: ‹C ∈# dom-m (get-clauses-wl S) ⟹ x1a < length (get-clauses-wl S ∝ C) ⟹
    atm-of (get-clauses-wl S ∝ C ! x1a) ∈# all-atms-st S› for x1a
    unfolding all-atms-st-alt-def
    by (auto intro!: nth-in-all-lits-stI simp del: all-atms-st-alt-def[symmetric])
  have [refine]: ‹((0, SUBSUMED-BY C), 0, SUBSUMED-BY C) ∈ nat-rel ×_f Id›
```

by *auto*
have *subsume-clauses-match-alt-def*:
‹*subsume-clauses-match C C′ N* = *do* {
ASSERT (*subsume-clauses-match-pre C C′ N*);
(*i*, *st*) ← *WHILE*$_T$ λ(*i*,*s*). *try-to-subsume C′ C* (*N* (*C* ↪ *take i* (*N* ∝ *C*))) *s* (λ(*i*, *st*). *i* < *length* (*N* ∝ *C*) ∧ *st* ≠ *NONE*)
(λ(*i*, *st*). *do* {
let L = *N* ∝ *C* ! *i*;
let lin = *L* ∈# *mset* (*N* ∝ *C′*);
if lin
then RETURN (*i*+*1*, *st*)
else let lin = −*L* ∈# *mset* (*N* ∝ *C′*) *in*
if lin
then if is-subsumed st
then RETURN (*i*+*1*, *STRENGTHENED-BY L C*)
else RETURN (*i*+*1*, *NONE*)
else RETURN (*i*+*1*, *NONE*)
})
(*0*, *SUBSUMED-BY C*);
RETURN st
}› **for** *C C′ N*
unfolding *subsume-clauses-match-def Let-def* **by** (*auto cong*: *if-cong*)
have [*refine*]: ‹(*x1a*, *x1*) ∈ *nat-rel* ⟹ (*get-clauses-wl S* ∝ *C* ! *x1a*, *get-clauses-wl S* ∝ *C* ! *x1*) ∈ *Id*› **for** *x1 x1a*
by *auto*
show *?thesis*
unfolding *N subsume-clauses-match2-def subsume-clauses-match-alt-def Let-def*[*of* ‹*length* -›] *eq mop-clauses-at-def nres-monad3*
apply (*refine-rcg DG*[*unfolded G*] *mop-ch-in*)
subgoal using *DG G* **unfolding** *subsume-clauses-match2-pre-def clause-hash-ref-def*
by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *subsume-clauses-match-pre-def*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *subsume-clauses-match-pre-def*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

definition *forward-subsumption-one-wl2-rel* **where**
‹*forward-subsumption-one-wl2-rel S*$_0$ *occs cands n C D* = {((*S*, *changed*, *occs′*, *D′*), (*T*, *changed′*)). (¬*changed* ⟶ *C* ∈# *dom-m* (*get-clauses-wl S*)) ∧
changed = *changed′* ∧ *set-mset* (*all-init-atms-st S*) = *set-mset* (*all-init-atms-st S*$_0$) ∧
(*changed* ⟶ (*D′*, {#}) ∈ *clause-hash-ref* (*all-init-atms-st S*)) ∧

$(\neg changed \longrightarrow D' = D \wedge occs' = occs \wedge get\text{-}clauses\text{-}wl\ S \propto C = get\text{-}clauses\text{-}wl\ S_0 \propto C) \wedge$
$correct\text{-}occurence\text{-}list\ S\ occs'\ (if\ changed\ then\ remove1\text{-}mset\ C\ cands\ else\ cands)$
$(if\ changed\ then\ size\ (get\text{-}clauses\text{-}wl\ S_0 \propto C)\ else\ n) \wedge$
$all\text{-}occurrences\ (all\text{-}init\text{-}atms\text{-}st\ S)\ occs' \subseteq\#\ add\text{-}mset\ C\ (all\text{-}occurrences\ (all\text{-}init\text{-}atms\text{-}st\ S)\ occs)$
$\wedge$
$(S,T) \in Id$
$\}\rangle$

lemma *literals-are-$\mathcal{L}_{in}$'-all-init-atms-alt-def*:
$\langle literals\text{-}are\text{-}\mathcal{L}_{in}'\ S \Longrightarrow$
$set\text{-}mset\ (all\text{-}init\text{-}atms\text{-}st\ S) = set\text{-}mset\ (all\text{-}atms\text{-}st\ S)\rangle$
unfolding *literals-are-$\mathcal{L}_{in}$'-def all-atms-st-alt-def*
by (*metis Un-subset-iff $\mathcal{L}_{all}$-all-atms $\mathcal{L}_{all}$-all-init-atms(2) all-atms-st-alt-def-sym*
all-lits-st-init-learned atms-of-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$ atms-of-cong-set-mset dual-order.antisym subset-refl)

lemma
inter-mset-empty-remove1-msetI:
$\langle A \cap\#\ B = \{\#\} \Longrightarrow A \cap\#\ remove1\text{-}mset\ c\ B = \{\#\}\rangle$ **and**
inter-mset-empty-removeAll-msetI:
$\langle A \cap\#\ B = \{\#\} \Longrightarrow A \cap\#\ removeAll\text{-}mset\ c\ B = \{\#\}\rangle$
apply (*meson disjunct-not-in in-diffD*)
by (*metis count-le-replicate-mset-subset-eq dual-order.refl inter-mset-empty-distrib-right*
subset-mset.diff-add)

lemma *push-to-occs-list2*:
assumes *occs*: $\langle correct\text{-}occurence\text{-}list\ S\ occs\ cands\ n\rangle$
$\langle C \in\#\ dom\text{-}m\ (get\text{-}clauses\text{-}wl\ S)\rangle$
$\langle 2 \leq length\ (get\text{-}clauses\text{-}wl\ S \propto C)\rangle$ **and**
lin: $\langle literals\text{-}are\text{-}\mathcal{L}_{in}'\ S\rangle$ **and**
undef: $\langle \forall L \in set\ (get\text{-}clauses\text{-}wl\ S \propto C).\ undefined\text{-}lit\ (get\text{-}trail\text{-}wl\ S)\ L\rangle$ **and**
notin: $\langle C \notin\#\ all\text{-}occurrences\ (mset\text{-}set\ (fst\ occs))\ occs\rangle$
shows $\langle push\text{-}to\text{-}occs\text{-}list2\ C\ S\ occs \leq SPEC\ (\lambda c.\ (c, ()) \in \{(occs', occs'').$
$all\text{-}occurrences\ (all\text{-}init\text{-}atms\text{-}st\ S)\ occs' = add\text{-}mset\ C\ (all\text{-}occurrences\ (all\text{-}init\text{-}atms\text{-}st\ S)\ occs) \wedge$
$correct\text{-}occurence\text{-}list\ S\ occs'\ (remove1\text{-}mset\ C\ cands)\ (max\ n\ (length\ (get\text{-}clauses\text{-}wl\ S \propto C))) \wedge fst\ occs = fst\ occs''\})\rangle$
proof −
have *1*: $\langle atm\text{-}of\ `\ set\ (get\text{-}clauses\text{-}wl\ S \propto C) \subseteq set\text{-}mset\ (all\text{-}atms\text{-}st\ S)\rangle$
using *nth-in-all-lits-stI[of C S] assms(2)*
unfolding *all-atms-st-alt-def*
by (*auto simp del*: *all-atms-st-alt-def[symmetric] simp*: *in-set-conv-nth*)
moreover have $\langle atm\text{-}of\ `\ set\ (get\text{-}clauses\text{-}wl\ S \propto C) \subseteq set\text{-}mset\ (all\text{-}init\text{-}atms\text{-}st\ S)\rangle$
using *1 lin* **unfolding** *literals-are-$\mathcal{L}_{in}$'-def* **by** (*simp add*: *lin literals-are-$\mathcal{L}_{in}$'-all-init-atms-alt-def*)
moreover have $\langle L \in set\ (get\text{-}clauses\text{-}wl\ S \propto C) \Longrightarrow$
$L \in\#\ all\text{-}init\text{-}lits\text{-}of\text{-}wl\ S\rangle$ **for** *L*
by (*metis $\mathcal{L}_{all}$-all-init-atms(2) calculation(2) image-subset-iff in-$\mathcal{L}_{all}$-atm-of-$\mathcal{A}_{in}$*)

ultimately show *?thesis*
using *literals-are-$\mathcal{L}_{in}$'-all-init-atms-alt-def[OF lin]*
unfolding *push-to-occs-list2-def*
apply (*refine-vcg mop-occ-list-append[THEN order-trans]*)
subgoal using *assms* **unfolding** *push-to-occs-list2-pre-def correct-occurence-list-def* **by** *fast*
subgoal using *assms* **unfolding** *occ-list-append-pre-def correct-occurence-list-def*
by *auto*
subgoal for *L occs'*
using *multi-member-split[of* $\langle atm\text{-}of\ L\rangle$ $\langle all\text{-}init\text{-}atms\text{-}st\ S\rangle$*]*

apply (*subgoal-tac* ‹*atm-of L ∈# all-init-atms-st S*›)
apply (*cases occs*)
by (*use assms literals-are-$\mathcal{L}_{in}$'-all-init-atms-alt-def*[*OF lin*] **in**
‹*auto 4 2 simp*: *occ-list-append-def correct-occurence-list-def*
all-occurrences-add-mset occ-list-def all-occurrences-insert-lit
all-occurrences-occ-list-append-r distinct-mset-remove1-All all-init-atms-st-alt-def
intro: *inter-mset-empty-removeAll-msetI*›)
done
qed

lemma *maybe-push-to-occs-list2*:
assumes *occs*: ‹*correct-occurence-list S occs cands n*›
‹*C ∈# dom-m* (*get-clauses-wl S*)›
‹*2 ≤ length* (*get-clauses-wl S ∝ C*)› **and**
lin: ‹*literals-are-$\mathcal{L}_{in}$' S*› **and**
undef: ‹∀ *L*∈*set* (*get-clauses-wl S ∝ C*). *undefined-lit* (*get-trail-wl S*) *L*› **and**
notin: ‹*C ∉# all-occurrences* (*mset-set* (*fst occs*)) *occs*›
shows ‹*maybe-push-to-occs-list2 C S occs ≤ SPEC* (λ*c*. (*c*, ()) ∈ {(*occs'*, *occs''*).
(*all-occurrences* (*all-init-atms-st S*) *occs'* = *add-mset C* (*all-occurrences* (*all-init-atms-st S*) *occs*) ∨
all-occurrences (*all-init-atms-st S*) *occs'* = *all-occurrences* (*all-init-atms-st S*) *occs*) ∧
correct-occurence-list S occs' (*remove1-mset C cands*) (*max n* (*length* (*get-clauses-wl S ∝ C*))) ∧ *fst occs* = *fst occs''*})›
proof −
have *1*: ‹*atm-of* ‘ *set* (*get-clauses-wl S ∝ C*) ⊆ *set-mset* (*all-atms-st S*)›
using *nth-in-all-lits-stI*[*of C S*] *assms*(*2*)
unfolding *all-atms-st-alt-def*
by (*auto simp del*: *all-atms-st-alt-def*[*symmetric*] *simp*: *in-set-conv-nth*)
moreover have ‹*atm-of* ‘ *set* (*get-clauses-wl S ∝ C*) ⊆ *set-mset* (*all-init-atms-st S*)›
using *1 lin* **unfolding** *literals-are-$\mathcal{L}_{in}$'-def* **by** (*simp add*: *lin literals-are-$\mathcal{L}_{in}$'-all-init-atms-alt-def*)
moreover have ‹*L ∈ set* (*get-clauses-wl S ∝ C*) ⟹
L ∈# all-init-lits-of-wl S› **for** *L*
by (*metis $\mathcal{L}_{all}$-all-init-atms*(*2*) *calculation*(*2*) *image-subset-iff in-$\mathcal{L}_{all}$-atm-of-$\mathcal{A}_{in}$*)

ultimately show *?thesis*
using *literals-are-$\mathcal{L}_{in}$'-all-init-atms-alt-def*[*OF lin*]
unfolding *maybe-push-to-occs-list2-def*
apply (*refine-vcg mop-occ-list-append*[*THEN order-trans*])
subgoal using *assms* **unfolding** *push-to-occs-list2-pre-def correct-occurence-list-def* **by** *fast*
subgoal using *assms* **unfolding** *occ-list-append-pre-def correct-occurence-list-def*
by *auto*
subgoal for *keep L occs'*
using *multi-member-split*[*of* ‹*atm-of L*› ‹*all-init-atms-st S*›]
apply (*subgoal-tac* ‹*atm-of L ∈# all-init-atms-st S*›)
apply (*cases occs*)
by (*use assms literals-are-$\mathcal{L}_{in}$'-all-init-atms-alt-def*[*OF lin*] **in**
‹*auto 4 2 simp*: *occ-list-append-def correct-occurence-list-def*
all-occurrences-add-mset occ-list-def all-occurrences-insert-lit
all-occurrences-occ-list-append-r distinct-mset-remove1-All all-init-atms-st-alt-def
intro: *inter-mset-empty-removeAll-msetI*›)
subgoal for *keep*
using *assms* **by** (*auto 4 2 simp*: *occ-list-append-def correct-occurence-list-def*
all-occurrences-add-mset occ-list-def all-occurrences-insert-lit
all-occurrences-occ-list-append-r distinct-mset-remove1-All all-init-atms-st-alt-def
intro: *inter-mset-empty-removeAll-msetI*)
done
qed

definition *forward-subsumption-one-wl2* :: ‹*nat* ⇒ *nat multiset* ⇒ *nat literal* ⇒ *occurences* ⇒ *clause-hash* ⇒
nat twl-st-wl ⇒ (*nat twl-st-wl* × *bool* × *occurences* × *clause-hash*) *nres*› **where**
‹*forward-subsumption-one-wl2* = (λ*C cands L occs D S*. *do* {
ASSERT (*forward-subsumption-one-wl2-pre C cands L S*);
ASSERT (*atm-of L* ∈ *fst occs*);
let ys = *occ-list occs L*;
let n = *length ys*;
(-, *s*) ←
WHILE$_T$ *forward-subsumption-one-wl2-inv S C cands ys* (λ(*i*, *s*). *i* < *n* ∧ *s* = *NONE*)
(λ(*i*, *s*). *do* {
C' ← *mop-occ-list-at occs L i*;
if C' ∉# *dom-m* (*get-clauses-wl S*)
then RETURN (*i*+*1*, *s*)
else do {
s ← *subsume-clauses-match2 C' C S D*;
RETURN (*i*+*1*, *s*)
}
})
(*0*, *NONE*);
D ← (*if s* ≠ *NONE then mop-ch-remove-all* (*mset* (*get-clauses-wl S* ∝ *C*)) *D else RETURN D*);
occs ← (*if is-strengthened s then maybe-push-to-occs-list2 C S occs else RETURN occs*);
S ← *subsume-or-strengthen-wl C s S*;
RETURN (*S*, *s* ≠ *NONE*, *occs*, *D*)
})›

lemma *case-wl-split*:
‹(*case T of* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*) ⇒ *P M N D NE UE NEk UEk NS US N0 U0 Q W*) =
P (*get-trail-wl T*) (*get-clauses-wl T*) (*get-conflict-wl T*) (*IsaSAT-Setup.get-unkept-unit-init-clss-wl T*)
(*IsaSAT-Setup.get-unkept-unit-learned-clss-wl T*) (*IsaSAT-Setup.get-kept-unit-init-clss-wl T*)
(*IsaSAT-Setup.get-kept-unit-learned-clss-wl T*) (*get-subsumed-init-clauses-wl T*)
(*get-subsumed-learned-clauses-wl T*) (*get-init-clauses0-wl T*) (*get-learned-clauses0-wl T*)
(*literals-to-update-wl T*) (*get-watched-wl T*)› **and**
state-wl-recompose:
‹(*get-trail-wl T*, *get-clauses-wl T*, *get-conflict-wl T*, *IsaSAT-Setup.get-unkept-unit-init-clss-wl T*,
IsaSAT-Setup.get-unkept-unit-learned-clss-wl T, *IsaSAT-Setup.get-kept-unit-init-clss-wl T*,
IsaSAT-Setup.get-kept-unit-learned-clss-wl T, *get-subsumed-init-clauses-wl T*,
get-subsumed-learned-clauses-wl T, *get-init-clauses0-wl T*, *get-learned-clauses0-wl T*,
literals-to-update-wl T, *get-watched-wl T*) = *T*›
by (*cases T*; *auto*; *fail*)+

lemma *clause-hash-ref-cong*: ‹*set-mset A* = *set-mset B* ⟹ *clause-hash-ref A* = *clause-hash-ref B*› **for** *A B*
unfolding *clause-hash-ref-def*
by *auto*
lemma *remdups-mset-set-mset-cong*: ‹*set-mset A* = *set-mset B* ⟹ *remdups-mset A* = *remdups-mset B*› **for** *A B*
by (*simp add*: *remdups-mset-def*)
lemma *all-occurrences-cong*: ‹*set-mset A* = *set-mset B* ⟹ *all-occurrences A* = *all-occurrences B*› **for** *A B*
using *remdups-mset-set-mset-cong*[*of A B*]
unfolding *all-occurrences-def*
by *auto*

lemma *correct-occurence-list-mono-candsI*:
‹*correct-occurence-list Sa occs* (*cands*) *n* ⟹
correct-occurence-list Sa occs (*remove1-mset C cands*) *n*›
unfolding *correct-occurence-list-def*
by (*auto intro*: *inter-mset-empty-remove1-msetI*)

lemma *correct-occurence-list-mono-candsI2*:
‹*correct-occurence-list Sa occs* (*add-mset C cands*) *n* ⟹
correct-occurence-list Sa occs (*cands*) *n*›
unfolding *correct-occurence-list-def*
by *auto*

lemma *correct-occurence-list-size-mono*:
‹*correct-occurence-list x1h occs cands n* ⟹ *m* ≥ *n* ⟹
correct-occurence-list x1h occs cands m›
by (*auto simp*: *correct-occurence-list-def ball-conj-distrib dest*!: *multi-member-split*)

lemma *all-occurrences-remove-dups-atms*[*simp*]:
‹*set-mset* (*all-occurrences* (*mset-set* (*set-mset* (*all-init-atms-st* S_0))) *occs*) =
set-mset (*all-occurrences* (*all-init-atms-st* S_0) *occs*)›
unfolding *all-occurrences-def*
by (*cases occs*) *auto*

lemma *forward-subsumption-one-wl2-forward-subsumption-one-wl*:
fixes S_0 *C*
defines *G*: ‹*G* ≡ *mset* (*get-clauses-wl* S_0 ∝ *C*)›
assumes
‹(*C*, *E*) ∈ *nat-rel*› **and**
DG: ‹(*D*, *G*) ∈ *clause-hash-ref* (*all-init-atms-st* S_0)› **and**
occs: ‹*correct-occurence-list* S_0 *occs cands n*› **and**
n: ‹*n*≤ *size* (*get-clauses-wl* S_0 ∝ *C*)› **and**
C-occs: ‹*C* ∉# *all-occurrences* (*all-init-atms-st* S_0) *occs*› **and**
L: ‹*atm-of L* ∈# *all-init-atms-st* S_0› **and**
‹(S_0, T_0) ∈ *Id*›
‹(*cands*, *cands*′) ∈ *Id*›
shows ‹*forward-subsumption-one-wl2 C cands L occs D* S_0 ≤ ⇓
(*forward-subsumption-one-wl2-rel* S_0 *occs cands n C D*)
(*forward-subsumption-one-wl E cands*′ T_0)›
proof −
have ‹*cands* ∩# *mset* (*occ-list occs L*) = {#}›
using *occs L n* **unfolding** *correct-occurence-list-def*
by (*auto simp*: *all-occurrences-add-mset inter-mset-empty-distrib-right subset-mset.inf.commute*
dest!: *multi-member-split*[*of* ‹*atm-of L*›])
then have [*refine*]:
‹*RETURN* (*occ-list occs L*)
≤ ⇓ *list-mset-rel* (*SPEC* (*forward-subsumption-one-wl-select C cands* S_0))›
using *occs C-occs L n* **unfolding** *forward-subsumption-one-wl-select-def*
by (*auto simp*: *correct-occurence-list-def all-occurrences-add-mset*
forward-subsumption-one-select-def list-mset-rel-def br-def
dest!: *multi-member-split*
intro!: *RETURN-RES-refine*
intro: *le-trans*[*OF* - *n*])
have [*refine*]: ‹(*occ-list occs L*, *ys*) ∈ *list-mset-rel* ⟹
((*0*, *NONE*), *ys*, *NONE*) ∈ {(*n*, *z*). *z* = *mset* (*drop n* (*occ-list occs L*))} ×$_f$ *Id*› **for** *ys*
by (*auto simp*: *list-mset-rel-def br-def*)

define *ISABELLE-YOU-ARE-SO-STUPID* :: ‹*nat multiset* ⇒ -› **where**
‹*ISABELLE-YOU-ARE-SO-STUPID xs* = *SPEC* (λ*C'*. *C'* ∈# *xs*)› **for** *xs*
have *H*: ‹(*if s* ≠ *NONE then RETURN* ({#} :: *nat clause*) *else RETURN G*) =
RETURN ((*if s* ≠ *NONE then* ({#} :: *nat clause*) *else G*))› **for** *s*
by *auto*
have *forward-subsumption-one-wl-alt-def*:
‹*forward-subsumption-one-wl* = (λ*C cands* S_0. *do* {
ASSERT (*forward-subsumption-one-wl-pre C cands* S_0);
ys ← *SPEC* (*forward-subsumption-one-wl-select C cands* S_0);
let - = *size ys*;
(*xs*, *s*) ←
$WHILE_T$ *forward-subsumption-one-wl-inv* S_0 *C ys* (λ(*xs*, *s*). *xs* ≠ {#} ∧ *s* = *NONE*)
(λ(*xs*, *s*). *do* {
C' ←*ISABELLE-YOU-ARE-SO-STUPID xs*;
if C' ∉# *dom-m* (*get-clauses-wl* S_0)
then RETURN (*remove1-mset C' xs*, *s*)
else do {
s ← *subsume-clauses-match C' C* (*get-clauses-wl* S_0);
RETURN (*remove1-mset C' xs*, *s*)
}
})
(*ys*, *NONE*);
- ← *RETURN* (*if s* ≠ *NONE then* ({#} :: *nat clause*) *else G*);
let - = (*if is-strengthened s then* () *else* ());
S ← *subsume-or-strengthen-wl C s* S_0;
ASSERT
(*literals-are-*$\mathcal{L}_{in}$' *S* ∧
set-mset (*all-init-lits-of-wl S*) = *set-mset* (*all-init-lits-of-wl* S_0));
RETURN (*S*, *s* ≠ *NONE*)
})›
unfolding *forward-subsumption-one-wl-def ISABELLE-YOU-ARE-SO-STUPID-def bind-to-let-conv H*
by (*auto split*: *intro*!: *bind-cong*[*OF refl*] *ext*)

have *ISABELLE-YOU-ARE-SO-STUPID*: ‹(*x*, *x'*) ∈ {(*n*, *z*). *z* = *mset* (*drop n* (*occ-list occs L*))} ×$_f$ *Id* ⟹
case x of (*i*, *s*) ⇒ *i* < *length* (*occ-list occs L*) ∧ *s* = *NONE* ⟹
x' = (*x1*, *x2*) ⟹
x = (*x1a*, *x2a*) ⟹
SPEC (λ*c*. (*c*, *occ-list-at occs L x1a*) ∈ *nat-rel*)
≤ ⇓ {(*a*,*b*). *a* = *b* ∧ *b* = *occ-list-at occs L x1a*}
(*ISABELLE-YOU-ARE-SO-STUPID x1*)› **for** *x1 x1a x x' x2 x2a*
by (*cases occs*, *auto simp*: *ISABELLE-YOU-ARE-SO-STUPID-def occ-list-at-def occ-list-def*
RES-refine
intro!: *in-set-dropI RES-refine*)
have *H*: ‹*is-strengthened x2a* ⟹
(*xa*, ()) ∈ *R* ⟹
(*xa*, *if is-strengthened x2 then* () *else* ()) ∈ (*if* ¬*is-strengthened x2a then* {(*a*,*b*). *a* = *occs*} *else R*)›
for *xa x2 x2a R*
by *auto*
have *itself*: ‹*subsume-or-strengthen-wl C s S* ≤⇓{(*x*,*y*). *x* = *y* ∧
get-trail-wl x = *get-trail-wl S* ∧ (¬*is-subsumed s* ⟶ *C*∈#*dom-m* (*get-clauses-wl x*)) ∧
(*s* = *NONE* ⟶ *x* = *S*) ∧
(*dom-m* (*get-clauses-wl x*) ⊆# *dom-m* (*get-clauses-wl S*)) ∧
(∀ *Ca*∈#*dom-m* (*get-clauses-wl x*). *Ca*∈#*dom-m* (*get-clauses-wl S*) ∧ *length* (*get-clauses-wl x* ∝ *Ca*) ≤ *length* (*get-clauses-wl S* ∝ *Ca*)) ∧

```
   (∀ Ca∈#dom-m (get-clauses-wl x). Ca∈#dom-m (get-clauses-wl S) ∧ set (get-clauses-wl x ∝ Ca) ⊆
set (get-clauses-wl S ∝ Ca))
   } b› if b: ‹b = subsume-or-strengthen-wl C s S› and
   S: ‹forward-subsumption-one-wl-pre C ys S›
   for a b s S ys
  proof −
   have [simp]: ‹x1 ∈# dom-m b ⟹ {#mset (fst x). x ∈# ran-m (fmupd x1 (b ∝ x1, True) b)#} =
    {#mset (fst x). x ∈# ran-m b#}› for b x1
    by (auto simp: ran-m-def dest!: multi-member-split intro: image-mset-cong)
   have [simp]:
   ‹get-conflict-wl S = None ⟹ (get-trail-wl S, get-clauses-wl S, None, IsaSAT-Setup.get-unkept-unit-init-clss-wl
S,
      IsaSAT-Setup.get-unkept-unit-learned-clss-wl S, IsaSAT-Setup.get-kept-unit-init-clss-wl S,
      IsaSAT-Setup.get-kept-unit-learned-clss-wl S, get-subsumed-init-clauses-wl S,
      get-subsumed-learned-clauses-wl S, get-init-clauses0-wl S, get-learned-clauses0-wl S,
      literals-to-update-wl S, get-watched-wl S) = S› for S
    by (cases S) auto
   have [simp]: ‹C ∉# remove1-mset C (dom-m S)› for C S
    using distinct-mset-dom[of ‹S›]
    by (cases ‹C ∈# dom-m S›) (auto dest!: multi-member-split)
   have H: ‹mset Ea = remove1-mset (− x21) (mset (get-clauses-wl S ∝ C)) ⟹
       length (get-clauses-wl S ∝ C) = length (get-clauses-wl S ∝ x22) ⟹
    length Ea ≤ length (get-clauses-wl S ∝ x22)›for Ea x21 S C x22
    by (metis size-Diff1-le size-mset)
   have H2: ‹mset Ea = remove1-mset (− x21) (mset (get-clauses-wl S ∝ C)) ⟹
    x ∈ set Ea ⟹ x ∈ set (get-clauses-wl S ∝ C)›for Ea x21 S C x22 x
    by (metis in-diffD in-multiset-in-set)
   show ?thesis
    unfolding subsume-or-strengthen-wl-def b strengthen-clause-wl-def case-wl-split
     state-wl-recompose
    apply refine-vcg
    subgoal
     using S unfolding forward-subsumption-one-wl-pre-def forward-subsumption-one-pre-def
      subsume-or-strengthen-wl-pre-def subsume-or-strengthen-pre-def
     apply −
     apply normalize-goal+
     subgoal for T U
      apply (simp only: split: subsumption.splits)
      apply (intro conjI)
      subgoal
       by refine-vcg
        (auto split: subsumption.splits simp: state-wl-recompose)
      subgoal
       by (auto split: subsumption.splits simp: state-wl-recompose)
      subgoal
       by (auto split: subsumption.splits simp: state-wl-recompose)
      subgoal
       by refine-vcg
        (auto split: subsumption.splits simp: state-wl-recompose
        intro: H H2)
      subgoal
       by (auto split: subsumption.splits simp: state-wl-recompose)
      done
    done
  done
 qed
```

have *mop-ch-remove-all2*:
‹*mop-ch-remove-all D C* ≤ *SPEC*(λ*c*. (*c*, {#}) ∈ *clause-hash-ref* $\mathcal{A}$)›
if ‹(*C*, *D*) ∈ *clause-hash-ref* $\mathcal{A}$› **for** *C D* $\mathcal{A}$
using *that* **unfolding** *mop-ch-remove-all-def*
apply *refine-vcg*
subgoal by (*auto simp*: *clause-hash-ref-def ch-remove-all-def*)
subgoal by (*auto simp*: *clause-hash-ref-def ch-remove-all-def dest*!: *multi-member-split*)
subgoal by (*auto simp*: *clause-hash-ref-def ch-remove-all-def*)
done
have *K*: ‹(*xa*, {#}) ∈ *clause-hash-ref* (*all-init-atms-st* S_0) ⟹ *x2* ≠ *NONE* ⟹
(*xa*, *if x2* ≠ *NONE then* {#} *else G*) ∈
{(*xa*, *b*). *if x2* ≠ *NONE then b* = {#} ∧ (*xa*, *b*)∈ *clause-hash-ref* (*all-init-atms-st* S_0)
else (*xa*,*b*) = (*D*,*G*)}› **for** *xa x2*
by *auto*

have *correct-occurence-list-cong*:
‹*correct-occurence-list T occs ys* (*length* (*get-clauses-wl* S_0 ∝ *C*)) ⟹
set-mset (*all-init-atms-st T*) = *set-mset* (*all-init-atms-st U*) ⟹
get-trail-wl T = *get-trail-wl U* ⟹
dom-m (*get-clauses-wl U*) ⊆# *dom-m* (*get-clauses-wl T*) ⟹
(∀ *Ca*∈#*dom-m* (*get-clauses-wl U*). *Ca*∈#*dom-m* (*get-clauses-wl T*) ∧ *length* (*get-clauses-wl U* ∝ *Ca*) ≤ *length* (*get-clauses-wl T* ∝ *Ca*)) ⟹
(∀ *Ca*∈#*dom-m* (*get-clauses-wl U*). *Ca*∈#*dom-m* (*get-clauses-wl T*) ∧ *set* (*get-clauses-wl U* ∝ *Ca*) ⊆ *set* (*get-clauses-wl T* ∝ *Ca*)) ⟹
correct-occurence-list U occs ys (*length* (*get-clauses-wl* S_0 ∝ *C*))› **for** *T U occs ys*
using *remdups-mset-set-mset-cong*[*of* ‹*all-init-atms-st T*› ‹*all-init-atms-st U*›]
all-occurrences-cong[*of* ‹*all-init-atms-st T*› ‹*all-init-atms-st U*›] *n*
unfolding *correct-occurence-list-def*
apply (*cases occs*)
apply (*auto*)
apply (*meson mset-subset-eqD order-trans*)
apply (*meson mset-subset-eqD subsetD*)
done

have *eq*: ‹T_0 = S_0› ‹*E* = *C*› ‹*cands*′ = *cands*›
using *assms* **by** *auto*
show *?thesis*
unfolding *forward-subsumption-one-wl2-def eq*
forward-subsumption-one-wl-alt-def
apply (*refine-vcg mop-occ-list-at*[*THEN order-trans*] *DG mop-ch-remove-all2 occs*
subsume-clauses-match2-subsume-clauses-match maybe-push-to-occs-list2 DG[*unfolded G*])
subgoal
using *assms* $\mathcal{L}_{all}$*-all-init-atms*(*2*) *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$
unfolding *forward-subsumption-one-wl2-pre-def* **by** *blast*
subgoal
using *assms occs*
by (*auto simp*: *correct-occurence-list-def*)
subgoal for *ys x x*′
unfolding *forward-subsumption-one-wl2-inv-def*
by (*cases x*′) (*auto simp*: *list-mset-rel-def br-def*)
subgoal by *auto*
subgoal using *occs L* **unfolding** *occ-list-at-pre-def correct-occurence-list-def*
by (*cases occs*, *auto simp*: *occ-list-def*)
apply (*rule ISABELLE-YOU-ARE-SO-STUPID*)
apply *assumption*+
subgoal by (*cases occs*, *auto simp*: *occ-list-at-def occ-list-def in-set-dropI*)

subgoal by (*auto simp flip*: *Cons-nth-drop-Suc occ-list-at-def*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *G* **by** *auto*
subgoal unfolding *forward-subsumption-one-wl-pre-def* **by** *blast*
subgoal by (*auto simp flip*: *Cons-nth-drop-Suc occ-list-at-def*)
apply (*rule K*; *assumption?*)
subgoal by *auto*
subgoal by *auto*
subgoal unfolding *forward-subsumption-one-wl-pre-def forward-subsumption-one-pre-def*
by *normalize-goal+ simp*
subgoal unfolding *forward-subsumption-one-wl-pre-def forward-subsumption-one-pre-def*
by *normalize-goal+ simp*
subgoal unfolding *forward-subsumption-one-wl-pre-def* **by** *blast*
subgoal unfolding *forward-subsumption-one-wl-pre-def forward-subsumption-one-pre-def*
by *normalize-goal+ simp*
subgoal using *C-occs occs* **unfolding** *correct-occurence-list-def* **by** *auto*
apply (*rule H*; *assumption*)
subgoal by *auto*
apply (*rule itself*)
subgoal by *auto*
apply *assumption*
subgoal for *ys x x' x1 x2 x1a x2a Da - occsa S Sa*
using *occs n*
clause-hash-ref-cong[*of* ‹*all-init-atms-st* S_0› ‹*atm-of* ‘# *all-init-lits-of-wl Sa*›]
all-occurrences-cong[*of* ‹*all-init-atms-st* S_0› ‹*atm-of* ‘# *all-init-lits-of-wl Sa*›]
by (*auto simp*: *forward-subsumption-one-wl2-rel-def all-init-atms-st-alt-def*
correct-occurence-list-cong[*of* S_0]
intro: *correct-occurence-list-size-mono intro*!: *correct-occurence-list-mono-candsI*
simp del: *multiset.set-map*
intro: *correct-occurence-list-cong*
split: *if-splits*)
done
qed

definition *try-to-forward-subsume-wl2* :: ‹*nat* ⇒ *occurences* ⇒ *nat multiset* ⇒ *clause-hash* ⇒ *nat multiset* ⇒ *nat twl-st-wl* ⇒ (*occurences* × *clause-hash* × *nat multiset* × *nat twl-st-wl*) *nres*› **where**
‹*try-to-forward-subsume-wl2 C occs cands D shrunken S* = *do* {
ASSERT (*try-to-forward-subsume-wl2-pre C cands shrunken S*);
let n = *length* (*get-clauses-wl S* ∝ *C*);
let n = *2* ∗ *n*;
ebreak ← *RES* {-::*bool*. *True*};
(-, *changed*, -, *occs*, *D*, *S*) ← $WHILE_T$ *try-to-forward-subsume-wl2-inv S cands C*
(λ(*i*, *changed*, *break*, *occs*, *D*, *S*). ¬*break* ∧ *i* < *n*)
(λ(*i*, *changed*, *break*, *occs*, *D*, *S*). *do* {
L ← *mop-clauses-at* (*get-clauses-wl S*) *C* (*i div 2*);
let L = (*if i mod 2* = *0 then L else* − *L*);
(*S*, *subs*, *occs*, *D*) ← *forward-subsumption-one-wl2 C cands L occs D S*;
ebreak ← *RES* {-::*bool*. *True*};
RETURN (*i*+*1*, *subs*, *subs* ∨ *ebreak*, *occs*, *D*, *S*)
})
(*0*, *False*, *ebreak*, *occs*, *D*, *S*);
ASSERT (¬*changed* ⟶ *C* ∈# *dom-m* (*get-clauses-wl S*));
D ← (*if* ¬*changed then mop-ch-remove-all* (*mset* (*get-clauses-wl S* ∝ *C*)) *D else RETURN D*);
add-to-shrunken ← *RES* (*UNIV* :: *bool set*);

let shrunken = (if add-to-shrunken then add-mset C shrunken else shrunken);
RETURN (occs, D, shrunken, S)
}⟩

definition *try-to-forward-subsume-wl2-pre0* :: ⟨-⟩ **where**
⟨*try-to-forward-subsume-wl2-pre0 G C occs cands D* S_0 *n* ⟷
correct-occurence-list S_0 *occs cands n* ∧
n≤ *length* (*get-clauses-wl* S_0 ∝ *C*) ∧
C ∉# *all-occurrences* (*all-init-atms-st* S_0) *occs* ∧
distinct-mset cands ∧
C ∈# *dom-m* (*get-clauses-wl* S_0) ∧
G = *mset* (*get-clauses-wl* S_0 ∝ *C*)⟩

definition *try-to-forward-subsume-wl-rel* :: ⟨-⟩ **where**
⟨*try-to-forward-subsume-wl-rel* S_0 *C cands occs n shrunken* ≡
{((*occs'*, *D'*, *shrunken'*, *T*), *U*). (*T*, *U*) ∈ *Id* ∧ (*D'*, {#}) ∈ *clause-hash-ref* (*all-init-atms-st T*) ∧
correct-occurence-list U occs' (*remove1-mset C cands*) (*max* (*length* (*get-clauses-wl* S_0 ∝ *C*)) *n*) ∧
shrunken' ⊆# *add-mset C shrunken* ∧
all-occurrences (*all-init-atms-st U*) *occs'* ⊆# *add-mset C* (*all-occurrences* (*all-init-atms-st* S_0) *occs*)}⟩

lemma *try-to-forward-subsume-wl2-try-to-forward-subsume-wl*:
assumes ⟨(*C*, *E*) ∈ *nat-rel*⟩ **and**
DG: ⟨(*D*, *G*) ∈ *clause-hash-ref* (*all-init-atms-st* T_0)⟩ **and**
pre: ⟨*try-to-forward-subsume-wl2-pre0 G C occs cands D* S_0 *n*⟩ **and**
⟨(S_0, T_0) ∈ *Id*⟩
⟨(*cands*, *cands'*) ∈ *Id*⟩
shows ⟨*try-to-forward-subsume-wl2 C occs cands D shrunken* S_0 ≤
⇓(*try-to-forward-subsume-wl-rel* S_0 *C cands occs n shrunken*)
(*try-to-forward-subsume-wl E cands'* T_0)⟩
proof −
have *eq*: ⟨T_0 = S_0⟩ ⟨*E* = *C*⟩ ⟨*cands'* = *cands*⟩
using *assms* **by** *auto*
have
occs: ⟨*correct-occurence-list* S_0 *occs cands n*⟩ **and**
n: ⟨*n*≤ *length* (*get-clauses-wl* S_0 ∝ *C*)⟩ **and**
C-occs: ⟨*C* ∉# *all-occurrences* (*all-init-atms-st* T_0) *occs*⟩ **and**
⟨*distinct-mset cands*⟩ **and**
G: ⟨*G* = *mset* (*get-clauses-wl* S_0 ∝ *C*)⟩
using *pre* **unfolding** *try-to-forward-subsume-wl2-pre0-def eq*
by *auto*
have *try-to-forward-subsume-wl-alt-def*:
⟨*try-to-forward-subsume-wl C cands S* = *do* {
ASSERT (*try-to-forward-subsume-wl-pre C cands S*);
n ← *RES* {-::*nat*. *True*};
ebreak ← *RES* {-::*bool*. *True*};
(-, -, *S*) ← $WHILE_T$ *try-to-forward-subsume-wl-inv S cands C*
(λ(*i*, *break*, *S*). ¬*break* ∧ *i* < *n*)
(λ(*i*, *break*, *S*). *do* {
- ← *SPEC* (λ*L* :: *nat literal*. *True*);
(*S*, *subs*) ← *forward-subsumption-one-wl C cands S*;
ebreak ← *RES* {-::*bool*. *True*};
RETURN (*i*+*1*, *subs* ∨ *ebreak*, *S*)
})
(*0*, *ebreak*, *S*);
- ← *RETURN* ({#} :: *nat clause*);

RETURN S
}
› **for** *S*
unfolding *try-to-forward-subsume-wl-def nres-monad3*
by *auto*

have [*refine*]: ‹(*ebreak*, *ebreaka*) ∈ *bool-rel* ⟹ *try-to-forward-subsume-wl-pre C cands* S_0 ⟹
((*0*, *False*, *ebreak*, *occs*, *D*, S_0), *0*, *ebreaka*, S_0) ∈
{((*p*, *changed*, *ebreak*, *occs′*, *D′*, *U*), (*q*, *ebreak′*, *V*)). (*p*,*q*)∈*nat-rel* ∧ (¬*changed* ⟶ *C*∈#*dom-m* (*get-clauses-wl U*)) ∧
(¬*changed* ⟶ (*D′*, *mset* (*get-clauses-wl U* ∝ *C*)) ∈ *clause-hash-ref* (*all-init-atms-st U*)) ∧
(*changed* ⟶ *ebreak* ∧ (*D′*, {#}) ∈ *clause-hash-ref* (*all-init-atms-st U*)) ∧
(*ebreak*, *ebreak′*) ∈ *bool-rel* ∧
(¬*changed* ⟶ *correct-occurence-list U occs′ cands n* ∧ *occs′* = *occs* ∧ *get-clauses-wl U* ∝ *C* = *get-clauses-wl* S_0 ∝ *C* ∧
all-occurrences (*all-init-atms-st V*) *occs′* = *all-occurrences* (*all-init-atms-st* S_0) *occs*) ∧
(*changed* ⟶ *correct-occurence-list U occs′* (*remove1-mset C cands*) (*max* (*length* (*get-clauses-wl* S_0 ∝ *C*)) *n*) ∧
all-occurrences (*all-init-atms-st V*) *occs′* ⊆# *add-mset C* (*all-occurrences* (*all-init-atms-st* S_0) *occs*)) ∧
U = *V*}› (**is** ‹- ⟹ - ⟹ - ∈ *?R*›)
for *ebreak ebreaka*
using *assms* **unfolding** *try-to-forward-subsume-wl-pre-def try-to-forward-subsume-pre-def*
by − (*normalize-goal+*; *simp add*: *try-to-forward-subsume-wl2-pre0-def try-to-forward-subsume-wl-pre-def*)
have *try-to-forward-subsume-wl2-invD*:
‹*try-to-forward-subsume-wl2-inv* S_0 *cands C x* ⟹ *set-mset* (*all-init-atms-st* (*snd* (*snd* (*snd* (*snd* (*snd x*)))))) = *set-mset*
(*all-init-atms-st* S_0)› **for** *x*
unfolding *try-to-forward-subsume-wl2-inv-def*
try-to-forward-subsume-wl-inv-def
try-to-forward-subsume-inv-def case-prod-beta
apply *normalize-goal+*
apply (*frule rtranclp-cdcl-twl-inprocessing-l-all-init-lits-of-l*)
by (*simp add*: *all-init-atms-st-alt-def*)
have *H*: ‹*C* ∈# *dom-m* (*get-clauses-wl x2a*) ⟹ *x1* < *2* ∗ *length* (*get-clauses-wl x2a* ∝ *C*) ⟹
atm-of (*get-clauses-wl x2a* ∝ *C* ! (*x1 div 2*))
∈ *atm-of* ‘
(*set-mset* (*all-lits-of-m* (*mset* (*get-clauses-wl x2a* ∝ *C*))))› **for** *x1 x2a*
by (*rule imageI*)
(*auto intro*!: *in-clause-in-all-lits-of-m nth-mem*)

have *K*: ‹*try-to-forward-subsume-wl-inv* S_0 *cands C* (*x1*, *False*, *x2a*) ⟹
x1 < *2* ∗ *length* (*get-clauses-wl x2a* ∝ *C*) ⟹
atm-of (*get-clauses-wl x2a* ∝ *C* ! (*x1 div 2*)) ∈# *all-init-atms-st x2a*› **for** *x1 x2a*
using *H*[*of x2a x1*]
unfolding *try-to-forward-subsume-wl-inv-def prod.simps try-to-forward-subsume-inv-def*
literals-are-$\mathcal{L}_{in}$*′-def* **apply** −
apply *normalize-goal+*
apply (*frule rtranclp-cdcl-twl-inprocessing-l-all-init-lits-of-l*,
frule rtranclp-cdcl-twl-inprocessing-l-all-learned-lits-of-l)
apply (*simp add*: *all-init-atms-st-alt-def*)
apply (*auto simp*: *all-init-atms-st-def ran-m-def all-init-atms-alt-def all-init-atms-def*
all-init-lits-of-wl-def
all-init-lits-def all-lits-of-mm-add-mset all-learned-lits-of-wl-def
dest!: *multi-member-split*[*of C*]

simp del: *all-init-atms-def*[*symmetric*] *all-init-lits-of-wl-def*[*symmetric*])
by *blast*

have *K0*: ‹*try-to-forward-subsume-wl-inv* S_0 *cands C* (*x1*, *False*, *x2a*) ⟹
atm-of ‘ *set* (*get-clauses-wl x2a* ∝ *C*) ⊆ *set-mset* (*all-init-atms-st x2a*)› **for** *x1 x2a*
unfolding *try-to-forward-subsume-wl-inv-def prod.simps try-to-forward-subsume-inv-def*
literals-are-$\mathcal{L}_{in}$*'-def* **apply** −
apply *normalize-goal*+
apply (*frule rtranclp-cdcl-twl-inprocessing-l-all-init-lits-of-l*,
frule rtranclp-cdcl-twl-inprocessing-l-all-learned-lits-of-l)
apply (*simp add*: *all-init-atms-st-alt-def*)
apply (*auto simp*: *all-init-atms-st-def ran-m-def all-init-atms-alt-def all-init-atms-def*
all-init-lits-of-wl-def
all-init-lits-def all-lits-of-mm-add-mset all-learned-lits-of-wl-def in-clause-in-all-lits-of-m
dest!: *multi-member-split*[*of C*]
simp del: *all-init-atms-def*[*symmetric*] *all-init-lits-of-wl-def*[*symmetric*])
by (*meson image-eqI in-clause-in-all-lits-of-m in-multiset-in-set subsetD*)

have *mop-ch-remove-all2*:
‹(*x*, *x'*) ∈ *?R* ⟹
x2 = (*x1a*, *x2a*) ⟹
x' = (*x1*, *x2*) ⟹
x2e = (*x1f*, *x2f*) ⟹
x2d = (*x1e*, *x2e*) ⟹
x2c = (*x1d*, *x2d*) ⟹
x2b = (*x1c*, *x2c*) ⟹
x = (*x1b*, *x2b*) ⟹ ¬*x1c* ⟹
mop-ch-remove-all (*mset* (*get-clauses-wl x2f* ∝ *C*)) *x1f* ≤ *SPEC*(λ*c*. (*c*, {#}) ∈ {(*a*,*b*). *b* = {#} ∧
(*a*,*b*) ∈ *clause-hash-ref* (*all-init-atms-st x2f*)})›
for *na ebreak ebreaka x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f*
unfolding *mop-ch-remove-all-def*
apply *refine-vcg*
subgoal
using *DG G*
by (*auto simp*: *clause-hash-ref-def ch-remove-all-def*)
subgoal by (*auto 5 3 simp add*: *clause-hash-ref-def*)
subgoal
by (*auto simp*: *clause-hash-ref-def ch-remove-all-def*)
done
show *?thesis*
unfolding *try-to-forward-subsume-wl2-def try-to-forward-subsume-wl-alt-def eq mop-clauses-at-def*
nres-monad3 bind-to-let-conv try-to-forward-subsume-wl-rel-def
apply (*subst Let-def*[*of* ‹*get-clauses-wl* - ∝ - ! -›])
apply (*subst Let-def*[*of* ‹*length* (*get-clauses-wl* - ∝ -)›])
apply (*refine-vcg DG*[*unfolded G*]
forward-subsumption-one-wl2-forward-subsumption-one-wl[**where** *n*=*n*])
subgoal using *assms* **unfolding** *try-to-forward-subsume-wl2-pre-def try-to-forward-subsume-wl2-pre0-def*
by *fast*
subgoal by *fast*
subgoal using *G* **unfolding** *try-to-forward-subsume-wl2-inv-def* **by** *auto*
subgoal by *auto*
subgoal unfolding *try-to-forward-subsume-wl-inv-def case-prod-beta try-to-forward-subsume-inv-def*
by *normalize-goal*+ *auto*
subgoal by *auto*
subgoal by *fast*
subgoal by *fast*

subgoal by *fast*
subgoal by *auto*
subgoal using *n* **by** *auto*
subgoal using *C-occs* **by** (*auto dest*: *all-occurrences-cong*[*OF try-to-forward-subsume-wl2-invD*] *simp*: *eq*)
subgoal for *a ebreak ebreaka x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f*
using *K*[*of x1 x2a*] **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal for *na ebreak ebreaka x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f uu- xa x'a x1g x2g x1h x2h x1i x2i*
x1j x2j ebreakb ebreakc
apply (*frule all-occurrences-cong*[*OF try-to-forward-subsume-wl2-invD*])
apply (*clarsimp simp add*: *forward-subsumption-one-wl2-rel-def*)
apply (*auto dest*:
intro: *correct-occurence-list-size-mono*[*of - - - - ‹*(*max* (*length* (*get-clauses-wl* S_0 ∝ *C*)) *n*)*›*]
simp: *clause-hash-ref-cong*[*of ‹all-init-atms-st x1g›*]
all-occurrences-cong[*of ‹all-init-atms-st x1g› ‹all-init-atms-st x2a›*]
forward-subsumption-one-wl2-rel-def
simp:)
done
subgoal
apply *simp*
by *auto*
apply (*rule mop-ch-remove-all2*; *assumption*)
subgoal by *auto*
subgoal for *na ebreak ebreaka x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f*
using *n* **by** (*auto simp*: *eq intro*: *correct-occurence-list-size-mono correct-occurence-list-mono-candsI*)
done
qed

definition *populate-occs-spec* **where**
‹populate-occs-spec S = (λ(*occs*, *cands*).
all-occurrences (*all-init-atms-st S*) *occs* + *mset cands* ⊆# *dom-m* (*get-clauses-wl S*) ∧
distinct cands ∧ *sorted-wrt* (λ*a b*. *length* (*get-clauses-wl S* ∝ *a*) ≤ *length* (*get-clauses-wl S* ∝ *b*)) *cands* ∧
correct-occurence-list S occs (*mset cands*) *2* ∧
(∀ *C*∈*set cands*. ∀ *L*∈ *set* (*get-clauses-wl S* ∝ *C*). *undefined-lit* (*get-trail-wl S*) *L*) ∧
(∀ *C*∈*set cands*. *length* (*get-clauses-wl S* ∝ *C*) > *2*))*›*

definition *all-lit-clause-unset* :: *‹-›* **where**
‹all-lit-clause-unset S C = *do* {
ASSERT (*forward-subsumption-all-wl-pre S*);
let b = *C* ∈# *dom-m* (*get-clauses-wl S*);
if ¬*b then RETURN False*
else do {
let n = *length* (*get-clauses-wl S* ∝ *C*);
(*i*, *all-undef*) ← *WHILE*$_T$ (λ(*i*, *all-undef*). *i* < *n* ∧ *all-undef*)
(λ(*i*, -). *do* {
ASSERT (*i*<*n*);
L ← *mop-clauses-at* (*get-clauses-wl S*) *C i*;
ASSERT (*L* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st S*));
val ← *mop-polarity-wl S L*;
RETURN (*i*+*1*, *val* = *None*)
}) (*0*, *True*);

```
    other ← SPEC (λ-. True);
    RETURN (all-undef ∧ other)
  }
 }›

lemma forward-subsumption-all-wl-preD: ‹forward-subsumption-all-wl-pre S ⟹ no-dup (get-trail-wl
S)›
  unfolding forward-subsumption-all-wl-pre-def
    forward-subsumption-all-pre-def twl-list-invs-def
    twl-struct-invs-def pcdcl-all-struct-invs-def
    cdcl_W-restart-mset.cdcl_W-all-struct-inv-def
    cdcl_W-restart-mset.cdcl_W-M-level-inv-def
  by normalize-goal+ simp

lemma all-lit-clause-unset-spec:
  ‹forward-subsumption-all-wl-pre S ⟹
  all-lit-clause-unset S C ≤ SPEC (λb. b ⟶ C∈#dom-m (get-clauses-wl S) ∧ (∀ L∈set (get-clauses-wl
S ∝ C). undefined-lit (get-trail-wl S) L))›
  unfolding all-lit-clause-unset-def mop-clauses-at-def nres-monad3 mop-polarity-wl-def
  apply (refine-vcg WHILET-rule[where R=‹measure (λ(i,-). length (get-clauses-wl S ∝ C) −i)› and
    I=‹λ(i, all-undef). i ≤ length (get-clauses-wl S ∝ C) ∧
    (all-undef ⟷ (∀ L∈set (take i (get-clauses-wl S ∝ C)). undefined-lit (get-trail-wl S) L))›])
  subgoal by fast
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal by auto
  subgoal
    using literals-are-\mathcal{L}_in'-literals-are-\mathcal{L}_in-iff(3)[of S, THEN \mathcal{L}_all-cong]
    unfolding forward-subsumption-all-wl-pre-def
      forward-subsumption-all-pre-def apply −
    by normalize-goal+
     (simp add: \mathcal{L}_all-all-atms)
  subgoal by auto
  subgoal by (auto dest: forward-subsumption-all-wl-preD)
  subgoal by (auto dest: forward-subsumption-all-wl-preD)
  subgoal by (auto simp: take-Suc-conv-app-nth polarity-def split: if-splits)
  subgoal by auto
  subgoal by auto
  subgoal by auto
  done


definition populate-occs :: ‹nat twl-st-wl ⇒ - nres› where
  ‹populate-occs S = do {
    ASSERT (forward-subsumption-all-wl-pre S);
    xs ← SPEC (λxs. distinct xs);
    let n = size xs;
    occs ← mop-occ-list-create (set-mset (all-init-atms-st S));
    let cands = [];
    (-, occs, cands) ← WHILE_T^{populate-occs-inv S xs} (λ(i, occs, cands). i < n)
    (λ(i, occs, cands). do {
      let C = xs ! i;
      ASSERT (C ∉ set cands);
```

all-undef ← *all-lit-clause-unset S C*;
if ¬*all-undef then*
RETURN (*i*+*1*, *occs*, *cands*)
else do {
ASSERT(*C* ∈# *dom-m* (*get-clauses-wl S*));
let le = *length* (*get-clauses-wl S* ∝ *C*) *in*
if le = *2 then do* {
occs ← *push-to-occs-list2 C S occs*;
RETURN (*i*+*1*, *occs*, *cands*)
}
else do {
ASSERT(*C* ∈# *dom-m* (*get-clauses-wl S*));
cand ← *RES* (*UNIV*::*bool set*);
if cand then RETURN (*i*+*1*, *occs*, *cands* @ [*C*])
else RETURN (*i*+*1*, *occs*, *cands*)
}
}
}
)
(*0*, *occs*, *cands*);
ASSERT (∀ *C*∈*set cands*. *C*∈#*dom-m*(*get-clauses-wl S*) ∧ *C*∈*set xs*∧ *length* (*get-clauses-wl S* ∝ *C*) > *2*);
cands ← *SPEC* (λ*cands*′. *mset cands*′ = *mset cands* ∧ *sorted-wrt* (λ*a b*. *length* (*get-clauses-wl S* ∝ *a*) ≤ *length* (*get-clauses-wl S* ∝ *b*)) *cands*′);
RETURN (*occs*, *cands*)
}›

lemma *forward-subsumption-all-wl-pre-length-ge2*:
‹*forward-subsumption-all-wl-pre S* ⟹ *C* ∈#*dom-m* (*get-clauses-wl S*) ⟹ *length* (*get-clauses-wl S* ∝ *C*) ≥ *2*›
unfolding *forward-subsumption-all-wl-pre-def forward-subsumption-all-pre-def*
twl-struct-invs-def twl-st-inv-alt-def
by *normalize-goal*+
simp

lemma *populate-occs-populate-occs-spec*:
assumes ‹*literals-are-*$\mathcal{L}_{in}$′ *S*› ‹*forward-subsumption-all-wl-pre S*›
shows ‹*populate-occs S* ≤ ⇓*Id* (*SPEC*(*populate-occs-spec S*))›
proof −
have [*refine*]: ‹*distinct xs* ⟹*wf* (*measure* (λ(*i*, *occs*, *cands*). *length xs* − *i*))› **for** *xs*
by *auto*
have *correct-occurence-list*: ‹⋀*x xa s a b aa ba*.
distinct x ⟹
populate-occs-inv S x s ⟹
s = (*a*, *b*) ⟹
b = (*aa*, *ba*) ⟹
correct-occurence-list S aa (*mset* (*drop a x*)) *2*›
by (*auto simp*: *populate-occs-inv-def*)
have *G0*: ‹*A* ⊆# *B* ⟹ *a* ∉# *A* ⟹ *a* ∈# *B* ⟹ *add-mset a A* ⊆# *B*› **for** *A B* :: ‹*nat multiset*› **and** *a*
by (*metis add-mset-remove-trivial-eq insert-subset-eq-iff subset-add-mset-notin-subset-mset*)
have *G*: ‹*distinct x* ⟹
a < *length x* ⟹
all-occurrences (*all-init-atms-st S*) *aa* + *mset ba* ⊆# *mset* (*take a x*) ⟹
x ! *a* ∈# *dom-m* (*get-clauses-wl S*) ⟹
all-occurrences (*all-init-atms-st S*) *xc* = *add-mset* (*x* ! *a*) (*all-occurrences* (*all-init-atms-st S*) *aa*)

$\Longrightarrow$
all-occurrences (*all-init-atms-st S*) *aa* + *mset ba* $\subseteq\#$ *dom-m* (*get-clauses-wl S*) $\Longrightarrow$
distinct ba $\Longrightarrow$
correct-occurence-list S aa (*add-mset* (*x* ! *a*) (*mset* (*drop* (*Suc a*) *x*))) *2* $\Longrightarrow$
add-mset (*x* ! *a*) (*all-occurrences* (*all-init-atms-st S*) *aa* + *mset ba*) $\subseteq\#$ *dom-m* (*get-clauses-wl S*)›
for *x xa s a b aa ba xb xc*
apply (*rule G0*)
apply *auto*
apply (*meson correct-occurence-list-def disjunct-not-in union-single-eq-member*)
by (*meson distinct-in-set-take-iff in-multiset-in-set less-Suc-eq mset-le-decr-left2 mset-subset-eqD not-less-eq*)
have *G'*: ‹*distinct x* $\Longrightarrow$
a < *length x* $\Longrightarrow$
all-occurrences (*all-init-atms-st S*) *aa* + *mset ba* $\subseteq\#$ *mset* (*take a x*) $\Longrightarrow$
x ! *a* $\in\#$ *dom-m* (*get-clauses-wl S*) $\Longrightarrow$
all-occurrences (*all-init-atms-st S*) *aa* + *mset ba* $\subseteq\#$ *dom-m* (*get-clauses-wl S*) $\Longrightarrow$
distinct ba $\Longrightarrow$
correct-occurence-list S aa (*add-mset* (*x* ! *a*) (*mset* (*drop* (*Suc a*) *x*))) *2* $\Longrightarrow$
add-mset (*x* ! *a*) (*all-occurrences* (*all-init-atms-st S*) *aa* + *mset ba*) $\subseteq\#$ *dom-m* (*get-clauses-wl S*)›
‹*distinct x* $\Longrightarrow$
a < *length x* $\Longrightarrow$
all-occurrences (*all-init-atms-st S*) *aa* + *mset ba* $\subseteq\#$ *mset* (*take a x*) $\Longrightarrow$
x ! *a* $\in\#$ *dom-m* (*get-clauses-wl S*) $\Longrightarrow$
all-occurrences (*all-init-atms-st S*) *aa* + *mset ba* $\subseteq\#$ *dom-m* (*get-clauses-wl S*) $\Longrightarrow$
distinct ba $\Longrightarrow$
correct-occurence-list S aa (*add-mset* (*x* ! *a*) (*mset* (*drop* (*Suc a*) *x*))) *2* $\Longrightarrow$ *x* ! *a* $\in$ *set ba* $\Longrightarrow$ *False*›
for *x xa s a b aa ba xb xc*
apply (*rule G0*)
apply (*auto simp add*: *correct-occurence-list-def*)
by (*smt* (*verit*, *ccfv-threshold*) *distinct-in-set-take-iff in-multiset-in-set nat-in-between-eq*(*2*) *not-less-eq set-mset-mono subsetD union-iff*)+

show *?thesis*
unfolding *populate-occs-def Let-def*
apply (*refine-vcg mop-occ-list-create*[*THEN order-trans*] *push-to-occs-list2 all-lit-clause-unset-spec*)
subgoal using *assms*(*2*) **by** *fast*
apply *assumption*
subgoal unfolding *populate-occs-inv-def* **by** (*auto simp*: *occurrence-list-def all-occurrences-def correct-occurence-list-def*)
subgoal
unfolding *populate-occs-inv-def* **by** (*auto simp*: *occurrence-list-def take-Suc-conv-app-nth Cons-nth-drop-Suc*[*symmetr*
dest: *subset-mset-imp-subset-add-mset intro*: *correct-occurence-list-mono-candsI2 intro*: *G'*)
subgoal unfolding *populate-occs-inv-def* **by** (*auto simp*: *occurrence-list-def take-Suc-conv-app-nth Cons-nth-drop-Suc*[*symmetric*]
dest: *subset-mset-imp-subset-add-mset intro*: *correct-occurence-list-mono-candsI2*)
subgoal by *auto*
subgoal by *auto*
apply (*rule correct-occurence-list*; *assumption*)
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal unfolding *populate-occs-inv-def*
apply *auto*
by (*metis add.commute distinct-in-set-take-iff mset-subset-eqD mset-subset-eq-add-right nat-neq-iff set-mset-mset*)

subgoal for *x xa s a b aa ba xb xc*
unfolding *populate-occs-inv-def* **apply** *simp*
by (*auto simp*: *take-Suc-conv-app-nth Cons-nth-drop-Suc*[*symmetric*] *intro*: *G*)
subgoal by *auto*
subgoal for *x xa s a b aa ba xb xc*
unfolding *populate-occs-inv-def* **by** (*auto simp*: *take-Suc-conv-app-nth Cons-nth-drop-Suc*[*symmetric*]
intro: *correct-occurence-list-mono-candsI2 dest*: *forward-subsumption-all-wl-pre-length-ge2 intro*!: *G'*)
subgoal by *auto*
subgoal unfolding *populate-occs-inv-def* **by** (*auto simp*: *take-Suc-conv-app-nth Cons-nth-drop-Suc*[*symmetric*]
intro: *correct-occurence-list-mono-candsI2 dest*: *subset-mset-imp-subset-add-mset*)
subgoal by *auto*
subgoal by (*auto simp*: *populate-occs-inv-def populate-occs-spec-def correct-occurence-list-def*
dest: *mset-eq-setD simp flip*: *distinct-mset-mset-distinct*)
subgoal by (*auto simp*: *populate-occs-inv-def populate-occs-spec-def correct-occurence-list-def*
dest: *mset-eq-setD simp flip*: *distinct-mset-mset-distinct*)
done
qed

definition *forward-subsumption-all-wl2* :: ‹*nat twl-st-wl* ⇒ - *nres*› **where**
‹*forward-subsumption-all-wl2* = (λ*S*. *do* {
ASSERT (*forward-subsumption-all-wl-pre S*);
(*occs*, *xs*) ← *SPEC* (*populate-occs-spec S*);
let m = *length xs*;
D ← *mop-ch-create* (*set-mset* (*all-init-atms-st S*));
let shrunken = {#};
(-, *occs*, *D*, *S*, -, *shrunken*) ←
WHILE$_T$ *forward-subsumption-all-wl2-inv S xs* (λ(*i*, *occs*, *D*, *S*, *n*, *shrunken*). *i* < *m* ∧ *get-conflict-wl S* = *None*)
(λ(*i*, *occs*, *D*, *S*, *n*, *shrunken*). *do* {
let C = *xs*!*i*;
ASSERT(*C*∈#*dom-m* (*get-clauses-wl S*));
D ← *mop-ch-add-all* (*mset* (*get-clauses-wl S* ∝ *C*)) *D*;
(*occs*, *D*, *shrunken*, *T*) ← *try-to-forward-subsume-wl2 C occs* (*mset* (*drop* (*i*) *xs*)) *D shrunken S*;
RETURN (*i*+*1*, *occs*, *D*, *T*, (*length* (*get-clauses-wl S* ∝ *C*)), *shrunken*)
})
(*0*, *occs*, *D*, *S*, *2*, *shrunken*);
ASSERT (*fst occs* = *set-mset* (*all-init-atms-st S*));
ASSERT (*shrunken* ⊆# *mset xs*);
ASSERT (*literals-are-*$\mathcal{L}_{in}$' *S*);
RETURN S
}
)›

lemma *forward-subsumption-all-wl2-forward-subsumption-all-wl*:
‹*forward-subsumption-all-wl2 S* ≤ ⇓*Id* (*forward-subsumption-all-wl S*)›
proof −
have *forward-subsumption-all-wl-alt-def*:
‹*forward-subsumption-all-wl* = (λ*S*. *do* {
ASSERT (*forward-subsumption-all-wl-pre S*);
xs ← *SPEC* (*forward-subsumption-wl-all-cands S*);
let - = *size xs*;
D ← *RETURN* ({#} :: *nat clause*);
let - = ({#} :: *nat multiset*);
ASSERT (*D* = {#});
(*xs*, *S*) ←

WHILE$_T$ *forward-subsumption-all-wl-inv S xs* (λ(*xs*, *S*). *xs* ≠ {#} ∧ *get-conflict-wl S* = *None*)
(λ(*xs*, *S*). *do* {
C ← *SPEC* (λ*C*. *C* ∈# *xs*);
let - = *mset* (*get-clauses-wl S* ∝ *C*) + {#};
T ← *try-to-forward-subsume-wl C xs S*;
ASSERT (∀ *D*∈#*remove1-mset C xs*. *get-clauses-wl T* ∝ *D* = *get-clauses-wl S* ∝ *D*);
RETURN (*remove1-mset C xs*, *T*)
})
(*xs*, *S*);
RETURN S
}
)›
unfolding *forward-subsumption-all-wl-def Let-def* **by** (*auto intro*!: *ext*)
have [*refine*]: ‹*SPEC*
(λ*xs*. *mset xs* ⊆# *dom-m* (*get-clauses-wl S*) ∧
sorted-wrt (λ*a b*. *length* (*get-clauses-wl S* ∝ *a*) ≤ *length* (*get-clauses-wl S* ∝ *b*)) *xs*)
≤ ⇓ {(*xs*,*ys*). *mset xs* = *ys* ∧ *mset xs* ⊆# *dom-m* (*get-clauses-wl S*) ∧
sorted-wrt (λ*a b*. *length* (*get-clauses-wl S* ∝ *a*) ≤ *length* (*get-clauses-wl S* ∝ *b*)) *xs*}
(*SPEC* (λ*xs*. *xs* ⊆# *dom-m* (*get-clauses-wl S*)))›
by (*auto intro*!: *RES-refine*)
have [*refine*]: ‹*SPEC* (*populate-occs-spec S*) ≤ ⇓ {((*occs*,*xs*), *ys*). *ys* = *mset xs* ∧ *populate-occs-spec S* (*occs*, *xs*)}(*SPEC* (*forward-subsumption-wl-all-cands S*))›
(**is** ‹- ≤ ⇓ *?populate* -›)
unfolding *populate-occs-spec-def prod.simps forward-subsumption-wl-all-cands-def*
by (*rule RES-refine*)
(*auto dest*: *mset-le-decr-left2 mset-le-decr-left1*)
define *loop-inv* **where** ‹*loop-inv xs* = {((*i*, *occs*′, *D* :: *clause-hash*, *U*, *n*, *shrunken* :: *nat multiset*), (*xsa*, *V*)). (*U*,*V*)∈*Id* ∧ *i* ≤ *length xs* ∧
(*D*, {#}) ∈ *clause-hash-ref* (*all-init-atms-st V*) ∧ *shrunken* ⊆# *mset* (*take i xs*) ∧
(*i* < *length xs* ⟶ *length* (*get-clauses-wl U* ∝ (*xs*!*i*)) ≥ *n*)∧
correct-occurence-list U occs′ *xsa n* ∧ *xsa* = *mset* (*drop i xs*)}› **for** *xs*
have *loop-init*: ‹(*occsxs*, *xsa*) ∈ *?populate* ⟹
occsxs = (*occs*, *xs*) ⟹
(*D*, *D*′) ∈ *clause-hash-ref* (*all-init-atms-st S*) ⟹
D′ = {#} ⟹
forward-subsumption-all-wl-inv S (*xsa*) (*xsa*, *S*) ⟹ ((*0*, *occs*, *D*, *S*, *2*, {#}), *xsa*, *S*) ∈ *loop-inv xs*›
(**is** ‹- ⟹ - ⟹ - ⟹- ⟹ - ⟹ - ∈ *?loopinv*›) **for** *D occs xsa aa xs occsxs uu D*′
by (*cases xs*) (*auto simp*: *populate-occs-spec-def loop-inv-def*)
have [*refine*]: ‹(*a*,*b*) ∈ *Id* ⟹ (*c*, *d*) ∈ *Id* ⟹ *simplify-clause-with-unit-st-wl a c* ≤⇓*Id* (*simplify-clause-with-unit-st-wl b d*)› **for** *a b c d*
by *auto*

have *try-to-forward-subsume-wl2-pre0*: ‹⋀*x xs x1 x2 D Da xa x*′ *x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e C Db shrunken x2d*′.
forward-subsumption-all-wl-pre S ⟹
(*x*, *xs*) ∈ *?populate* ⟹
x = (*x1*, *x2*) ⟹
(*D*, *Da*) ∈ *clause-hash-ref* (*all-init-atms-st S*) ⟹
Da = {#} ⟹
(*xa*, *x*′) ∈ *loop-inv x2* ⟹
(*case xa of* (*i*, *occs*, *D*, *S*, *n*, *shrunken*) ⇒ *i* < *length x2* ∧ *get-conflict-wl S* = *None*) ⟹
(*case x*′ *of* (*xs*, *S*) ⇒ *xs* ≠ {#} ∧ *get-conflict-wl S* = *None*) ⟹
forward-subsumption-all-wl2-inv S x2 xa ⟹
forward-subsumption-all-wl-inv S xs x′ ⟹
x′ = (*x1a*, *x2a*) ⟹

x2d' = (*x2e*, *shrunken*) ⟹
x2d = (*x1e*, *x2d'*) ⟹
x2c = (*x1d*, *x2d*) ⟹
x2b = (*x1c*, *x2c*) ⟹
xa = (*x1b*, *x2b*) ⟹
(*x2* ! *x1b*, *C*) ∈ *nat-rel* ⟹
(*Db*, *mset* (*get-clauses-wl x2a* ∝ *C*) + {#}) ∈ *clause-hash-ref* (*all-init-atms-st x1e*) ⟹
try-to-forward-subsume-wl2-pre0 (*mset* (*get-clauses-wl x2a* ∝ *C*) + {#}) (*x2* ! *x1b*) *x1c* (*mset* (*drop x1b x2*)) *Db x1e x2e*›
unfolding *forward-subsumption-all-inv-def forward-subsumption-all-wl-inv-def forward-subsumption-all-wl2-inv-def loop-inv-def*
apply (*hypsubst*, *unfold prod.simps*, *normalize-goal*+)
subgoal for *x xs x1 x2 D Da xa x' x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e C Db xb xc*
apply (*auto simp add*: *try-to-forward-subsume-wl2-pre0-def populate-occs-spec-def drop-Suc-nth*)
apply (*metis add-mset-disjoint*(*2*) *correct-occurence-list-def insert-DiffM union-single-eq-member*)
by (*metis add-mset-add-mset-same-iff correct-occurence-list-def distinct-mem-diff-mset insert-DiffM set-mset-mset union-single-eq-member*)
done

have *subsumed-postinv*: ‹
forward-subsumption-all-wl-pre S ⟹
(*x*, *xs*) ∈ *?populate* ⟹
x ∈ *Collect* (*populate-occs-spec S*) ⟹
xs ∈ *Collect* (*forward-subsumption-wl-all-cands S*) ⟹
x = (*x1*, *x2*) ⟹
(*xa*, *x'*) ∈ *loop-inv x2* ⟹
case xa of (*i*, *occs*, *D*, *S*, *n*, *shrunken*) ⇒ *i* < *length x2* ∧ *get-conflict-wl S* = *None* ⟹
case x' of (*xs*, *S*) ⇒ *xs* ≠ {#} ∧ *get-conflict-wl S* = *None* ⟹
forward-subsumption-all-wl2-inv S x2 xa ⟹
forward-subsumption-all-wl-inv S xs x' ⟹
x' = (*x1a*, *x2a*) ⟹
x2d' = (*x2e*, *shrunken*) ⟹
x2d = (*x1e*, *x2d'*) ⟹
x2c = (*x1d*, *x2d*) ⟹
x2b = (*x1c*, *x2c*) ⟹
xa = (*x1b*, *x2b*) ⟹
(*x2* ! *x1b*, *C*) ∈ *nat-rel* ⟹
x2 ! *x1b* ∈# *dom-m* (*get-clauses-wl x1e*) ⟹
inres (*mop-ch-add-all* (*mset* (*get-clauses-wl x1e* ∝ (*x2* ! *x1b*))) *x1d*) *Db* ⟹
(*Db*, *mset* (*get-clauses-wl x2a* ∝ *C*) + {#}) ∈ *clause-hash-ref* (*all-init-atms-st x1e*) ⟹
(*xb*, *Sa*) ∈ *try-to-forward-subsume-wl-rel x1e* (*x2* ! *x1b*) (*mset* (*drop x1b x2*)) *x1c x2e shrunken* ⟹
∀ *D*∈#*remove1-mset C x1a*. *get-clauses-wl Sa* ∝ *D* = *get-clauses-wl x2a* ∝ *D* ⟹
xb = (*a*, *b*) ⟹
b = (*aa*, *ba*) ⟹
ba = (*ab*, *bb*) ⟹
((*x1b* + *1*, *a*, *aa*, *bb*, *length* (*get-clauses-wl x1e* ∝ (*x2* ! *x1b*)), *ab*), *remove1-mset C x1a*, *Sa*)
∈ *loop-inv x2*›
for *x xs x1 x2 D Da xa x' x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e C Db xb Sa a b aa ba shrunken x2d' ab bb*
unfolding *forward-subsumption-all-inv-def forward-subsumption-all-wl-inv-def forward-subsumption-all-wl2-inv-def loop-inv-def populate-occs-spec-def mem-Collect-eq try-to-forward-subsume-wl-rel-def*
apply (*hypsubst*, *unfold prod.simps*, *normalize-goal*+)
apply (*auto simp add*: *try-to-forward-subsume-wl2-pre0-def populate-occs-spec-def drop-Suc-nth take-Suc-conv-app-nth*
dest: *sorted-wrt-nth-less*[*of* - - *x1b* ‹*Suc x1b*›] *intro*: *subset-mset-imp-subset-add-mset*)
by (*meson mset-subset-eq-add-mset-cancel subset-mset.dual-order.trans*)

have *in-all-lits-in-cls*: ‹*xaa* ∈ *set* (*get-clauses-wl S* ∝ *C*) ⟹ *C* ∈# *dom-m* (*get-clauses-wl S*) ⟹
xaa ∈# *all-lits-st S*› **for** *C S xaa*
by (*auto simp*: *all-lits-st-def all-lits-def ran-m-def all-lits-of-mm-add-mset in-clause-in-all-lits-of-m*
dest!: *multi-member-split*)

have *in-atms*: ‹*forward-subsumption-all-wl-pre S* ⟹
forward-subsumption-all-wl-pre S ⟹
(*x*, *xs*) ∈ {((*occs*, *xs*), *ys*). *ys* = *mset xs* ∧ *populate-occs-spec S* (*occs*, *xs*)} ⟹
x ∈ *Collect* (*populate-occs-spec S*) ⟹
xs ∈ *Collect* (*forward-subsumption-wl-all-cands S*) ⟹
x = (*x1*, *x2*) ⟹
(*D*, *Da*) ∈ *clause-hash-ref* (*all-init-atms-st S*) ⟹
Da = {#} ⟹
(*xa*, *x'*) ∈ *loop-inv x2* ⟹
case xa of (*i*, *occs*, *D*, *S*, *n*, *shrunken*) ⇒ *i* < *length x2* ∧ *get-conflict-wl S* = *None* ⟹
case x' of (*xs*, *S*) ⇒ *xs* ≠ {#} ∧ *get-conflict-wl S* = *None* ⟹
forward-subsumption-all-wl2-inv S x2 xa ⟹
forward-subsumption-all-wl-inv S xs x' ⟹
x' = (*x1a*, *x2a*) ⟹
x2d = (*x1e*, *x2d'*) ⟹
x2c = (*x1d*, *x2d*) ⟹
x2b = (*x1c*, *x2c*) ⟹
xa = (*x1b*, *x2b*) ⟹
(*x2* ! *x1b*, *C*) ∈ *nat-rel* ⟹
atms-of (*mset* (*get-clauses-wl x1e* ∝ (*x2* ! *x1b*))) ⊆ *set-mset* (*all-init-atms-st x1e*)›
for *x xs x1 x2 D Da xa x' x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e C x2d' shrunken*
unfolding *forward-subsumption-all-inv-def forward-subsumption-all-wl-inv-alt-def*
forward-subsumption-all-wl2-inv-def loop-inv-def populate-occs-spec-def mem-Collect-eq
all-init-atms-st-alt-def atms-of-def
apply (*cases x2d'*, *hypsubst*, *unfold prod.simps*, *normalize-goal*+)
apply (*frule literals-are-*$\mathcal{L}_{in}$*'-all-init-atms-alt-def*[*of x1e*])
subgoal
by (*auto simp*: *all-init-atms-st-alt-def all-atms-st-alt-def drop-Suc-nth*
simp del: *all-atms-st-alt-def*[*symmetric*] *intro*!: *imageI*
in-all-lits-in-cls[*of* - - ‹*x2* ! *x1b*›])
done
have *distinct*: ‹*forward-subsumption-all-wl-pre S* ⟹
forward-subsumption-all-wl-inv S xs x' ⟹
x' = (*x1a*, *x2a*) ⟹
(*C'*, *C*) ∈ *nat-rel* ⟹
C' ∈# *dom-m* (*get-clauses-wl x2a*) ⟹
distinct-mset (*mset* (*get-clauses-wl x2a* ∝ *C*))›
for *x' x1a x2a xa x1b x2b x1c x2c x1d x2d x1e x2e x2 C xs C'*
unfolding *forward-subsumption-all-wl-inv-alt-def case-prod-beta*
forward-subsumption-all-inv-def
apply *normalize-goal*+
apply (*elim rtranclp-cdcl-twl-inprocessing-l-twl-st-l*)
apply *assumption*+
unfolding *twl-struct-invs-def twl-st-inv-alt-def*
by (*simp add*: *mset-take-mset-drop-mset'*)

show *?thesis*
unfolding *forward-subsumption-all-wl-alt-def forward-subsumption-all-wl2-def*
apply (*rewrite at* ‹*let* - = *length* - *in* -› *Let-def*)
apply (*refine-vcg mop-occ-list-create mop-ch-create mop-ch-add-all*

try-to-forward-subsume-wl2-try-to-forward-subsume-wl
WHILEIT-refine-with-inv)
apply (*rule loop-init*; *assumption*)
subgoal unfolding *forward-subsumption-all-wl2-inv-def loop-inv-def* **by** *auto*
subgoal by (*auto simp*: *loop-inv-def*)
subgoal by (*auto simp*: *in-set-dropI loop-inv-def*)
subgoal
unfolding *loop-inv-def populate-occs-spec-def mem-Collect-eq*
forward-subsumption-all-inv-def forward-subsumption-all-wl-inv-alt-def
forward-subsumption-all-wl2-inv-def case-prod-beta **apply** *normalize-goal+*
by (*auto simp add*: *forward-subsumption-wl-all-cands-def*
simp flip: *Cons-nth-drop-Suc dest*: *mset-subset-eq-insertD*)
apply (*solves* ‹*auto simp*: *loop-inv-def*›)
subgoal by (*rule in-atms*)
subgoal by (*auto simp*: *loop-inv-def*)
subgoal by *auto*
subgoal apply (*rule distinct*)
by *assumption+* (*simp add*: *loop-inv-def case-prod-beta*)
apply (*subst clause-hash-ref-cong*)
defer apply *assumption*
apply (*rule try-to-forward-subsume-wl2-pre0*; *assumption?*)
subgoal by (*auto simp*: *loop-inv-def*)
subgoal by (*auto simp*: *loop-inv-def*)
subgoal for *x xs x1 x2 D Da xa x' x1a x2a x1b x2b x1c x2c x1d x2d x1e x2d' x2e shrunken C Db*
xb Sa a b aa ba
by (*rule subsumed-postinv*)
subgoal by (*auto simp*: *loop-inv-def correct-occurence-list-def*)
subgoal by (*auto simp*: *loop-inv-def intro*: *subset-mset.dual-order.trans*[*OF mset-take-subseteq*])
subgoal by (*auto simp*: *forward-subsumption-all-wl2-inv-def*
dest!: *forward-subsumption-all-wl-inv-literals-are-*$\mathcal{L}_{in}$*'D*)
subgoal by (*auto simp*: *loop-inv-def*)
subgoal by (*auto simp*: *loop-inv-def*)
done
qed

22.2.2 Refinement to isasat.

definition *is-candidate-forward-subsumption* **where**
‹*is-candidate-forward-subsumption C N* = *do* {
ASSERT (*C* ∈# *dom-m N*);
SPEC (λ*b* :: *bool*. *b* ⟶ ¬*irred N C* ∧ *length* (*N* ∝ *C*) ≠ *2*)
}›

lemma *incr-forward-rounds-st*:
‹(*S*, *S'*) ∈ *twl-st-heur-restart-occs' r u* ⟹
(*incr-forward-rounds-st S*, *S'*) ∈ *twl-st-heur-restart-occs' r u*›
by (*simp add*: *IsaSAT-Restart.all-init-atms-alt-def twl-st-heur-restart-occs-def incr-forward-rounds-st-def*)

lemma *red-in-dom-number-of-learned-ge1-twl-st-heur-restart-occs*:
assumes ‹(*S*,*T*) ∈ *twl-st-heur-restart-occs' r u*› **and**
‹*C* ∈# *dom-m* (*get-clauses-wl T*)›
‹*arena-status* (*get-clauses-wl-heur S*) *C* ≠ *IRRED*›
shows ‹*1* ≤ *get-learned-count-number S*›
proof −
have ‹*clss-size-corr-restart* (*get-clauses-wl T*) (*IsaSAT-Setup.get-unkept-unit-init-clss-wl T*) {#}

(*IsaSAT-Setup.get-kept-unit-init-clss-wl T*) (*IsaSAT-Setup.get-kept-unit-learned-clss-wl T*)
(*get-subsumed-init-clauses-wl T*) {#} (*get-init-clauses0-wl T*) {#} (*get-learned-count S*)› **and**
‹*valid-arena* (*get-clauses-wl-heur S*) (*get-clauses-wl T*) (*set* (*get-vdom S*))›
using *assms*(*1*) **unfolding** *twl-st-heur-restart-occs-def Let-def* **by** *auto*
then show *?thesis*
using *assms*(*2−*) *red-in-dom-number-of-learned-ge1*[*of C* ‹*get-clauses-wl T*›]
by (*auto simp*: *clss-size-corr-restart-def clss-size-def clss-size-lcount-def*
arena-lifting)
qed

lemma *red-in-dom-number-of-learned-ge1-twl-st-heur-restart-occs2*:
assumes ‹(*S*,*T*) ∈ *twl-st-heur-restart-occs′ r u*› **and**
‹*C* ∈# *dom-m* (*get-clauses-wl T*)›
‹¬*irred* (*get-clauses-wl T*) *C*›
shows ‹*1* ≤ *get-learned-count-number S*›
proof −
have ‹*clss-size-corr-restart* (*get-clauses-wl T*) (*IsaSAT-Setup.get-unkept-unit-init-clss-wl T*) {#}
(*IsaSAT-Setup.get-kept-unit-init-clss-wl T*) (*IsaSAT-Setup.get-kept-unit-learned-clss-wl T*)
(*get-subsumed-init-clauses-wl T*) {#} (*get-init-clauses0-wl T*) {#} (*get-learned-count S*)›
using *assms*(*1*) **unfolding** *twl-st-heur-restart-occs-def Let-def* **by** *auto*
then show *?thesis*
using *assms*(*2−*) *red-in-dom-number-of-learned-ge1*[*of C* ‹*get-clauses-wl T*›]
by (*auto simp*: *clss-size-corr-restart-def clss-size-def clss-size-lcount-def*
arena-lifting)
qed

lemma *mop-cch-remove-one-mop-ch-remove-one*:
assumes
‹(*L*,*L′*)∈*Id*› **and**
DD′: ‹(*D*,*D′*)∈*clause-hash*›
shows ‹*mop-cch-remove-one L D*
≤ ⇓ *clause-hash*
(*mop-ch-remove L′ D′*)›
using *assms*
unfolding *mop-cch-remove-one-def mop-ch-remove-def*
apply (*refine-vcg*,
auto simp: *ch-remove-pre-def clause-hash-def ch-remove-def distinct-mset-remove1-All*
dest: *bspec*[*of - - L*])
apply (*cases L*; *auto*)
done

lemma *mop-arena-status2*:
assumes ‹(*C*,*C′*)∈*nat-rel*› ‹*C* ∈ *vdom*›
‹*valid-arena arena N vdom*›
shows
‹*mop-arena-status arena C*
≤ *SPEC*
(λ*c*. (*c*, *C′* ∈# *dom-m N*)
∈ {(*a*,*b*). (*b* ⟶ (*a* = *IRRED* ⟷ *irred N C*) ∧ (*a* = *LEARNED* ⟷ ¬*irred N C*)) ∧ (*a* = *DELETED* ⟷ ¬*b*)})›
using *assms arena-dom-status-iff*[*of arena N vdom C*] **unfolding** *mop-arena-status-def*
by (*cases* ‹*C* ∈# *dom-m N*›)

(*auto intro*!: *ASSERT-leI simp*: *arena-is-valid-clause-vdom-def*
arena-lifting)

lemma *isa-subsume-clauses-match2-subsume-clauses-match2*:
assumes
SS': ‹(*S*, *S'*) ∈ *twl-st-heur-restart-occs' r u*› **and**
CC': ‹(*C*,*C'*)∈*nat-rel*› **and**
EE': ‹(*E*,*E'*)∈*nat-rel*› **and**
DD': ‹(*D*,*D'*)∈*clause-hash*›
shows
‹*isa-subsume-clauses-match2 C E S D*
≤ ⇓ *Id*
(*subsume-clauses-match2 C' E' S' D'*)›
proof −
have [*refine*]: ‹((*0*, *SUBSUMED-BY C*), *0*, *SUBSUMED-BY C'*) ∈ *nat-rel* ×*r Id*›
using *assms* **by** *auto*
show *?thesis*
unfolding *isa-subsume-clauses-match2-def subsume-clauses-match2-def mop-arena-length-st-def*
Let-def[*of* ‹*mark-literal-for-unit-deletion* -›]
apply (*refine-vcg mop-arena-length*[**where** *vdom*=‹*set* (*get-vdom S*)›, *THEN fref-to-Down-curry*,
unfolded comp-def]
mop-arena-lit2[**where** *vdom*=‹*set* (*get-vdom S*)›] *mop-cch-in-mop-ch-in*)
subgoal using *assms* **unfolding** *isa-subsume-clauses-match2-pre-def* **by** *fast*
subgoal unfolding *subsume-clauses-match2-pre-def subsume-clauses-match-pre-def*
by *auto*
subgoal using *SS' CC' EE'* **by** (*auto simp*: *twl-st-heur-restart-occs-def*)
subgoal using *SS' CC' arena-lifting*(*10*)[*of* ‹*get-clauses-wl-heur S*› ‹*get-clauses-wl S'*› ‹*set* (*get-vdom*
S)› *C*] **apply** −
unfolding *isa-subsume-clauses-match2-pre-def subsume-clauses-match2-pre-def subsume-clauses-match-pre-def*
by *normalize-goal*+ (*simp add*: *twl-st-heur-restart-occs-def*)
subgoal using *EE'* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *SS' CC'* **by** (*auto simp*: *twl-st-heur-restart-occs-def*)
subgoal using *CC'* **by** *auto*
subgoal by *auto*
subgoal using *DD'* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *DD'* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *CC'* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

lemma *valid-occs-in-vdomI*:
‹*valid-occs occs* (*get-aivdom S*) ⟹ *x1* < *length* (*occs* ! *nat-of-lit L*) ⟹
nat-of-lit L < *length occs* ⟹ *cocc-list-at occs L x1* ∈ *set* (*get-vdom S*)›
apply (*drule nth-mem*)
unfolding *valid-occs-def cocc-list-at-def Union-eq subset-iff*
by (*auto dest*: *spec*[*of* - ‹*occs* ! *nat-of-lit L* ! *x1*›] *simp*: *cocc-content-set-def*)

definition *mop-ch-remove-all-clauses* **where**
‹*mop-ch-remove-all-clauses C D = do {*
(-, D) ← WHILE$_T$ (λ(C, D). C ≠ {#})
(λ(C, D). do {L ← SPEC (λL. L ∈# C); D ← mop-ch-remove L D; RETURN (remove1-mset L C, D)})
(C, D);
RETURN D
} ›

lemma *diff-mono-mset*: *a ⊆# b ⟹ a − c ⊆# b − c*
by (*meson subset-eq-diff-conv subset-mset.dual-order.eq-iff subset-mset.dual-order.trans*)

lemma *mop-ch-remove-all-clauses-mop-ch-remove-all*:
‹*mop-ch-remove-all-clauses C D ≤ ⇓Id (mop-ch-remove-all C D)*›
unfolding *mop-ch-remove-all-clauses-def mop-ch-remove-all-def mop-ch-remove-def nres-monad3*
apply (*refine-vcg WHILET-rule*[**where** *R = ‹measure (λ(i, -). size i)›* **and**
I = ‹ λ(C', D'). C' ⊆# C ∧ C' ⊆# snd D' ∧ snd D = snd D' + C − C' ∧ fst D = fst D'›])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal for *s x b* **unfolding** *ch-remove-pre-def* **by** *force*
subgoal by *auto*
subgoal unfolding *ch-remove-def* **by** (*auto simp*: *case-prod-beta diff-mono-mset*)
subgoal
by (*drule multi-member-split*)
(*clarsimp simp*: *ch-remove-def case-prod-beta ch-remove-pre-def*)
subgoal by (*auto simp*: *ch-remove-def case-prod-beta*)
subgoal by (*clarsimp dest!*: *multi-member-split simp*: *size-Diff-singleton-if*)
subgoal for *s a b* **by** (*cases b*) (*auto simp*: *ch-remove-all-def case-prod-beta*)
done

lemma *mop-cch-remove-all-clauses-mop-ch-remove-all-clauses*:
assumes
SS': ‹*(S, S') ∈ twl-st-heur-restart-occs' r u*›**and**
‹*(C,C')∈nat-rel*› **and**
DD': ‹*(D,D')∈clause-hash*› **and**
C: ‹*C ∈# dom-m (get-clauses-wl S')*›
shows ‹*mop-cch-remove-all-clauses S C D*
≤ ⇓ clause-hash
(mop-ch-remove-all (mset (get-clauses-wl S' ∝ C')) D')›
proof −
define *f* **where** *f C = SPEC (λL. L ∈# C)* **for** *C* :: ‹*nat clause*›
have *eq*[*simp*]: ‹*C' = C*›
using *assms* **by** *auto*
have *valid*: ‹*valid-arena (get-clauses-wl-heur S) (get-clauses-wl S') (set (get-vdom S))*›
using *SS' C* **unfolding** *arena-is-valid-clause-idx-and-access-def arena-is-valid-clause-idx-def twl-st-heur-restart-occs-de*
by (*auto simp*: *arena-lifting*)
have *f*: ‹*(x, x')*
∈ {((n, D), m, D'). (D, D') ∈ clause-hash ∧ m = mset (drop n (get-clauses-wl S' ∝ C))} ⟹
case x of (i, D) ⇒ i < arena-length (get-clauses-wl-heur S) C ⟹
case x' of (C, D) ⇒ C ≠ {#} ⟹

```
    x' = (x1, x2) ⟹
    x = (x1a, x2a) ⟹SPEC (λc. (c, get-clauses-wl S' ∝ C ! x1a) ∈ nat-lit-lit-rel)
    ≤ ⇓ {(a,b). a = b ∧ a = get-clauses-wl S' ∝ C ! x1a} (f x1)› for x1 x1a x2 x2a x x'
    unfolding f-def by (auto intro!: RES-refine in-set-dropI)
  show ?thesis
    apply (rule ref-two-step[OF - mop-ch-remove-all-clauses-mop-ch-remove-all[unfolded Down-id-eq]])
    unfolding mop-cch-remove-all-clauses-def mop-ch-remove-all-clauses-def mop-arena-length-def
      nres-monad3 bind-to-let-conv f-def[symmetric]
    apply (subst Let-def[of ‹arena-length - -›])
    apply (refine-vcg WHILET-refine[where R = ‹{((n, D), (m, D'). (D,D')∈ clause-hash ∧
      m = mset (drop n (get-clauses-wl S' ∝ C))}›] mop-cch-remove-one-mop-ch-remove-one)
    subgoal using valid C unfolding arena-is-valid-clause-idx-def twl-st-heur-restart-occs-def apply −
      by (rule exI[of - ‹get-clauses-wl S'›], auto intro!: exI[of - ‹get-vdom S›])
    subgoal using DD' by auto
    subgoal using valid by (auto simp: arena-lifting C)
    subgoal using valid arena-lifting(4,7)[OF valid C] by (force simp: C)
    apply (rule mop-arena-lit[THEN order-trans])
    apply (rule valid)
    apply (rule C)
    subgoal by auto
    apply (rule refl)
    subgoal using valid by (auto simp: arena-lifting C)
    apply (rule f; assumption)
    subgoal by (auto intro!: in-set-dropI)
    subgoal using DD' by auto
    subgoal by (auto simp flip: Cons-nth-drop-Suc add-mset-remove-trivial-If)
    subgoal by auto
    done
qed

lemma find-best-subsumption-candidate:
  assumes
    SS': ‹(S, S') ∈ twl-st-heur-restart-occs' r u› and
    pre0: ‹push-to-occs-list2-pre C S' occs› and
    occs: ‹(get-occs S, occs) ∈ occurrence-list-ref› and
    le-bound: ‹length (get-clauses-wl S' ∝ C) ≤ Suc (unat32-max div 2)›
  shows ‹find-best-subsumption-candidate C S ≤ SPEC (λL. L ∈# mset (get-clauses-wl S' ∝ C))›
proof −
  have valid: ‹valid-occs (get-occs S) (get-aivdom S)› and
    arena: ‹valid-arena (get-clauses-wl-heur S) (get-clauses-wl S') (set (get-vdom S))›
    using SS' by (auto simp: twl-st-heur-restart-occs-def)
  have
    C: ‹C ∈# dom-m (get-clauses-wl S')› and
    le: ‹2 ≤ length (get-clauses-wl S' ∝ C)› and
    fst: ‹fst occs = set-mset (all-init-atms-st S')› and
    lits: ‹atm-of ‘ set (get-clauses-wl S' ∝ C) ⊆ set-mset (all-init-atms-st S')›
    using pre0 unfolding push-to-occs-list2-pre-def by blast+
  have pre2: ‹arena-lit-pre (get-clauses-wl-heur S) (C+i)›
    if ‹i < length (get-clauses-wl S' ∝ C)› for i
    using that unfolding arena-lit-pre-def apply −
    by (rule exI[of - C])
   (use SS' arena C le in ‹auto simp: arena-is-valid-clause-idx-and-access-def intro!: exI[of - ‹get-clauses-wl
S'›] exI[of - ‹set (get-vdom S)›]›)

  have pre: ‹arena-lit-pre (get-clauses-wl-heur S) C›
    unfolding arena-lit-pre-def
```

by (*rule exI*[*of* - *C*])
(*use SS′ arena C le* **in** ‹*auto simp*: *arena-is-valid-clause-idx-and-access-def intro*!: *exI*[*of* - ‹*get-clauses-wl S′*›] *exI*[*of* - ‹*set* (*get-vdom S*)›]›)

have *lit*[*simp*]: ‹*arena-lit* (*get-clauses-wl-heur S*) (*C* + *i*) = *get-clauses-wl S′* ∝ *C* ! *i*›
if ‹*i* < *length* (*get-clauses-wl S′* ∝ *C*)› **for** *i*
using *that C arena* **by** (*auto simp*: *arena-lifting*)
from *this*[*of 0*] **have** [*simp*]: ‹*arena-lit* (*get-clauses-wl-heur S*) *C* = *get-clauses-wl S′* ∝ *C* ! *0*›
using *le* **by** *fastforce*
have [*simp*]: ‹*arena-length* (*get-clauses-wl-heur S*) *C* = *length* (*get-clauses-wl S′* ∝ *C*)›
using *arena C* **by** (*auto simp*: *arena-lifting*)
have *H*[*intro*]: ‹*ba* ∈ *set* (*get-clauses-wl S′* ∝ *C*) ⟹ *nat-of-lit ba* < *length* (*get-occs S*)› **for** *ba*
using *lits occs* **by** (*cases ba*) (*auto simp*: *map-fun-rel-def occurrence-list-ref-def*
dest!: *bspec*[*of* - - ‹*ba*›] *simp flip*: *fst*)
have [*intro*]: ‹*nat-of-lit* (*get-clauses-wl S′* ∝ *C* ! *i*) < *length* (*get-occs S*)›
if ‹*i* < *length* (*get-clauses-wl S′* ∝ *C*)› **for** *i*
by (*rule H*) (*metis* (*no-types*, *opaque-lifting*) *access-lit-in-clauses-def nth-mem that*)

show *?thesis*
using *le*
unfolding *find-best-subsumption-candidate-def mop-arena-lit-def nres-monad3 mop-arena-length-st-def*
mop-arena-length-def mop-arena-lit2-def mop-cocc-list-length-def
apply (*refine-vcg arena WHILET-rule*[**where** *R* = ‹*measure* (λ(*i*, -). *length* (*get-clauses-wl S′* ∝ *C*) − *i*)› **and**
I = ‹ λ(*i*, *score*, *L*). *L* ∈ *set* (*get-clauses-wl S′* ∝ *C*)›])
subgoal using *pre* **by** *auto*
subgoal by *auto*
subgoal unfolding *cocc-list-length-pre-def* **by** *auto*
subgoal using *C arena* **unfolding** *arena-is-valid-clause-idx-def* **by** *fast*
subgoal by *auto*
subgoal by (*use le* **in** ‹*auto intro*!: *nth-mem*›)
subgoal using *le-bound* **by** *auto*
subgoal by (*auto intro*!: *pre2*)
subgoal by *auto*
subgoal unfolding *cocc-list-length-pre-def* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

lemma *find-best-subsumption-candidate-and-push*:
assumes
SS′: ‹(*S*, *S′*) ∈ *twl-st-heur-restart-occs′ r u*› **and**
pre0: ‹*push-to-occs-list2-pre C S′ occs*› **and**
occs: ‹(*get-occs S*, *occs*) ∈ *occurrence-list-ref*› **and**
le-bound: ‹*length* (*get-clauses-wl S′* ∝ *C*) ≤ *Suc* (*unat32-max div 2*)›
shows ‹*find-best-subsumption-candidate-and-push C S* ≤ *SPEC* (λ(*L*, *push*). *L* ∈# *mset* (*get-clauses-wl S′* ∝ *C*))›
proof −
have *valid*: ‹*valid-occs* (*get-occs S*) (*get-aivdom S*)› **and**
arena: ‹*valid-arena* (*get-clauses-wl-heur S*) (*get-clauses-wl S′*) (*set* (*get-vdom S*))› **and**
heur: ‹*heuristic-rel* (*all-init-atms-st S′*) (*get-heur S*)›
using *SS′* **by** (*auto simp*: *twl-st-heur-restart-occs-def IsaSAT-Restart.all-init-atms-alt-def*)

have
C: ‹*C* ∈# *dom-m* (*get-clauses-wl S′*)› **and**
le: ‹*2* ≤ *length* (*get-clauses-wl S′* ∝ *C*)› **and**
fst: ‹*fst occs* = *set-mset* (*all-init-atms-st S′*)› **and**
lits: ‹*atm-of* ‘ *set* (*get-clauses-wl S′* ∝ *C*) ⊆ *set-mset* (*all-init-atms-st S′*)›
using *pre0* **unfolding** *push-to-occs-list2-pre-def* **by** *blast+*
have *pre2*: ‹*arena-lit-pre* (*get-clauses-wl-heur S*) (*C+i*)›
if ‹*i* < *length* (*get-clauses-wl S′* ∝ *C*)› **for** *i*
using *that* **unfolding** *arena-lit-pre-def* **apply** −
by (*rule exI*[*of* - *C*])
(*use SS′ arena C le* **in** ‹*auto simp*: *arena-is-valid-clause-idx-and-access-def intro*!: *exI*[*of* - ‹*get-clauses-wl S′*›] *exI*[*of* - ‹*set* (*get-vdom S*)›]›)

have *pre*: ‹*arena-lit-pre* (*get-clauses-wl-heur S*) *C*›
unfolding *arena-lit-pre-def*
by (*rule exI*[*of* - *C*])
(*use SS′ arena C le* **in** ‹*auto simp*: *arena-is-valid-clause-idx-and-access-def intro*!: *exI*[*of* - ‹*get-clauses-wl S′*›] *exI*[*of* - ‹*set* (*get-vdom S*)›]›)

have *lit*[*simp*]: ‹*arena-lit* (*get-clauses-wl-heur S*) (*C* + *i*) = *get-clauses-wl S′* ∝ *C* ! *i*›
if ‹*i* < *length* (*get-clauses-wl S′* ∝ *C*)› **for** *i*
using *that C arena* **by** (*auto simp*: *arena-lifting*)
from *this*[*of 0*] **have** [*simp*]: ‹*arena-lit* (*get-clauses-wl-heur S*) *C* = *get-clauses-wl S′* ∝ *C* ! *0*›
using *le* **by** *fastforce*
have [*simp*]: ‹*arena-length* (*get-clauses-wl-heur S*) *C* = *length* (*get-clauses-wl S′* ∝ *C*)›
using *arena C* **by** (*auto simp*: *arena-lifting*)
have *H*[*intro*]: ‹*ba* ∈ *set* (*get-clauses-wl S′* ∝ *C*) ⟹ *nat-of-lit ba* < *length* (*get-occs S*)› **for** *ba*
using *lits occs* **by** (*cases ba*) (*auto simp*: *map-fun-rel-def occurrence-list-ref-def*
dest!: *bspec*[*of* - - ‹*ba*›] *simp flip*: *fst*)
have [*intro*]: ‹*nat-of-lit* (*get-clauses-wl S′* ∝ *C* ! *i*) < *length* (*get-occs S*)›
if ‹*i* < *length* (*get-clauses-wl S′* ∝ *C*)› **for** *i*
by (*rule H*) (*metis* (*no-types*, *opaque-lifting*) *access-lit-in-clauses-def nth-mem that*)

have [*intro*!]: ‹*is-marked-added-heur-pre* (*get-heur S*) (*atm-of L*)›
if ‹*L* ∈ *set* (*get-clauses-wl S′* ∝ *C*)› **for** *L*
proof −
have ‹*atm-of L* ∈ *atm-of* ‘ *set* (*get-clauses-wl S′* ∝ *C*)›
using *that* **by** *auto*
then have ‹*atm-of L* ∈# *all-init-atms-st S′*›
using *lits* **by** *auto*
then show *?thesis*
using *heur lits that*
unfolding *is-marked-added-heur-pre-def*
by (*auto simp*: *is-marked-added-heur-pre-stats-def heuristic-rel-def phase-saving-def*
$\mathcal{L}_{all}$*-add-mset atms-of-def*
heuristic-rel-stats-def dest!: *multi-member-split*)
qed
show *?thesis*
using *le*
unfolding *find-best-subsumption-candidate-and-push-def mop-arena-lit-def nres-monad3 mop-arena-length-st-def*
mop-arena-length-def mop-arena-lit2-def mop-cocc-list-length-def mop-is-marked-added-heur-def
apply (*refine-vcg arena WHILET-rule*[**where** *R* = ‹*measure* (λ(*i*, -). *length* (*get-clauses-wl S′* ∝ *C*) − *i*)› **and**
I = ‹ λ(*i*, *score*, *L*, -). *L* ∈ *set* (*get-clauses-wl S′* ∝ *C*)›])
subgoal using *pre* **by** *auto*
subgoal by *auto*

subgoal unfolding *cocc-list-length-pre-def* **by** *auto*
subgoal using *C arena* **unfolding** *arena-is-valid-clause-idx-def* **by** *fast*
subgoal by *auto*
subgoal by (*use le* **in** ‹*auto intro*!: *nth-mem*›)
subgoal using *le-bound* **by** *auto*
subgoal by (*auto intro*!: *pre2*)
subgoal by *auto*
subgoal unfolding *cocc-list-length-pre-def* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

lemma *twl-st-heur-restart-occs-set-occsI*:
‹(*S*,*S'*)∈*twl-st-heur-restart-occs* ⟹ *valid-occs occs* (*get-aivdom S*) ⟹ (*set-occs-wl-heur occs S*, *S'*) ∈ *twl-st-heur-restart-occs*›
by (*auto simp*: *twl-st-heur-restart-occs-def*)

lemma *valid-occs-append*:
‹*C* ∈ *set* (*get-vdom-aivdom vdm*) ⟹
C ∉# *cocc-content coccs* ⟹ *valid-occs coccs vdm* ⟹ *nat-of-lit La* < *length coccs* ⟹ *valid-occs* (*cocc-list-append C coccs La*) *vdm*›
by (*auto simp*: *valid-occs-def in-set-upd-eq*[*of* ‹*nat-of lit La*› *coccs*])

lemma *in-cocc-content-iff*: ‹*C* ∈# *cocc-content occs* ⟷ (∃ *xs*. *xs* ∈ *set occs* ∧ *C* ∈ *set xs*)›
by (*induction occs*) *auto*

lemma *notin-all-occurrences-notin-cocc*: ‹(*occs*, *occs'*) ∈ *occurrence-list-ref* ⟹ *finite* (*fst occs'*) ⟹ *C* ∉# *all-occurrences* (*mset-set* (*fst occs'*)) *occs'* ⟹ *C* ∉# *cocc-content occs*›
apply (*auto simp*: *occurrence-list-ref-def all-occurrences-def image-Un map-fun-rel-def image-image in-cocc-content-iff in-set-conv-nth*)
apply (*subgoal-tac* ‹*ia div 2* ∈ *x*›)
defer
subgoal for *x y i ia*
by (*drule-tac x*=‹*ia*› **in** *spec*) *auto*
subgoal for *x y i ia*
apply (*drule-tac x*= ‹*if even ia then* (*ia*, *Pos* (*ia div 2*)) *else* (*ia*, *Neg* (*ia div 2*))› **in** *bspec*)
apply *auto*
apply (*rule-tac x*=‹*ia div 2*› **in** *image-eqI*)
apply *auto*
by (*auto split*: *if-splits dest*: *spec*[*of* - *ia*])
done

lemma *set-mset-cocc-content-set*[*simp*]: ‹*set-mset* (*cocc-content occs*) = *cocc-content-set occs*›
by (*auto simp*: *cocc-content-set-def in-mset-sum-list-iff*)

lemma *isa-push-to-occs-list-st-push-to-occs-list2*:
assumes
SS': ‹(*S*, *S'*) ∈ *twl-st-heur-restart-occs' r u*› **and**

```
    CC': ‹(C,C')∈nat-rel› and
    occs: ‹(get-occs S, occs) ∈ occurrence-list-ref› and
    C: ‹C ∈ set (get-vdom S)› and
    length: ‹length (get-clauses-wl S' ∝ C) ≤ Suc (unat32-max div 2)›
  shows ‹isa-push-to-occs-list-st C S
    ≤ ⇓ {(U, occs'). (get-occs U, occs') ∈ occurrence-list-ref ∧ (U, S') ∈ twl-st-heur-restart-occs' r u ∧
get-aivdom U = get-aivdom S} (push-to-occs-list2 C' S' occs)›
proof −
  have eq: ‹C' = C›
    using CC' by auto
  have  push-to-occs-list2-alt-def:
  ‹push-to-occs-list2 C S occs = do {
     ASSERT (push-to-occs-list2-pre C S occs);
     L ← SPEC (λL. L ∈# mset (get-clauses-wl S ∝ C));
     ASSERT (occ-list-append-pre occs L);
     occs ← mop-occ-list-append C occs L;
     RETURN occs
  }› for C S occs
    unfolding push-to-occs-list2-def mop-occ-list-append-def
    apply (subst (3)conj-absorb[symmetric])
    unfolding summarize-ASSERT-conv[symmetric] by auto
  have valid: ‹valid-occs (get-occs S) (get-aivdom S)›
    using SS' unfolding twl-st-heur-restart-occs-def by auto
  have length-bounded: ‹length (get-occs S ! nat-of-lit L) < length (get-clauses-wl-heur S)›
    if
      push: ‹push-to-occs-list2-pre C S' occs› and
      ‹(L, La) ∈ nat-lit-lit-rel› and
      ‹La ∈ {L. L ∈# mset (get-clauses-wl S' ∝ C)}› and
      pre: ‹occ-list-append-pre occs La›
      for L La
  proof −
    define n where ‹n = get-occs S ! nat-of-lit L›
    have arena: ‹valid-arena (get-clauses-wl-heur S) (get-clauses-wl S') (set (get-vdom S))› and
      occs: ‹valid-occs (get-occs S) (get-aivdom S)› and
      aivdom: ‹aivdom-inv-dec (get-aivdom S) (dom-m (get-clauses-wl S'))›
      using SS' unfolding twl-st-heur-restart-occs-def
      by fast+
    have H: ‹add-mset C (cocc-content (get-occs S)) ⊆# mset (get-vdom S)›
     using occs notin-all-occurrences-notin-cocc[OF assms(3), of C] unfolding valid-occs-def apply −
      by (subst distinct-subseteq-iff[symmetric])
       (use push C in ‹auto simp: push-to-occs-list2-pre-def›)
    have ‹nat-of-lit L < length (get-occs S)›
       using pre that(2) assms(3) unfolding occ-list-append-pre-def by (cases L) (auto simp: occur-
rence-list-ref-def map-fun-rel-def
        dest!: bspec[of - - L])
    from nth-mem[OF this] have ‹length (get-occs S ! nat-of-lit L) < length (get-vdom S)›
     using multi-member-split[of ‹get-occs S ! nat-of-lit L› ‹mset (get-occs S)›] H[THEN size-mset-mono]
      by (auto dest!: split-list simp: n-def[symmetric])
    moreover have ‹length (get-vdom S) ≤ length (get-clauses-wl-heur S)›
      using valid-arena-vdom-le(2)[OF arena] aivdom unfolding aivdom-inv-dec-alt-def
      by (simp add: distinct-card)
    finally show ?thesis .
  qed
  show ?thesis
    unfolding isa-push-to-occs-list-st-def push-to-occs-list2-alt-def eq
    apply (refine-vcg find-best-subsumption-candidate[OF SS' - occs]
```

```
    mop-cocc-list-append-mop-occ-list-append occs)
   subgoal using length by auto
   subgoal by (rule length-bounded)
   subgoal by auto
   subgoal using SS' C valid notin-all-occurrences-notin-cocc[OF occs, of C]
    by (auto 9 2 intro!: twl-st-heur-restart-occs-set-occsI
     intro!: valid-occs-append
     simp: push-to-occs-list2-pre-def)
   done
qed

lemma isa-maybe-push-to-occs-list-st-push-to-occs-list2:
 assumes
   SS': ‹(S, S') ∈ twl-st-heur-restart-occs' r u› and
   CC': ‹(C,C')∈nat-rel›and
   occs: ‹(get-occs S, occs) ∈ occurrence-list-ref› and
   C: ‹C ∈ set (get-vdom S)› and
   length: ‹length (get-clauses-wl S' ∝ C) ≤ Suc (unat32-max div 2)›
 shows ‹isa-maybe-push-to-occs-list-st C S
   ≤ ⇓ {(U, occs'). (get-occs U, occs') ∈ occurrence-list-ref ∧ (U, S') ∈ twl-st-heur-restart-occs' r u ∧
get-aivdom U = get-aivdom S} (maybe-push-to-occs-list2 C' S' occs)›
proof −
 have eq: ‹C' = C›
   using CC' by auto
 have maybe-push-to-occs-list2-alt-def:
 ‹maybe-push-to-occs-list2 C S occs = do {
   ASSERT (push-to-occs-list2-pre C S occs);
   b ← SPEC (λ-. True);
  if b then do {
    L ← SPEC (λL. L ∈# mset (get-clauses-wl S ∝ C));
    ASSERT (occ-list-append-pre occs L);
    occs ← mop-occ-list-append C occs L;
    RETURN occs
  } else RETURN occs
 }› for C S occs
   unfolding maybe-push-to-occs-list2-def mop-occ-list-append-def
   apply (subst (3)conj-absorb[symmetric])
   unfolding nres-monad3
   unfolding summarize-ASSERT-conv[symmetric]
   by (auto cong: if-cong intro: bind-cong[OF refl] simp: bind-ASSERT-eq-if)

 have valid: ‹valid-occs (get-occs S) (get-aivdom S)›
   using SS' unfolding twl-st-heur-restart-occs-def by auto
 have length-bounded: ‹length (get-occs S ! nat-of-lit L) < length (get-clauses-wl-heur S)›
   if
    push: ‹push-to-occs-list2-pre C S' occs› and
    ‹(L, La) ∈ nat-lit-lit-rel› and
    ‹(x, b) ∈ {((L, push), push'). L ∈# mset (get-clauses-wl S' ∝ C) ∧ push = push'}›
    ‹x = (L, x2)› and
    pre: ‹occ-list-append-pre occs La›
    for L La x1 b x2 x
 proof −
   define n where ‹n = get-occs S ! nat-of-lit L›
   have arena: ‹valid-arena (get-clauses-wl-heur S) (get-clauses-wl S') (set (get-vdom S))› and
    occs: ‹valid-occs (get-occs S) (get-aivdom S)› and
    aivdom: ‹aivdom-inv-dec (get-aivdom S) (dom-m (get-clauses-wl S'))›
```

```
    using SS' unfolding twl-st-heur-restart-occs-def
    by fast+
  have H: ‹add-mset C (cocc-content (get-occs S)) ⊆# mset (get-vdom S)›
   using occs notin-all-occurrences-notin-cocc[OF assms(3), of C] unfolding valid-occs-def apply −
    by (subst distinct-subseteq-iff[symmetric])
     (use push C in ‹auto simp: push-to-occs-list2-pre-def›)
  have ‹nat-of-lit L < length (get-occs S)›
     using pre that(2) assms(3) unfolding occ-list-append-pre-def by (cases L) (auto simp: occur-
rence-list-ref-def map-fun-rel-def
      dest!: bspec[of - - L])
  from nth-mem[OF this] have ‹length (get-occs S ! nat-of-lit L) < length (get-vdom S)›
   using multi-member-split[of ‹get-occs S ! nat-of-lit L› ‹mset (get-occs S)›] H[THEN size-mset-mono]
    by (auto dest!: split-list simp: n-def[symmetric])
  moreover have ‹length (get-vdom S) ≤ length (get-clauses-wl-heur S)›
    using valid-arena-vdom-le(2)[OF arena] aivdom unfolding aivdom-inv-dec-alt-def
    by (simp add: distinct-card)
  finally show ?thesis .
 qed
 have find-best-subsumption-candidate-and-push:
  ‹find-best-subsumption-candidate-and-push C S ≤ ⇓ {((L, push), push'). L ∈# mset (get-clauses-wl
S' ∝ C)∧ push = push'} (SPEC (λ-. True))›
  if
   SS': ‹(S, S') ∈ twl-st-heur-restart-occs' r u› and
   pre0: ‹push-to-occs-list2-pre C S' occs› and
   occs: ‹(get-occs S, occs) ∈ occurrence-list-ref› and
   le-bound: ‹length (get-clauses-wl S' ∝ C) ≤ Suc (unat32-max div 2)›
 by (rule find-best-subsumption-candidate-and-push[OF that, THEN order-trans]) (auto simp: conc-fun-RES)
 show ?thesis
   unfolding isa-maybe-push-to-occs-list-st-def maybe-push-to-occs-list2-alt-def eq
   apply (refine-vcg find-best-subsumption-candidate-and-push[OF SS' - occs]
     mop-cocc-list-append-mop-occ-list-append occs)
   subgoal using length by auto
   subgoal by auto
   subgoal by auto
   subgoal by (rule length-bounded)
   subgoal by auto
   subgoal using SS' C valid notin-all-occurrences-notin-cocc[OF occs, of C]
     by (auto 9 2 intro!: twl-st-heur-restart-occs-set-occsI
     intro!: valid-occs-append
     simp: push-to-occs-list2-pre-def)
   subgoal using SS' C valid occs by auto
   done
qed

lemma subsumption-cases-lhs:
 assumes
   ‹(a,a')∈Id›
   ‹⋀b b'. a = SUBSUMED-BY b ⟹ a' = SUBSUMED-BY b' ⟹ f b ≤ ⇓S (f' b')›
   ‹⋀b b' c c'. a = STRENGTHENED-BY b c ⟹ a' = STRENGTHENED-BY b' c' ⟹ g b c ≤ ⇓S
(g' b' c')›
   ‹⋀b b' c c'. a = NONE ⟹ a' = NONE ⟹ h ≤ ⇓S h'›
 shows ‹(case a of SUBSUMED-BY b ⇒ f b | STRENGTHENED-BY b c ⇒ g b c | NONE ⇒ h) ≤⇓
S
    (case a' of SUBSUMED-BY b ⇒ f' b| STRENGTHENED-BY b c ⇒ g' b c | NONE ⇒ h')›
 using assms by (auto split: subsumption.splits)
```

definition *arena-promote-st-wl* :: ‹'v twl-st-wl ⇒ nat ⇒ 'v twl-st-wl› **where**
‹*arena-promote-st-wl* = (λ(*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *WS*, *Q*) *C*.
(*M*, *fmupd C* (*N* ∝ *C*, *True*) *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *WS*, *Q*))›

lemma *clss-size-corr-restart-promote*:
‹*clss-size-corr-restart b d* {#} *f g h* {#} *j* {#} (*lcount*) ⟹
¬*irred b C* ⟹ *C* ∈# *dom-m b* ⟹
clss-size-corr-restart (*fmupd C* (*b* ∝ *C*, *True*) *b*) *d* {#} *f g h* {#} *j* {#}
(*clss-size-decr-lcount* (*lcount*))›
unfolding *clss-size-corr-restart-def*
by (*auto simp*: *clss-size-decr-lcount-def clss-size-def*
learned-clss-l-mapsto-upd-in-irrelev size-remove1-mset-If)

lemma *vdom-m-promote-same*:
‹*C* ∈# *dom-m b* ⟹ *vdom-m A m* (*fmupd C* (*b* ∝ *C*, *True*) *b*) = *vdom-m A m* (*b*)›
by (*auto simp*: *vdom-m-def*)

lemma *mop-arena-promote-st-spec*:
assumes *T*: ‹(*S*, *T*) ∈ *twl-st-heur-restart-occs' r u*› **and**
C: ‹*C* ∈# *dom-m* (*get-clauses-wl T*)› **and**
irred: ‹¬*irred* (*get-clauses-wl T*) *C*› **and**
eq: ‹*set-mset* (*all-init-atms-st* (*arena-promote-st-wl T C*)) = *set-mset* (*all-init-atms-st T*)›
shows ‹*mop-arena-promote-st S C* ≤ *SPEC* (λ*U*. (*U*, *arena-promote-st-wl T C*)∈{(*U*,*V*). (*U*,*V*)∈*twl-st-heur-restart-occs*
r u ∧ *get-occs U* = *get-occs S* ∧ *get-aivdom U* = *get-aivdom S*})›
proof −
have *H*: ‹*A* = *B* ⟹ *x* ∈ *A* ⟹ *x* ∈ *B*› **for** *A B x*
by *auto*
have *H'*: ‹*A* = *B* ⟹ *A x* ⟹ *B x*› **for** *A B x*
by *auto*

note *cong* = *trail-pol-cong*[*of* - - ‹(*get-trail-wl-heur S*, *get-trail-wl T*)›]
heuristic-rel-cong[*of* - - ‹*get-heur S*›]
option-lookup-clause-rel-cong[*of* - - ‹(*get-conflict-wl-heur S*, *get-conflict-wl T*)›]
isa-vmtf-cong[*of* - - ‹*get-vmtf-heur S*›]
vdom-m-cong[*THEN H*, *of* - - - ‹*get-watched-wl T*› ‹*get-clauses-wl* (*T*)›]
isasat-input-nempty-cong[*THEN iffD1*]
isasat-input-bounded-cong[*THEN iffD1*]
cach-refinement-empty-cong[*THEN H'*, *of* - - ‹*get-conflict-cach S*›]
phase-saving-cong[*THEN H'*]
$\mathcal{L}_{all}$-*cong*[*THEN H*]
D_0-*cong*[*THEN H*]
map-fun-rel-D_0-*cong*[*of* - - ‹(*get-watched-wl-heur S*, *get-watched-wl T*)›]
vdom-m-cong[*symmetric*, *of* - - ‹*get-watched-wl T*› ‹*get-clauses-wl* (*T*)›]
$\mathcal{L}_{all}$-*cong isasat-input-nempty-cong*
note *cong* = *cong*[*OF eq*[*symmetric*]]
have *valid*: ‹*valid-arena* (*get-clauses-wl-heur S*) (*get-clauses-wl T*) (*set* (*get-vdom S*))› **and**
size: ‹*clss-size-corr-restart* (*get-clauses-wl T*)
(*IsaSAT-Setup.get-unkept-unit-init-clss-wl T*) {#}
(*IsaSAT-Setup.get-kept-unit-init-clss-wl T*)
(*IsaSAT-Setup.get-kept-unit-learned-clss-wl T*) (*get-subsumed-init-clauses-wl T*) {#}
(*get-init-clauses0-wl T*) {#} (*get-learned-count S*)›
using *T* **unfolding** *twl-st-heur-restart-occs-def* **by** *fast+*
then have *irred'*: ‹*arena-status* (*get-clauses-wl-heur S*) *C* ≠ *IRRED*›
using *irred* **by** (*simp add*: *C arena-lifting*(*24*))
have *1*: ‹*1* ≤ *get-learned-count-number S*›

by (*rule red-in-dom-number-of-learned-ge1-twl-st-heur-restart-occs*[*OF T C irred'*])
have *2*: ‹*arena-is-valid-clause-idx* (*get-clauses-wl-heur S*) *C*›
using *C valid* **unfolding** *arena-is-valid-clause-idx-def* **by** *auto*
have *valid'*: ‹*valid-arena* (*arena-set-status* (*get-clauses-wl-heur S*) *C IRRED*)
(*fmupd C* (*get-clauses-wl T* ∝ *C*, *True*) (*get-clauses-wl T*)) (*set* (*get-vdom S*))›
by (*rule valid-arena-arena-set-status*[*OF valid*])
(*use C* **in** *auto*)
show *?thesis*
unfolding *mop-arena-promote-st-def mop-arena-set-status-def*
nres-monad3
apply *refine-vcg*
subgoal by (*rule 1*)
subgoal by (*rule 2*)
subgoal
apply (*cases T*)
using *T C valid' irred vdom-m-promote-same cong*[*unfolded all-init-atms-st-def*] *cong*[*unfolded all-init-atms-st-def*]
by (*auto simp add*: *twl-st-heur-restart-occs-def arena-promote-st-wl-def*
clss-size-corr-restart-promote vdom-m-promote-same simp flip: *learned-clss-count-def*)
done
qed

definition *mark-garbage-wl2* :: ‹*nat* ⇒ *'v twl-st-wl* ⇒ *'v twl-st-wl*› **where**
‹*mark-garbage-wl2* = (λ*C* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *WS*, *Q*).
(*M*, *fmdrop C N*, *D*, *NE*, *UE*, *NEk*, *UEk*, (*if irred N C then add-mset* (*mset* (*N* ∝ *C*)) *else id*) *NS*,
(*if* ¬*irred N C then add-mset* (*mset* (*N* ∝ *C*)) *else id*) *US*, *N0*, *U0*, *WS*, *Q*))›

definition *mark-garbage-wl-no-learned-reset* :: ‹*nat* ⇒ *'v twl-st-wl* ⇒ *'v twl-st-wl*› **where**
‹*mark-garbage-wl-no-learned-reset* = (λ*C* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *WS*, *Q*).
(*M*, *fmdrop C N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *WS*, *Q*))›

definition *add-clauses-to-subsumed-wl* :: ‹*nat* ⇒ *'v twl-st-wl* ⇒ *'v twl-st-wl*› **where**
‹*add-clauses-to-subsumed-wl* = (λ*C* (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *WS*, *Q*).
(*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, (*if irred N C then add-mset* (*mset* (*N* ∝ *C*)) *else id*) *NS*,
(*if* ¬*irred N C then add-mset* (*mset* (*N* ∝ *C*)) *else id*) *US*, *N0*, *U0*, *WS*, *Q*))›

lemma *subsume-or-strengthen-wl-alt-def*[*unfolded Down-id-eq*]:
‹⇓*Id* (*subsume-or-strengthen-wl C s T*) ≥ *do* {
ASSERT(*subsume-or-strengthen-wl-pre C s T*);
(*case s of*
NONE ⇒ *RETURN T*
| *SUBSUMED-BY C'* ⇒ *do* {
let - = *C* ∈#*dom-m* (*get-clauses-wl T*);
let - = *C'* ∈#*dom-m* (*get-clauses-wl T*);
let - = *log-clause T C*;
let U = *mark-garbage-wl2 C T*;
let V = (*if* ¬*irred* (*get-clauses-wl T*) *C'* ∧ *irred* (*get-clauses-wl T*) *C then arena-promote-st-wl U C' else U*);
ASSERT (*set-mset* (*all-init-atms-st V*) = *set-mset* (*all-init-atms-st T*));
let V = *V*;
RETURN V
}
| *STRENGTHENED-BY L C'* ⇒ *strengthen-clause-wl C C' L T*)
}›
proof −
have *subsume-or-strengthen-wl-def*:

‹subsume-or-strengthen-wl = (λC s (M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, Q, W). do {
ASSERT(subsume-or-strengthen-wl-pre C s (M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, Q, W));
(case s of
NONE ⇒ RETURN (M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, Q, W)
| SUBSUMED-BY C' ⇒ do {
let - = ();
let T = (M, fmdrop C (if ¬irred N C' ∧ irred N C then fmupd C' (N ∝ C', True) N else N), D,
NE, UE, NEk, UEk, (if irred N C then add-mset (mset (N ∝ C)) else id) NS,
(if ¬irred N C then add-mset (mset (N ∝ C)) else id) US, N0, U0, Q, W);
ASSERT (set-mset (all-init-atms-st T) = set-mset (all-init-atms-st (M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, Q, W)));
RETURN T
}
| STRENGTHENED-BY L C' ⇒ strengthen-clause-wl C C' L (M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, Q, W))
})›
unfolding *subsume-or-strengthen-wl-def Let-def* **by** *auto*
show *?thesis*
unfolding *subsume-or-strengthen-wl-def*
case-wl-split state-wl-recompose
apply (*refine-vcg subsumption-cases-lhs*)
subgoal by *auto*
subgoal
by (*cases T*)
(*auto simp*: *arena-promote-st-wl-def mark-garbage-wl2-def state-wl-l-def fmdrop-fmupd*
subsume-or-strengthen-wl-pre-def subsume-or-strengthen-pre-def)
subgoal
by (*cases T*)
(*auto simp*: *arena-promote-st-wl-def mark-garbage-wl2-def state-wl-l-def fmdrop-fmupd*
subsume-or-strengthen-wl-pre-def subsume-or-strengthen-pre-def)
subgoal by *auto*
done
qed

lemma *mark-garbage-wl2-simp*[*simp*]:
‹*get-trail-wl* (*mark-garbage-wl2 C S*) = *get-trail-wl S*›
‹*IsaSAT-Setup.get-unkept-unit-init-clss-wl* (*mark-garbage-wl2 C S*) = *IsaSAT-Setup.get-unkept-unit-init-clss-wl S*›
‹*IsaSAT-Setup.get-kept-unit-init-clss-wl* (*mark-garbage-wl2 C S*) = *IsaSAT-Setup.get-kept-unit-init-clss-wl S*›
‹*irred* (*get-clauses-wl S*) *C* ⟹
get-subsumed-init-clauses-wl (*mark-garbage-wl2 C S*) = *add-mset* (*mset* (*get-clauses-wl S* ∝ *C*)) (*get-subsumed-init-clauses-wl S*)›
‹¬*irred* (*get-clauses-wl S*) *C* ⟹
get-subsumed-init-clauses-wl (*mark-garbage-wl2 C S*) = (*get-subsumed-init-clauses-wl S*)›
‹*IsaSAT-Setup.get-unkept-unit-learned-clss-wl* (*mark-garbage-wl2 C S*) = *IsaSAT-Setup.get-unkept-unit-learned-clss-wl S*›
‹*IsaSAT-Setup.get-kept-unit-learned-clss-wl* (*mark-garbage-wl2 C S*) = *IsaSAT-Setup.get-kept-unit-learned-clss-wl S*›
‹*irred* (*get-clauses-wl S*) *C* ⟹
get-subsumed-learned-clauses-wl (*mark-garbage-wl2 C S*) = (*get-subsumed-learned-clauses-wl S*)›
‹¬*irred* (*get-clauses-wl S*) *C* ⟹
get-subsumed-learned-clauses-wl (*mark-garbage-wl2 C S*) = *add-mset* (*mset* (*get-clauses-wl S* ∝ *C*)) (*get-subsumed-learned-clauses-wl S*)›
‹*literals-to-update-wl* (*mark-garbage-wl2 C S*) = *literals-to-update-wl S*›

‹get-watched-wl (mark-garbage-wl2 C S) = get-watched-wl S›
‹get-clauses-wl (mark-garbage-wl2 C S) = fmdrop C (get-clauses-wl S)›
‹get-init-clauses0-wl (mark-garbage-wl2 C S) = get-init-clauses0-wl (S)›
‹get-learned-clauses0-wl (mark-garbage-wl2 C S) = get-learned-clauses0-wl (S)›
‹get-conflict-wl (mark-garbage-wl2 C S) = get-conflict-wl S›
apply (*solves* ‹*cases S*; *auto simp*: *mark-garbage-wl2-def*›)+
done

lemma *mark-garbage-wl2-simp2*[*simp*]:
‹C ∈# dom-m (get-clauses-wl S) ⟹ all-init-atms-st (mark-garbage-wl2 C S) = all-init-atms-st (S)›
by (*cases S*)
(*auto simp*: *mark-garbage-wl2-def all-init-atms-st-def all-init-atms-def all-init-lits-def*
learned-clss-l-mapsto-upd-in-irrelev size-remove1-mset-If init-clss-l-fmdrop-if image-mset-remove1-mset-if
simp del: *all-init-atms-def*[*symmetric*]
cong: *image-mset-cong2 filter-mset-cong2*)

definition *remove-lit-from-clause-wl* :: ‹- ⇒ -› **where**
‹*remove-lit-from-clause-wl C L′* = (λ(*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *W*).
(*M*, *fmupd C* (*remove1 L′* (*N* ∝ *C*)), *irred N C*) *N*, *D*, *NE*, *UE*, *NEk*, *UEk*,
(*if irred N C then add-mset* (*mset* (*N* ∝ *C*)) *else id*) *NS*,
(*if* ¬*irred N C then add-mset* (*mset* (*N* ∝ *C*)) *else id*) *US*, *N0*, *U0*, *Q*, *W*))›

lemma *remove-lit-from-clauses-wl-simp*[*simp*]:
‹*C* ∈# *dom-m* (*get-clauses-wl S*) ⟹ *dom-m* (*get-clauses-wl* (*remove-lit-from-clause-wl C L′ S*)) =
dom-m (*get-clauses-wl S*)›
‹*get-trail-wl* (*remove-lit-from-clause-wl C L′ S*) = *get-trail-wl S*›
‹*IsaSAT-Setup.get-unkept-unit-init-clss-wl* (*remove-lit-from-clause-wl C L′ S*) = *IsaSAT-Setup.get-unkept-unit-init-clss-*
S›
‹*IsaSAT-Setup.get-kept-unit-init-clss-wl* (*remove-lit-from-clause-wl C L′ S*) = *IsaSAT-Setup.get-kept-unit-init-clss-wl*
S›
‹*irred* (*get-clauses-wl S*) *C* ⟹
get-subsumed-init-clauses-wl (*remove-lit-from-clause-wl C L′ S*) = *add-mset* (*mset* (*get-clauses-wl S*
∝ *C*)) (*get-subsumed-init-clauses-wl S*)›
‹¬*irred* (*get-clauses-wl S*) *C* ⟹
get-subsumed-init-clauses-wl (*remove-lit-from-clause-wl C L′ S*) = (*get-subsumed-init-clauses-wl S*)›
‹*IsaSAT-Setup.get-unkept-unit-learned-clss-wl* (*remove-lit-from-clause-wl C L′ S*) = *IsaSAT-Setup.get-unkept-unit-learne*
S›
‹*IsaSAT-Setup.get-kept-unit-learned-clss-wl* (*remove-lit-from-clause-wl C L′ S*) = *IsaSAT-Setup.get-kept-unit-learned-cls*
S›
‹*irred* (*get-clauses-wl S*) *C* ⟹
get-subsumed-learned-clauses-wl (*remove-lit-from-clause-wl C L′ S*) = (*get-subsumed-learned-clauses-wl*
S)›
‹¬*irred* (*get-clauses-wl S*) *C* ⟹
get-subsumed-learned-clauses-wl (*remove-lit-from-clause-wl C L′ S*) = *add-mset* (*mset* (*get-clauses-wl*
S ∝ *C*)) (*get-subsumed-learned-clauses-wl S*)›
‹*literals-to-update-wl* (*remove-lit-from-clause-wl C L′ S*) = *literals-to-update-wl S*›
‹*get-watched-wl* (*remove-lit-from-clause-wl C L′ S*) = *get-watched-wl S*›
‹*get-clauses-wl* (*remove-lit-from-clause-wl C L′ S*) = (*get-clauses-wl S*) (*C* ↪ *remove1 L′* (*get-clauses-wl*
S ∝ *C*))›
‹*get-init-clauses0-wl* (*remove-lit-from-clause-wl C L′ S*) = *get-init-clauses0-wl* (*S*)›
‹*get-learned-clauses0-wl* (*remove-lit-from-clause-wl C L′ S*) = *get-learned-clauses0-wl* (*S*)›
‹*get-conflict-wl* (*remove-lit-from-clause-wl C L′ S*) = *get-conflict-wl S*›
by (*solves* ‹*cases S*; *auto simp*: *remove-lit-from-clause-wl-def*›)+

lemma *in-all-lits-of-wl-ain-atms-of-iff*: ‹*L* ∈# *all-init-lits-of-wl N* ⟷ *atm-of L* ∈# *all-init-atms-st N*›
using $\mathcal{L}_{all}$-*all-init-atms*(2) *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ **by** *blast*

lemma *init-clss-lf-mapsto-upd-irrelev*: ‹*C* ∈# *dom-m N* ⟹ ¬*irred N C* ⟹
init-clss-lf (*fmupd C* (*D*, *True*) *N*) = *add-mset D* (*init-clss-lf N*)›
by (*simp add*: *init-clss-l-mapsto-upd-irrelev*)

lemma *arena-promote-dom-m-get-clauses-wl*[*simp*]:
‹*C* ∈# *dom-m* (*get-clauses-wl S*) ⟹
dom-m (*get-clauses-wl* (*arena-promote-st-wl S C*)) = *dom-m* (*get-clauses-wl S*)›
by (*cases S*) (*auto simp*: *arena-promote-st-wl-def*)

The assertions here are an artefact of how the refinement frameworks handles if-then-else. It splits lhs ifs, but not rhs splits when they appear whithin an expression.

lemma *strengthen-clause-wl-alt-def*[*unfolded Down-id-eq*]:
‹⇓*Id*(*strengthen-clause-wl C D L′ S*) ≥ *do* {
ASSERT (*subsume-or-strengthen-wl-pre C* (*STRENGTHENED-BY L′ D*) *S*);
let m = *length* (*get-clauses-wl S* ∝ *C*);
let n = *length* (*get-clauses-wl S* ∝ *D*);
let E = *remove1* (− *L′*) (*get-clauses-wl S* ∝ *C*);
let - = *C* ∈# *dom-m* (*get-clauses-wl S*);
let - = *D* ∈# *dom-m* (*get-clauses-wl S*);
let T = *remove-lit-from-clause-wl C* (−*L′*) *S*;
- ← *log-clause2 T C*;
if False then RETURN T
else if m = *n then do* {
let T = *add-clauses-to-subsumed-wl D* (*T*);
ASSERT (*set-mset* (*all-init-atms-st T*) = *set-mset* (*all-init-atms-st S*));
ASSERT (*set-mset* (*all-init-atms-st* (*if* ¬*irred* (*get-clauses-wl S*) *C* ∧ *irred* (*get-clauses-wl S*) *D* *then arena-promote-st-wl T C else T*)) = *set-mset* (*all-init-atms-st S*));
let U = (*if* ¬*irred* (*get-clauses-wl S*) *C* ∧ *irred* (*get-clauses-wl S*) *D then arena-promote-st-wl T C* *else T*);
ASSERT (*set-mset* (*all-init-atms-st* (*mark-garbage-wl-no-learned-reset D U*)) = *set-mset* (*all-init-atms-st* *S*));
let U = *mark-garbage-wl-no-learned-reset D U*;
RETURN U
}
else RETURN T
}›
proof −
have *le2*: ‹*length* (*get-clauses-wl S* ∝ *C*) ≠ *2*› **and**
CD: ‹*C* ≠ *D*› **and**
C-dom: ‹*C* ∈# *dom-m* (*get-clauses-wl S*)›**and**
D-dom: ‹*D* ∈# *dom-m* (*get-clauses-wl S*)› **and**
subs: ‹*remove1-mset L′* (*mset* (*get-clauses-wl S* ∝ *D*)) ⊆# *remove1-mset* (− *L′*) (*mset* (*get-clauses-wl* *S* ∝ *C*))› **and**
eq-le: ‹*length* (*get-clauses-wl S* ∝ *D*) = *length* (*get-clauses-wl S* ∝ *C*) ⟹
remove1-mset L′ (*mset* (*get-clauses-wl S* ∝ *D*)) = *remove1-mset* (− *L′*) (*mset* (*get-clauses-wl S* ∝ *C*))›
if *pre*: ‹*subsume-or-strengthen-wl-pre C* (*STRENGTHENED-BY L′ D*) *S*›
proof −
have
L: ‹−*L′* ∈# *mset* (*get-clauses-wl S* ∝ *C*)›
‹*L′* ∈# *mset* (*get-clauses-wl S* ∝ *D*)› **and**
‹*remove1-mset L′* (*mset* (*get-clauses-wl S* ∝ *D*)) ⊆# *remove1-mset* (− *L′*) (*mset* (*get-clauses-wl S* ∝ *C*))›
‹¬ *tautology* (*mset* (*get-clauses-wl S* ∝ *D*))›
‹¬ *tautology*
(*remove1-mset L′* (*mset* (*get-clauses-wl S* ∝ *D*)) +

```
      remove1-mset (− L') (mset (get-clauses-wl S ∝ C))›
    using that unfolding subsume-or-strengthen-wl-pre-def
      subsume-or-strengthen-pre-def apply −
    by (solves ‹normalize-goal+; clarsimp›)+
  show ‹length (get-clauses-wl S ∝ C) ≠ 2› and ‹C ≠ D› and ‹C ∈# dom-m (get-clauses-wl S)› and
    ‹D ∈# dom-m (get-clauses-wl S)› and
    subs: ‹remove1-mset L' (mset (get-clauses-wl S ∝ D)) ⊆# remove1-mset (− L') (mset (get-clauses-wl S ∝ C))›
    using that unfolding subsume-or-strengthen-wl-pre-def
      subsume-or-strengthen-pre-def apply −
    by (solves ‹normalize-goal+; clarsimp›)+

  show ‹remove1-mset L' (mset (get-clauses-wl S ∝ D)) = remove1-mset (− L') (mset (get-clauses-wl S ∝ C))›
    if ‹length (get-clauses-wl S ∝ D) = length (get-clauses-wl S ∝ C)›
    using multi-member-split[OF L(1)] multi-member-split[OF L(2)] subs that
    by (auto simp del: size-mset simp flip: size-mset simp: subseteq-mset-size-eql-iff)

 qed
 have add-subsumed-same:
   ‹set-mset (all-init-atms-st (add-clauses-to-subsumed-wl D S)) =  set-mset (all-init-atms-st S)›
   if ‹D ∈# dom-m (get-clauses-wl S)› for S D
   using that
   apply (cases S)
   apply (auto simp: all-init-atms-def all-init-atms-st-def add-clauses-to-subsumed-wl-def
     all-init-lits-def ran-m-def all-lits-of-mm-add-mset
     dest!: multi-member-split[of D]
     simp del: all-init-atms-def[symmetric])
   done

 have H1: ‹set-mset (all-init-atms-st (arena-promote-st-wl S C)) = set-mset (all-init-atms-st S)›
   if ‹set-mset (all-lits-of-m (mset (get-clauses-wl S ∝ C))) ⊆ set-mset (all-init-lits-of-wl S)› and
     ‹C ∈# dom-m (get-clauses-wl S)›
   for C S
   using that
   apply (cases ‹irred (get-clauses-wl S) C›)
   subgoal
     using fmupd-same[of C ‹get-clauses-wl S›]
     apply (cases S; cases ‹fmlookup (get-clauses-wl S) C›)
     apply (auto simp:  all-init-atms-st-def add-clauses-to-subsumed-wl-def
       all-init-atms-def all-init-lits-def ran-m-def all-lits-of-mm-add-mset arena-promote-st-wl-def
       remove-lit-from-clause-wl-def image-Un
       dest!: multi-member-split[of D]
       simp del: all-init-atms-def[symmetric] fmupd-same
       cong: filter-mset-cong image-mset-cong2)
     apply auto
    done
   subgoal
     by (cases S)
      (auto simp:  all-init-atms-st-def add-clauses-to-subsumed-wl-def all-init-atms-def
       all-init-lits-def all-lits-of-mm-add-mset arena-promote-st-wl-def init-clss-l-mapsto-upd
       init-clss-l-mapsto-upd-irrelev all-init-lits-of-wl-def ac-simps
       dest!: multi-member-split[of D]
       simp del: all-init-atms-def[symmetric])
   done
```

have K: ‹*set-mset*
(*all-init-atms-st*
(*arena-promote-st-wl* (*add-clauses-to-subsumed-wl D* (*remove-lit-from-clause-wl C* (− *L′*) *S*)) *C*))
=
set-mset (*all-init-atms-st S*)› (**is** *?A*) **and**
K2: ‹*set-mset* (*all-init-atms-st* (*remove-lit-from-clause-wl C* (− *L′*) *S*)) = *set-mset* (*all-init-atms-st S*)› (**is** *?B*) **and**
K3: ‹*length* (*get-clauses-wl S* ∝ *C*) = *length* (*get-clauses-wl S* ∝ *D*) ⟹
set-mset (*all-init-atms-st* (*mark-garbage-wl-no-learned-reset D*
(*if* ¬ *irred* (*get-clauses-wl S*) *C* ∧ *irred* (*get-clauses-wl S*) *D*
then arena-promote-st-wl (*add-clauses-to-subsumed-wl D* (*remove-lit-from-clause-wl C* (− *L′*) *S*)) *C*
else add-clauses-to-subsumed-wl D (*remove-lit-from-clause-wl C* (− *L′*) *S*)))) =
set-mset (*all-init-atms-st S*)› (**is** ‹- ⟹ *?C*›)
if *pre*: ‹*subsume-or-strengthen-wl-pre C* (*STRENGTHENED-BY L′ D*) *S*›
proof −
obtain *x xa* **where**
Sx: ‹(*S*, *x*) ∈ *state-wl-l None*› **and**
‹2 < *length* (*get-clauses-wl S* ∝ *C*)› **and**
‹2 ≤ *length* (*get-clauses-l x* ∝ *C*)› **and**
‹*C* ∈# *dom-m* (*get-clauses-l x*)› **and**
‹*count-decided* (*get-trail-l x*) = 0› **and**
‹*distinct* (*get-clauses-l x* ∝ *C*)› **and**
‹∀ *L*∈*set* (*get-clauses-l x* ∝ *C*). *undefined-lit* (*get-trail-l x*) *L*› **and**
‹*get-conflict-l x* = *None*› **and**
‹*C* ∉ *set* (*get-all-mark-of-propagated* (*get-trail-l x*))› **and**
‹*clauses-to-update-l x* = {#}› **and**
‹*twl-list-invs x*› **and**
xxa: ‹(*x*, *xa*) ∈ *twl-st-l None*› **and**
struct: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*state*$_W$*-of xa*)› **and**
inv: ‹*twl-struct-invs xa*› **and**
‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-learned-clauses-entailed-by-init* (*state*$_W$*-of xa*)› **and**
‹*L′* ∈# *mset* (*get-clauses-l x* ∝ *D*)› **and**
‹− *L′* ∈# *mset* (*get-clauses-l x* ∝ *C*)› **and**
‹¬ *tautology* (*mset* (*get-clauses-l x* ∝ *D*))› **and**
‹*D* ≠ 0› **and**
‹*remove1-mset L′* (*mset* (*get-clauses-l x* ∝ *D*)) ⊆# *remove1-mset* (− *L′*) (*mset* (*get-clauses-l x* ∝ *C*))› **and**
‹*D* ∈# *dom-m* (*get-clauses-l x*)› **and**
‹*distinct* (*get-clauses-l x* ∝ *D*)› **and**
‹*D* ∉ *set* (*get-all-mark-of-propagated* (*get-trail-l x*))› **and**
‹2 ≤ *length* (*get-clauses-l x* ∝ *D*)› **and**
‹¬ *tautology*
(*remove1-mset L′* (*mset* (*get-clauses-l x* ∝ *D*)) +
remove1-mset (− *L′*) (*mset* (*get-clauses-l x* ∝ *C*)))›
using *pre* **unfolding** *subsume-or-strengthen-wl-pre-def subsume-or-strengthen-pre-def subsumption.simps*
apply − **apply** *normalize-goal*+ **by** *blast*

have ‹*cdcl*$_W$*-restart-mset.no-strange-atm* (*state*$_W$*-of xa*)›
using *struct* **unfolding** *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def* **by** *fast*
then have ‹*atms-of-mm* (*learned-clss* (*state*$_W$*-of xa*)) ⊆ *atms-of-mm* (*init-clss* (*state*$_W$*-of xa*))›
unfolding *cdcl*$_W$*-restart-mset.no-strange-atm-def* **by** *fast*
then have *lits-C-in-all*:
‹*set-mset* (*all-lits-of-m* (*mset* (*get-clauses-wl S* ∝ *C*))) ⊆ *set-mset* (*all-init-lits-of-wl S*)›
if ‹¬*irred* (*get-clauses-wl S*) *C*›
using *that C-dom*[*OF pre*] *Sx xxa* **unfolding** *set-eq-iff in-set-all-atms-iff*

in-set-all-atms-iff in-set-all-init-atms-iff get-unit-clauses-wl-alt-def
by (*cases xa*; *cases x*; *cases S*)
(*auto 4 3 simp*: *twl-st-l-def state-wl-l-def mset-take-mset-drop-mset′ ran-m-def image-image*
conj-disj-distribR Collect-disj-eq image-Un Collect-conv-if all-init-lits-of-wl-def
in-all-lits-of-mm-ain-atms-of-iff in-all-lits-of-m-ain-atms-of-iff atm-of-eq-atm-of
dest!: *multi-member-split*[*of* ‹*C* :: *nat*›])
have [*simp*]: ‹*get-clauses-wl* (*add-clauses-to-subsumed-wl D* (*remove-lit-from-clause-wl C* (− *L′*) *S*)) ∝ *C* =
get-clauses-wl (*remove-lit-from-clause-wl C* (− *L′*) *S*) ∝ *C*›
by (*cases S*) (*auto simp*: *add-clauses-to-subsumed-wl-def remove-lit-from-clause-wl-def*)

have *init-decomp*:
‹*irred N D* ⟹ *D* ∈# *dom-m N* ⟹ *init-clss-l N* = *add-mset* ((*N* ∝ *D*, *irred N D*)) (*init-clss-l* (*fmdrop D N*))› **for** *D N*
using *distinct-mset-dom*[*of N*] **apply** (*cases* ‹*the* (*fmlookup N D*)›)
by (*auto simp*: *ran-m-def dest*!: *multi-member-split*
intro!: *image-mset-cong2 intro*!: *filter-mset-cong2*)

have [*simp*]: ‹*fmdrop C* (*fmdrop D* (*fmupd C E b*)) = *fmdrop C* (*fmdrop D b*)› **for** *E b*
by (*metis fmdrop-comm fmdrop-fmupd-same*)

show *?A*
using *C-dom*[*OF that*] *D-dom*[*OF that*] *CD*[*OF that*] *subs*[*OF that*]
all-lits-of-m-mono[*of* ‹*mset* (*remove1* (−*L′*) (*get-clauses-wl S* ∝ *C*))›
‹*mset* (*get-clauses-wl S* ∝ *C*)›, *THEN set-mset-mono*]
all-lits-of-m-mono[*of* ‹*mset* (*remove1* (−*L′*) (*get-clauses-wl S* ∝ *D*))›
‹*mset* (*get-clauses-wl S* ∝ *D*)›, *THEN set-mset-mono*]
apply (*cases S*)
apply (*auto simp*: *remove-lit-from-clause-wl-def add-clauses-to-subsumed-wl-def*
arena-promote-st-wl-def all-init-atms-def all-init-atms-st-def all-init-lits-def image-Un fmdrop-fmupd-same
all-lits-of-mm-add-mset init-clss-l-mapsto-upd init-decomp[*of - D*] *init-decomp*[*of* ‹*fmdrop D -› C*]
init-clss-l-fmdrop-irrelev
simp del: *all-init-atms-def*[*symmetric*] *dest*!: *multi-member-split*[*of D*])
apply (*auto simp*: *remove-lit-from-clause-wl-def add-clauses-to-subsumed-wl-def*
arena-promote-st-wl-def all-init-atms-def all-init-atms-st-def all-init-lits-def image-Un fmdrop-fmupd-same
all-lits-of-mm-add-mset init-clss-l-mapsto-upd init-decomp[*of - D*] *init-decomp*[*of* ‹-› *C*] *init-clss-l-fmdrop-irrelev*
simp del: *all-init-atms-def*[*symmetric*] *dest*!: *multi-member-split*[*of D*])
using *lits-C-in-all*[*unfolded all-init-lits-of-wl-def*, *simplified*]
apply (*auto simp*: *remove-lit-from-clause-wl-def add-clauses-to-subsumed-wl-def*
arena-promote-st-wl-def all-init-atms-def all-init-atms-st-def all-init-lits-def image-Un fmdrop-fmupd-same
all-lits-of-mm-add-mset init-clss-l-mapsto-upd init-decomp[*of - D*] *init-decomp*[*of* ‹-› *C*] *ac-simps*
simp del: *all-init-atms-def*[*symmetric*] *dest*!: *multi-member-split*[*of D*])
done

show *?B*
using *C-dom*[*OF that*] *D-dom*[*OF that*] *CD*[*OF that*] *subs*[*OF that*]
all-lits-of-m-mono[*of* ‹*mset* (*remove1* (−*L′*) (*get-clauses-wl S* ∝ *C*))›
‹*mset* (*get-clauses-wl S* ∝ *C*)›, *THEN set-mset-mono*]
all-lits-of-m-mono[*of* ‹*mset* (*remove1* (−*L′*) (*get-clauses-wl S* ∝ *D*))›
‹*mset* (*get-clauses-wl S* ∝ *D*)›, *THEN set-mset-mono*]
apply (*cases S*)
apply (*auto simp*: *remove-lit-from-clause-wl-def add-clauses-to-subsumed-wl-def*
arena-promote-st-wl-def all-init-atms-def all-init-atms-st-def all-init-lits-def image-Un fmdrop-fmupd-same
all-lits-of-mm-add-mset init-clss-l-mapsto-upd init-decomp[*of - D*] *init-decomp*[*of* ‹*fmdrop D -› C*]
init-clss-l-fmdrop-irrelev
simp del: *all-init-atms-def*[*symmetric*] *dest*!: *multi-member-split*[*of D*])

apply (*auto simp*: *remove-lit-from-clause-wl-def add-clauses-to-subsumed-wl-def*
arena-promote-st-wl-def all-init-atms-def all-init-atms-st-def all-init-lits-def image-Un fmdrop-fmupd-same
all-lits-of-mm-add-mset init-clss-l-mapsto-upd init-decomp[*of - D*] *init-decomp*[*of* ‹-› *C*] *init-clss-l-fmdrop-irrelev init-clss-l-mapsto-upd-irrel*
simp del: *all-init-atms-def*[*symmetric*] *dest*!: *multi-member-split*[*of D*])
done
have *KK*[*simp*]: ‹*E* ∈# *b* ⟹ *add-mset* (*mset E*) (*mset* ‘# *remove1-mset E b* + *F*) = *mset* ‘# *b* + *F*› **for** *b E F*
by (*auto dest*!: *multi-member-split*)

have *1*: ‹*set-mset* (*all-init-atms-st* (*mark-garbage-wl-no-learned-reset D* (*add-clauses-to-subsumed-wl D S*))) =
set-mset (*all-init-atms-st S*)›
if ‹*D* ∈# *dom-m* (*get-clauses-wl S*)› **for** *S*
using *that*
apply (*cases S*)
apply (*clarsimp simp only*: *remove-lit-from-clause-wl-def add-clauses-to-subsumed-wl-def*
all-lits-of-mm-add-mset init-clss-l-mapsto-upd init-decomp[*of - D*]
init-clss-l-fmdrop-irrelev mark-garbage-wl-no-learned-reset-def arena-promote-st-wl-def
all-init-atms-st-def all-init-atms-def get-clauses-wl.simps all-init-lits-def fmupd-idem
init-clss-l-mapsto-upd-irrel init-clss-lf-fmdrop-irrelev init-clss-lf-mapsto-upd-irrelev
split: *if-splits intro*!: *impI conjI*
simp del: *all-init-atms-def*[*symmetric*] *dest*!: *multi-member-split*[*of D*])
apply (*intro conjI impI*)
apply *simp*
apply *simp*
apply (*subst* (*2*)*init-decomp*[*of - D*])
apply (*auto simp add*: *all-lits-of-mm-add-mset*)[*3*]
apply (*subst* (*2*)*init-decomp*[*of - D*])
apply (*auto simp add*: *all-lits-of-mm-add-mset*)[*3*]
done

have *2*: ‹*set-mset* (*all-init-atms-st* (*remove-lit-from-clause-wl C* (− *L′*) *S*)) =
set-mset (*all-init-atms-st S*)› **if** ‹*C* ∈# *dom-m* (*get-clauses-wl S*)›**for** *S*
using *that*
all-lits-of-m-mono[*of* ‹*mset* (*remove1* (−*L′*) (*get-clauses-wl S* ∝ *C*))›
‹*mset* (*get-clauses-wl S* ∝ *C*)›, *THEN set-mset-mono*]
apply (*cases S*)
apply (*auto simp*: *remove-lit-from-clause-wl-def add-clauses-to-subsumed-wl-def*
arena-promote-st-wl-def all-init-atms-def all-init-atms-st-def all-init-lits-def image-Un fmdrop-fmupd-same
all-lits-of-mm-add-mset init-clss-l-mapsto-upd init-decomp[*of - C*]
init-clss-l-fmdrop-irrelev init-clss-l-mapsto-upd-irrel
simp del: *all-init-atms-def*[*symmetric*] *dest*!: *multi-member-split*[*of C*])
done

have *init-decomp*:
‹*NO-MATCH* (*fmdrop D N′*) *N* ⟹ *irred N D* ⟹ *D* ∈# *dom-m N* ⟹ *init-clss-l N* = *add-mset* ((*N* ∝ *D*, *irred N D*)) (*init-clss-l* (*fmdrop D N*))› **for** *D N*
using *distinct-mset-dom*[*of N*] **apply** (*cases* ‹*the* (*fmlookup N D*)›)
by (*auto simp*: *ran-m-def dest*!: *multi-member-split*
intro!: *image-mset-cong2 intro*!: *filter-mset-cong2*)
assume ‹*length* (*get-clauses-wl S* ∝ *C*) = *length* (*get-clauses-wl S* ∝ *D*)›
note *eq* = *eq-le*[*OF pre this*[*symmetric*]]
have *3*: ‹*irred* (*get-clauses-wl S*) *D* ⟹
(*mark-garbage-wl-no-learned-reset D*
(*arena-promote-st-wl* (*add-clauses-to-subsumed-wl D* (*remove-lit-from-clause-wl C* (− *L′*) *S*)) *C*)) =

```
(mark-garbage-wl-no-learned-reset D
    (add-clauses-to-subsumed-wl D (arena-promote-st-wl (remove-lit-from-clause-wl C (− L′) S) C)))›
  apply (cases S)
  apply (auto simp: mark-garbage-wl-no-learned-reset-def arena-promote-st-wl-def
add-clauses-to-subsumed-wl-def remove-lit-from-clause-wl-def)
  done
 have ‹set-mset (all-init-atms-st (arena-promote-st-wl (remove-lit-from-clause-wl C (− L′) S) C)) =
  set-mset (all-init-atms-st S)›
  if ‹¬irred (get-clauses-wl S) C› ‹irred (get-clauses-wl S) D›
  using that D-dom[OF pre] C-dom[OF pre] eq[symmetric]
    all-lits-of-m-mono[of ‹mset (remove1 (L′) (get-clauses-wl S ∝ D))›
      ‹mset (get-clauses-wl S ∝ D)›, THEN set-mset-mono]
  apply (cases S)
  apply (auto simp: remove-lit-from-clause-wl-def add-clauses-to-subsumed-wl-def
  arena-promote-st-wl-def all-init-atms-def all-init-atms-st-def all-init-lits-def image-Un fmdrop-fmupd-same
    all-lits-of-mm-add-mset init-clss-l-mapsto-upd init-clss-l-mapsto-upd-irrel
    init-clss-l-fmdrop-irrelev init-clss-l-mapsto-upd-irrel all-init-atms-st-def
    init-clss-l-mapsto-upd-irrel init-clss-lf-mapsto-upd-irrelev
    simp del: all-init-atms-def[symmetric] dest!: )
  apply (subst init-decomp[of D])
  apply (auto simp add: all-lits-of-mm-add-mset image-Un)
  done
 then show ?C
   using C-dom[OF that] D-dom[OF that] CD[OF that] subs[OF that]
    all-lits-of-m-mono[of ‹mset (remove1 (−L′) (get-clauses-wl S ∝ C))›
      ‹mset (get-clauses-wl S ∝ C)›, THEN set-mset-mono]
    all-lits-of-m-mono[of ‹mset (remove1 (−L′) (get-clauses-wl S ∝ D))›
      ‹mset (get-clauses-wl S ∝ D)›, THEN set-mset-mono]
   apply (cases ‹¬ irred (get-clauses-wl S) C ∧ irred (get-clauses-wl S) D›)
   subgoal
    apply (simp only: if-True simp-thms)
    apply (simp add: 1 2 3)
    done
   subgoal
    apply (simp only: if-False 1)
    apply (auto simp: 2 1)
    done
  done
qed

 have strengthen-clause-wl-alt-def:
  ‹strengthen-clause-wl = (λC C′ L (M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, Q, W). do {
  ASSERT (subsume-or-strengthen-wl-pre C (STRENGTHENED-BY L C′) (M, N, D, NE, UE, NEk,
UEk, NS, US, N0, U0, Q, W));
  E ← SPEC (λE. mset E = mset (remove1 (−L) (N ∝ C)));
  let - = ();
  let - = ();
  let - = ();
  - ← RETURN ();
  if length (N ∝ C) = 2
  then do {
    ASSERT (length (remove1 (−L) (N ∝ C)) = 1);
    let L = hd E;
     RETURN (Propagated L 0 # M, fmdrop C′ (fmdrop C N), D,
     (if irred N C′ then add-mset (mset (N ∝ C′)) else id) NE,
     (if ¬irred N C′ then add-mset (mset (N ∝ C′)) else id) UE,
```

```
        (if irred N C then add-mset {#L#} else id) NEk, (if ¬irred N C then add-mset {#L#} else
id) UEk,
        ((if irred N C then add-mset (mset (N ∝ C)) else id)) NS,
        ((if ¬irred N C then add-mset (mset (N ∝ C)) else id)) US,
        N0, U0, add-mset (−L) Q, W)
    }
    else if length (N ∝ C) = length (N ∝ C')
    then RETURN (M, fmdrop C' (fmupd C (E, irred N C ∨ irred N C') N), D, NE, UE, NEk, UEk,
      ((if irred N C' then add-mset (mset (N ∝ C')) else id)  o (if irred N C then add-mset (mset (N
∝ C)) else id)) NS,
     ((if ¬irred N C' then add-mset (mset (N ∝ C')) else id) o (if ¬irred N C then add-mset (mset (N
∝ C)) else id)) US,
      N0, U0, Q, W)
    else RETURN (M, fmupd C (E, irred N C) N, D, NE, UE, NEk, UEk,
     (if irred N C then add-mset (mset (N ∝ C)) else id) NS,
     (if ¬irred N C then add-mset (mset (N ∝ C)) else id) US, N0, U0, Q, W)})›
     unfolding strengthen-clause-wl-def Let-def by auto
  have [refine0]: ‹subsume-or-strengthen-wl-pre C (STRENGTHENED-BY L' D) S ⟹
    subsume-or-strengthen-wl-pre C (STRENGTHENED-BY L' D) S ⟹
    (remove1 (− L') (get-clauses-wl S ∝ C), E) ∈ Id ⟹
    log-clause2 (remove-lit-from-clause-wl C (− L') S) C
   ≤ Refine-Basic.SPEC (λc. (c, ()) ∈ Id)› for E
   by (rule log-clause2-log-clause[THEN fref-to-Down-curry, THEN order-trans]) auto

  show ?thesis
   unfolding strengthen-clause-wl-alt-def case-wl-split state-wl-recompose Let-def [of ‹length -›]
   apply (refine-vcg )
   subgoal by auto
   apply assumption
   subgoal using le2 by blast
   subgoal by auto
   subgoal by auto
   subgoal using D-dom by (clarsimp simp add: add-subsumed-same K K2)
   subgoal using D-dom by (clarsimp simp add: add-subsumed-same K K2)
   subgoal using D-dom by (clarsimp simp add: add-subsumed-same K3)
   subgoal
    using CD
    by (cases S)
     (auto simp: mark-garbage-wl-no-learned-reset-def add-clauses-to-subsumed-wl-def
      arena-promote-st-wl-def remove-lit-from-clause-wl-def)
   subgoal
    by (cases S) (auto simp: remove-lit-from-clause-wl-def add-clauses-to-subsumed-wl-def)
   done
qed


fun set-clauses-wl :: ‹- ⇒ 'v twl-st-wl ⇒ 'v twl-st-wl› where
  ‹set-clauses-wl N (M, -, D, NE, UE, NEk, UEk, NS, US, N0, U0, WS, Q) =
    (M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, WS, Q) ›
fun add-clause-to-subsumed :: ‹- ⇒ - ⇒ 'v twl-st-wl ⇒ 'v twl-st-wl› where
  ‹add-clause-to-subsumed b E (M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, WS, Q) =
    (M, N, D, NE, UE, NEk, UEk, (if b then add-mset E else id) NS,
    (if ¬b then add-mset E else id) US, N0, U0, WS, Q) ›


lemma fmap-eq-iff-dom-m-lookup: ‹f = g ⟷ (dom-m f = dom-m g ∧ (∀ k∈#dom-m f. fmlookup f k
```

= *fmlookup g k*))›
by (*metis fmap-ext in-dom-m-lookup-iff*)

lemma *mop-arena-shorten*:
assumes ‹*valid-arena N N′ vdom*› **and**
‹(*i*,*j*)∈*nat-rel*› **and**
‹(*C*,*C′*)∈*nat-rel*› **and**
‹*C* ∈# *dom-m N′*› **and**
‹*i*≥2› ‹*i* ≤ *length* (*N′* ∝ *C*)›
shows
‹*mop-arena-shorten C i N*
≤ *SPEC* (λ*c*. (*c*, *N′*(*C′* ↪ *take j* (*N′* ∝ *C′*)))
∈ {(*N*$_1$,*N*$_1$′). *valid-arena N*$_1$ *N*$_1$′ *vdom* ∧ *length N*$_1$ = *length N*})›
proof −
show *?thesis*
unfolding *mop-arena-shorten-def*
apply *refine-vcg*
subgoal using *assms* **unfolding** *arena-shorten-pre-def arena-is-valid-clause-idx-def*
by (*auto intro*!: *exI*[*of* - *N′*] *simp*: *arena-lifting*)
subgoal
using *assms* **by** (*auto intro*!: *valid-arena-arena-shorten simp*: *arena-lifting*)
done
qed

lemma *count-list-distinct-If*: ‹*distinct xs* ⟹ *count-list xs x* = (*if x* ∈ *set xs then 1 else 0*)›
by (*simp add*: *count-mset-count-list distinct-count-atmost-1*)

lemma *set-clauses-wl-simp*[*simp*]:
‹*get-trail-wl* (*set-clauses-wl N S*) = *get-trail-wl S*›
‹*IsaSAT-Setup.get-unkept-unit-init-clss-wl* (*set-clauses-wl N S*) = *IsaSAT-Setup.get-unkept-unit-init-clss-wl S*›
‹*IsaSAT-Setup.get-kept-unit-init-clss-wl* (*set-clauses-wl N S*) = *IsaSAT-Setup.get-kept-unit-init-clss-wl S*›
‹*get-subsumed-init-clauses-wl* (*set-clauses-wl N S*) = (*get-subsumed-init-clauses-wl S*)›
‹*get-subsumed-init-clauses-wl* (*set-clauses-wl N S*) = (*get-subsumed-init-clauses-wl S*)›
‹*IsaSAT-Setup.get-unkept-unit-learned-clss-wl* (*set-clauses-wl N S*) = *IsaSAT-Setup.get-unkept-unit-learned-clss-wl S*›
‹*IsaSAT-Setup.get-kept-unit-learned-clss-wl* (*set-clauses-wl N S*) = *IsaSAT-Setup.get-kept-unit-learned-clss-wl S*›
‹*get-subsumed-learned-clauses-wl* (*set-clauses-wl N S*) = (*get-subsumed-learned-clauses-wl S*)›
‹*get-subsumed-learned-clauses-wl* (*set-clauses-wl N S*) = (*get-subsumed-learned-clauses-wl S*)›
‹*literals-to-update-wl* (*set-clauses-wl N S*) = *literals-to-update-wl S*›
‹*get-watched-wl* (*set-clauses-wl N S*) = *get-watched-wl S*›
‹*get-clauses-wl* (*set-clauses-wl N S*) = *N*›
‹*get-init-clauses0-wl* (*set-clauses-wl N S*) = *get-init-clauses0-wl* (*S*)›
‹*get-learned-clauses0-wl* (*set-clauses-wl N S*) = *get-learned-clauses0-wl* (*S*)›
‹*get-conflict-wl* (*set-clauses-wl N S*) = *get-conflict-wl S*›
apply (*solves* ‹*cases S*; *auto simp*: ›)+
done

lemma *add-clause-to-subsumed-simp*[*simp*]:
‹*get-trail-wl* (*add-clause-to-subsumed b N S*) = *get-trail-wl S*›
‹*IsaSAT-Setup.get-unkept-unit-init-clss-wl* (*add-clause-to-subsumed b N S*) = *IsaSAT-Setup.get-unkept-unit-init-clss-wl S*›
‹*IsaSAT-Setup.get-kept-unit-init-clss-wl* (*add-clause-to-subsumed b N S*) = *IsaSAT-Setup.get-kept-unit-init-clss-wl*

S›
‹*b* ⟹ *get-subsumed-init-clauses-wl* (*add-clause-to-subsumed b N S*) = *add-mset N* (*get-subsumed-init-clauses-wl* *S*)›
‹¬*b* ⟹ *get-subsumed-init-clauses-wl* (*add-clause-to-subsumed b N S*) = (*get-subsumed-init-clauses-wl* *S*)›
‹*IsaSAT-Setup.get-unkept-unit-learned-clss-wl* (*add-clause-to-subsumed b N S*) = *IsaSAT-Setup.get-unkept-unit-learned-* *S*›
‹*IsaSAT-Setup.get-kept-unit-learned-clss-wl* (*add-clause-to-subsumed b N S*) = *IsaSAT-Setup.get-kept-unit-learned-clss-w* *S*›
‹*b* ⟹
get-subsumed-learned-clauses-wl (*add-clause-to-subsumed b N S*) = (*get-subsumed-learned-clauses-wl* *S*)›
‹¬*b* ⟹ *get-subsumed-learned-clauses-wl* (*add-clause-to-subsumed b N S*) = *add-mset N* (*get-subsumed-learned-clauses-w* *S*)›
‹*literals-to-update-wl* (*add-clause-to-subsumed b N S*) = *literals-to-update-wl S*›
‹*get-watched-wl* (*add-clause-to-subsumed b N S*) = *get-watched-wl S*›
‹*get-clauses-wl* (*add-clause-to-subsumed b N S*) = *get-clauses-wl S*›
‹*get-init-clauses0-wl* (*add-clause-to-subsumed b N S*) = *get-init-clauses0-wl* (*S*)›
‹*get-learned-clauses0-wl* (*add-clause-to-subsumed b N S*) = *get-learned-clauses0-wl* (*S*)›
‹*get-conflict-wl* (*add-clause-to-subsumed b N S*) = *get-conflict-wl S*›
apply (*solves* ‹*cases S*; *auto simp*: ›)+
done

lemma *add-clauses-to-subsumed-wl-simp*[*simp*]:
‹*get-trail-wl* (*add-clauses-to-subsumed-wl N S*) = *get-trail-wl S*›
‹*IsaSAT-Setup.get-unkept-unit-init-clss-wl* (*add-clauses-to-subsumed-wl N S*) = *IsaSAT-Setup.get-unkept-unit-init-clss-w* *S*›
‹*IsaSAT-Setup.get-kept-unit-init-clss-wl* (*add-clauses-to-subsumed-wl N S*) = *IsaSAT-Setup.get-kept-unit-init-clss-wl* *S*›
‹*irred* (*get-clauses-wl S*) *N* ⟹
get-subsumed-init-clauses-wl (*add-clauses-to-subsumed-wl N S*) = *add-mset* (*mset* (*get-clauses-wl S* ∝ *N*)) (*get-subsumed-init-clauses-wl S*)›
‹¬*irred* (*get-clauses-wl S*) *N* ⟹
get-subsumed-init-clauses-wl (*add-clauses-to-subsumed-wl N S*) = (*get-subsumed-init-clauses-wl S*)›
‹*IsaSAT-Setup.get-unkept-unit-learned-clss-wl* (*add-clauses-to-subsumed-wl N S*) = *IsaSAT-Setup.get-unkept-unit-learne* *S*›
‹*IsaSAT-Setup.get-kept-unit-learned-clss-wl* (*add-clauses-to-subsumed-wl N S*) = *IsaSAT-Setup.get-kept-unit-learned-clss* *S*›
‹*irred* (*get-clauses-wl S*) *N* ⟹
get-subsumed-learned-clauses-wl (*add-clauses-to-subsumed-wl N S*) = (*get-subsumed-learned-clauses-wl* *S*)›
‹¬*irred* (*get-clauses-wl S*) *N* ⟹ *get-subsumed-learned-clauses-wl* (*add-clauses-to-subsumed-wl N S*) = *add-mset* (*mset* (*get-clauses-wl S* ∝ *N*)) (*get-subsumed-learned-clauses-wl S*)›
‹*literals-to-update-wl* (*add-clauses-to-subsumed-wl N S*) = *literals-to-update-wl S*›
‹*get-watched-wl* (*add-clauses-to-subsumed-wl N S*) = *get-watched-wl S*›
‹*get-clauses-wl* (*add-clauses-to-subsumed-wl N S*) = *get-clauses-wl S*›
‹*get-init-clauses0-wl* (*add-clauses-to-subsumed-wl N S*) = *get-init-clauses0-wl* (*S*)›
‹*get-learned-clauses0-wl* (*add-clauses-to-subsumed-wl N S*) = *get-learned-clauses0-wl* (*S*)›
‹*get-conflict-wl* (*add-clauses-to-subsumed-wl N S*) = *get-conflict-wl S*›
apply (*solves* ‹*cases S*; *auto simp*: *add-clauses-to-subsumed-wl-def*›)+
done

lemma *remove-lit-from-clause-st*:
assumes
T: ‹(*T*, *S*) ∈ *twl-st-heur-restart-occs′ r u*› **and**
LL′: ‹(*L*,*L′*)∈*nat-lit-lit-rel*› **and**

CC': ‹(*C*,*C'*)∈*nat-rel*› **and**
C-dom: ‹*C* ∈# *dom-m* (*get-clauses-wl S*)› **and**
dist: ‹*distinct* (*get-clauses-wl S* ∝ *C*)›**and**
le: ‹*length* (*get-clauses-wl S* ∝ *C*) > *2*›
shows
‹*remove-lit-from-clause-st T C L*
≤ *SPEC* (λ*c*. (*c*, *remove-lit-from-clause-wl C' L' S*) ∈ {(*U*,*V*). (*U*,*V*)∈*twl-st-heur-restart-occs' r u* ∧ *get-occs U* = *get-occs T* ∧ *get-aivdom U* = *get-aivdom T*})›
proof −
define *I* **where**
‹*I* = (λ(*i*,*j*,*N*). *dom-m N* = *dom-m* (*get-clauses-wl S*) ∧
(∀ *D*∈#*dom-m* (*get-clauses-wl S*). *D* ≠ *C* ⟶ *fmlookup N D* = *fmlookup* (*get-clauses-wl S*) *D*) ∧
(*take i* (*N* ∝ *C*) = (*removeAll L* (*take j* (*get-clauses-wl S* ∝ *C'*)))) ∧
(∀ *k* < *length* (*N* ∝ *C*). *k* ≥ *j* ⟶ (*N* ∝ *C*) !*k* = (*get-clauses-wl S* ∝ *C'*) ! *k*) ∧
(*length* (*N* ∝ *C*) = *length* (*get-clauses-wl S* ∝ *C'*)) ∧
irred N C = *irred* (*get-clauses-wl S*) *C* ∧
i ≤ *j* ∧ *j* ≤ *length* (*get-clauses-wl S* ∝ *C'*))›
define Φ **where**
‹Φ = (λ(*i*,*j*,*N*). *dom-m N* = *dom-m* (*get-clauses-wl S*) ∧
(∀ *D*∈#*dom-m* (*get-clauses-wl S*). *D* ≠ *C* ⟶ *fmlookup N D* = *fmlookup* (*get-clauses-wl S*) *D*) ∧
j = *length* (*get-clauses-wl S* ∝ *C'*) ∧
(*take i* (*N* ∝ *C*) = (*removeAll L* (*get-clauses-wl S* ∝ *C'*))) ∧
(*length* (*N* ∝ *C*) = *length* (*get-clauses-wl S* ∝ *C'*)) ∧
irred N C = *irred* (*get-clauses-wl S*) *C* ∧ *C'* ∈# *dom-m N* ∧
i ≤ *j* ∧ *j* = *length* (*get-clauses-wl S* ∝ *C'*))›
have *ge*: ‹⇓*Id* (*RETURN* (*remove-lit-from-clause-wl C' L' S*)) ≥ *do* {
let n = *length* (*get-clauses-wl S* ∝ *C'*);
(*i*, *j*, *N*) ← *WHILE*$_T$ (λ(*i*, *j*, *N*). *j* < *n*)
(λ(*i*, *j*, *N*). *do* {
ASSERT (*i* < *n*);
ASSERT (*j* < *n*);
K ← *mop-clauses-at N C j*;
if K ≠ *L then do* {
N ← *mop-clauses-swap N C i j*;
RETURN (*i*+*1*, *j*+*1*, *N*)}
else RETURN (*i*, *j*+*1*, *N*)
}) (*0*, *0*, *get-clauses-wl S*);
ASSERT (*C'* ∈# *dom-m N*);
ASSERT (*i* ≤ *length* (*N* ∝ *C'*));
ASSERT (*i* ≥ *2*);
let N = *N*(*C'* ↪ *take i* (*N* ∝ *C'*));
ASSERT (*N* = (*get-clauses-wl S*)(*C'* ↪ *removeAll L* (*get-clauses-wl S* ∝ *C'*)));
let N = *N*;
RETURN (*set-clauses-wl N* (*add-clause-to-subsumed* (*irred* (*get-clauses-wl S*) *C'*) (*mset* (*get-clauses-wl S* ∝ *C'*)) *S*))
}›
unfolding *Let-def*
apply (*refine-vcg*)
apply (*rule WHILET-rule*[**where** *I*= ‹*I*› **and** *R* = ‹*measure* (λ(*i*,*j*,*N*). *length* (*get-clauses-wl S* ∝ *C'*) − *j*)› **and** Φ=Φ, *THEN order-trans*])
subgoal by *auto*
subgoal using *CC'* **unfolding** *I-def* **by** *auto*
subgoal
unfolding *mop-clauses-at-def nres-monad3 mop-clauses-swap-def*
apply (*refine-vcg*)
subgoal using *C-dom* **by** (*auto simp*: *I-def*)

subgoal using *CC′* **by** (*auto simp*: *I-def*)
subgoal using *CC′ C-dom* **by** (*auto simp*: *I-def*)
subgoal using *CC′* **by** (*auto simp*: *I-def swap-only-first-relevant take-Suc-conv-app-nth*)
subgoal using *CC′* **by** (*auto simp*: *I-def swap-only-first-relevant take-Suc-conv-app-nth*)
subgoal unfolding *I-def prod.simps* **apply** (*intro conjI*)
subgoal using *CC′* **by** (*auto simp*: *I-def swap-only-first-relevant take-Suc-conv-app-nth*)
subgoal using *CC′* **by** (*auto simp*: *I-def swap-only-first-relevant take-Suc-conv-app-nth*)
subgoal
using *CC′* **apply** (*auto simp*: *I-def swap-only-first-relevant take-Suc-conv-app-nth*)
by (*metis order-mono-setup.refl take-update take-update-cancel*)+
subgoal using *CC′* **by** (*auto simp*: *I-def swap-only-first-relevant take-Suc-conv-app-nth*)
subgoal using *CC′* **by** (*auto simp*: *I-def swap-only-first-relevant take-Suc-conv-app-nth*)
subgoal using *CC′* **by** (*auto simp*: *I-def swap-only-first-relevant take-Suc-conv-app-nth*)
subgoal using *CC′* **by** (*auto simp*: *I-def swap-only-first-relevant take-Suc-conv-app-nth*)
subgoal using *CC′* **by** (*auto simp*: *I-def swap-only-first-relevant take-Suc-conv-app-nth*)
done
subgoal by (*auto simp*: *I-def*)
subgoal by (*auto simp*: *I-def swap-only-first-relevant take-Suc-conv-app-nth*)
subgoal by (*auto simp*: *I-def*)
done
subgoal using *C-dom CC′* **unfolding** Φ-*def I-def* **by** *auto*
subgoal using *dist CC′ LL′ le*
unfolding Φ-*def* **by** (*cases S*) (*auto 4 4 simp*: *remove-lit-from-clause-wl-def distinct-remove1-removeAll*
fmap-eq-iff-dom-m-lookup count-list-distinct-If length-removeAll-count-list
intro!: *ASSERT-leI dest*: *arg-cong*[*of* ‹*take* - -› - *length*])
done
have *valid*: ‹*valid-arena* (*get-clauses-wl-heur T*) (*get-clauses-wl S*) (*set* (*get-vdom T*))› **and**
corr: ‹*clss-size-corr-restart* (*get-clauses-wl S*)
(*IsaSAT-Setup.get-unkept-unit-init-clss-wl S*) {#} (*IsaSAT-Setup.get-kept-unit-init-clss-wl S*)
(*IsaSAT-Setup.get-kept-unit-learned-clss-wl S*)
(*get-subsumed-init-clauses-wl S*)
{#} (*get-init-clauses0-wl S*) {#} (*get-learned-count T*)›
using *T* **unfolding** *twl-st-heur-restart-occs-def* **by** *fast*+
have [*refine*]: ‹((*0*, *0*, *get-clauses-wl-heur T*), *0*, *0*, *get-clauses-wl S*) ∈ *nat-rel* ×$_r$ *nat-rel* ×$_r$ {(*N*,*N′*).
valid-arena N N′ (*set* (*get-vdom T*)) ∧ *length N* = *length* (*get-clauses-wl-heur T*)}›
using *valid* **by** *auto*
have *H*: ‹*A* = *B* ⟹ *x* ∈ *A* ⟹ *x* ∈ *B*› **for** *A B x*
by *auto*
have *H′*: ‹*A* = *B* ⟹ *A x* ⟹ *B x*› **for** *A B x*
by *auto*

note *cong* = *trail-pol-cong heuristic-rel-cong*
option-lookup-clause-rel-cong isa-vmtf-cong
vdom-m-cong[*THEN H*] *isasat-input-nempty-cong*[*THEN iffD1*]
isasat-input-bounded-cong[*THEN iffD1*]
cach-refinement-empty-cong[*THEN H′*]
phase-saving-cong[*THEN H′*]
$\mathcal{L}_{all}$-*cong*[*THEN H*]
D_0-*cong*[*THEN H*]
map-fun-rel-D_0*-cong*
vdom-m-cong[*symmetric*] $\mathcal{L}_{all}$-*cong isasat-input-nempty-cong*

have *mset-removeAll-subseteq*: ‹*mset* (*removeAll L M*) ⊆# *mset M*› **for** *L M*
by (*subst mset-removeAll*[*symmetric*]) (*rule diff-subset-eq-self*)
have [*simp*]: ‹ *C′* ∈# *dom-m b* ⟹ *irred b C′* ⟹
set-mset (*all-lits-of-mm*

```
      (add-mset (mset (removeAll L (b ∝ C')))
      ({#mset (fst x). x ∈# init-clss-l b#} + dj))) =
   set-mset (all-lits-of-mm
      ({#mset (fst x). x ∈# init-clss-l b#} + dj))› for b dj i
    using all-lits-of-m-mono[of ‹mset (removeAll L (b ∝ C'))› ‹mset (b ∝ C')›]
    by (auto simp: all-lits-of-mm-add-mset ran-m-def mset-take-subseteq mset-removeAll-subseteq
        dest!: multi-member-split[of C'])

  have ‹set-mset (all-init-atms-st (set-clauses-wl ((get-clauses-wl S)(C' ↪ removeAll L (get-clauses-wl
S ∝ C'))))
    (add-clause-to-subsumed (irred (get-clauses-wl S) C') (mset (get-clauses-wl S ∝ C')) S))) =
    set-mset (all-init-atms-st S)› for i
    using C-dom CC'
    by (cases S)
     (auto simp: all-init-atms-st-def simp del: all-init-atms-def[symmetric]
     simp: all-init-lits-def all-init-atms-def init-clss-l-fmdrop-if init-clss-l-mapsto-upd init-clss-l-clause-upd
     image-mset-remove1-mset-if add-mset-commute[of - ‹mset (removeAll - -)›] init-clss-l-mapsto-upd-irrel)
  note cong1 = cong[OF this(1)[symmetric]]

  have [simp]: ‹clss-size-corr-restart ((get-clauses-wl S)(C' ↪ removeAll L (get-clauses-wl S ∝ C')))
  (IsaSAT-Setup.get-unkept-unit-init-clss-wl S) {#} (IsaSAT-Setup.get-kept-unit-init-clss-wl S)
  (IsaSAT-Setup.get-kept-unit-learned-clss-wl S)
  (get-subsumed-init-clauses-wl
    (add-clause-to-subsumed (irred (get-clauses-wl S) C') (mset (get-clauses-wl S ∝ C')) S))
  {#} (get-init-clauses0-wl S) {#} (get-learned-count T)›
    using C-dom CC' corr by (auto simp: clss-size-corr-restart-def clss-size-def)
  have[simp]:
   ‹vdom-m (all-init-atms-st S) (get-watched-wl S) ((get-clauses-wl S)(C' ↪ removeAll L (get-clauses-wl
S ∝ C'))) =
     vdom-m (all-init-atms-st S) (get-watched-wl S) ((get-clauses-wl S))›
    using C-dom CC' by (auto simp: vdom-m-def)

  have still-in-dom-after-shortening: ‹C' ∈# dom-m x2c ⟹
    x1b ≤ length (x2c ∝ C') ⟹
    TIER-ONE-MAXIMUM ≤ x1b ⟹
    (xb, x2c(C' ↪ take x1b (x2c ∝ C')))
    ∈ {(N₁, N₁'). valid-arena N₁ N₁' (set (get-vdom T)) ∧ length N₁ = length x2a} ⟹
    C ∈# dom-m (x2c(C' ↪ take x1b (x2c ∝ C')))›for x1b x2c xb x2a
   using CC' by auto
  have H: ‹(xb, x2c(C' ↪ take x1b (x2c ∝ C')))
    ∈ {(N₁, N₁'). valid-arena N₁ N₁' (set (get-vdom T)) ∧ length N₁ = length x2a} ⟹
    (xc, x2c(C' ↪ take x1b (x2c ∝ C')))
    ∈ {(c, N').
    N' = x2c(C' ↪ take x1b (x2c ∝ C')) ∧
    valid-arena c (x2c(C' ↪ take x1b (x2c ∝ C'))) (set (get-vdom T)) ∧
    length c = length xb} ⟹
    (xc, x2c(C' ↪ take x1b (x2c ∝ C'))) ∈ {(c, N').
    N' = x2c(C' ↪ take x1b (x2c ∝ C')) ∧
    valid-arena c (x2c(C' ↪ take x1b (x2c ∝ C'))) (set (get-vdom T)) ∧
    length c = length x2a}›for x1b xb x2a xc x2c
   by auto
  show ?thesis
   unfolding conc-fun-RETURN[symmetric]
   apply (rule ref-two-step)
   defer apply (rule ge[unfolded Down-id-eq])
   unfolding remove-lit-from-clause-st-def remove-lit-from-clause-def nres-monad3
```

apply (*refine-vcg mop-arena-length*[*of* ‹*set* (*get-vdom T*)›, *THEN fref-to-Down-curry*, *unfolded comp-def*]
mop-arena-lit2[*of* - - ‹*set* (*get-vdom T*)›] *mop-arena-swap2*[*of* - - ‹*set* (*get-vdom T*)›]
mop-arena-shorten[*of* - - - - - *C C'*] *update-lbd-shrunk-clause-valid*[*of* - - - ‹*set* (*get-vdom T*)›])
subgoal using *C-dom CC'* **by** *auto*
subgoal using *CC' valid* **by** *auto*
subgoal using *CC' C-dom* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
apply (*solves auto*)[]
apply (*solves auto*)[]
subgoal using *CC'* **by** *auto*
subgoal using *CC'* **by** *auto*
subgoal by *auto*
subgoal using *CC'* **by** *auto*
apply (*rule still-in-dom-after-shortening*; *assumption*)
subgoal by *auto*
apply (*rule H*; *assumption*)
subgoal using *T CC' C-dom*
by (*clarsimp simp add*: *twl-st-heur-restart-occs-def cong1 IsaSAT-Restart.all-init-atms-alt-def*
simp del: *isasat-input-nempty-def*)
done
qed

lemma *add-clauses-to-subsumed-wl-twl-st-heur-restart-occs*:
assumes ‹(*S*, *T*) ∈ *twl-st-heur-restart-occs' r u*› **and**
D: ‹*D* ∈# *dom-m* (*get-clauses-wl T*)›
shows ‹(*S*, *add-clauses-to-subsumed-wl D T*) ∈ {(*U*,*V*). (*U*,*V*)∈*twl-st-heur-restart-occs' r u* ∧ *get-occs U* = *get-occs S* ∧ *get-aivdom U* = *get-aivdom S*}›
proof −
have *H*: ‹*A* = *B* ⟹ *x* ∈ *A* ⟹ *x* ∈ *B*› **for** *A B x*
by *auto*
have *H'*: ‹*A* = *B* ⟹ *A x* ⟹ *B x*› **for** *A B x*
by *auto*

note *cong* = *trail-pol-cong heuristic-rel-cong*
option-lookup-clause-rel-cong isa-vmtf-cong
vdom-m-cong[*THEN H*] *isasat-input-nempty-cong*[*THEN iffD1*]
isasat-input-bounded-cong[*THEN iffD1*]
cach-refinement-empty-cong[*THEN H'*]
phase-saving-cong[*THEN H'*]
$\mathcal{L}_{all}$-*cong*[*THEN H*]
D_0-*cong*[*THEN H*]
map-fun-rel-D_0-*cong*
vdom-m-cong[*symmetric*] $\mathcal{L}_{all}$-*cong isasat-input-nempty-cong*
have ‹*set-mset* (*all-init-atms-st* (*add-clauses-to-subsumed-wl D T*)) =
set-mset (*all-init-atms-st T*)›

using *D* **by** (*cases T*)
(*auto simp*: *all-init-atms-st-def add-clauses-to-subsumed-wl-def all-init-atms-def*
all-init-lits-def ran-m-def all-lits-of-mm-add-mset
dest!: *multi-member-split*[*of* ‹- :: *nat*›]
simp del: *all-init-atms-def*[*symmetric*])
note *cong1* = *cong*[*OF this*[*symmetric*]]
show *?thesis*
using *assms*
unfolding *twl-st-heur-restart-occs-def IsaSAT-Restart.all-init-atms-alt-def*
by (*cases* ‹*irred* (*get-clauses-wl T*) *D*›)
(*clarsimp-all simp add*: *cong1 simp del*: *isasat-input-bounded-def isasat-input-nempty-def*)
qed

lemma *mark-garbage-wl-no-learned-reset-simp*[*simp*]:
‹*get-trail-wl* (*mark-garbage-wl-no-learned-reset C S*) = *get-trail-wl S*›
‹*IsaSAT-Setup.get-unkept-unit-init-clss-wl* (*mark-garbage-wl-no-learned-reset C S*) = *IsaSAT-Setup.get-unkept-unit-init-*
S›
‹*IsaSAT-Setup.get-kept-unit-init-clss-wl* (*mark-garbage-wl-no-learned-reset C S*) = *IsaSAT-Setup.get-kept-unit-init-clss-*
S›
‹*get-subsumed-init-clauses-wl* (*mark-garbage-wl-no-learned-reset C S*) = (*get-subsumed-init-clauses-wl*
S)›
‹*IsaSAT-Setup.get-unkept-unit-learned-clss-wl* (*mark-garbage-wl-no-learned-reset C S*) = *IsaSAT-Setup.get-unkept-unit-*
S›
‹*IsaSAT-Setup.get-kept-unit-learned-clss-wl* (*mark-garbage-wl-no-learned-reset C S*) = *IsaSAT-Setup.get-kept-unit-learne*
S›
‹*get-subsumed-learned-clauses-wl* (*mark-garbage-wl-no-learned-reset C S*) = (*get-subsumed-learned-clauses-wl*
S)›
‹*literals-to-update-wl* (*mark-garbage-wl-no-learned-reset C S*) = *literals-to-update-wl S*›
‹*get-watched-wl* (*mark-garbage-wl-no-learned-reset C S*) = *get-watched-wl S*›
‹*get-clauses-wl* (*mark-garbage-wl-no-learned-reset C S*) = *fmdrop C* (*get-clauses-wl S*)›
‹*get-init-clauses0-wl* (*mark-garbage-wl-no-learned-reset C S*) = *get-init-clauses0-wl* (*S*)›
‹*get-learned-clauses0-wl* (*mark-garbage-wl-no-learned-reset C S*) = *get-learned-clauses0-wl* (*S*)›
‹*get-conflict-wl* (*mark-garbage-wl-no-learned-reset C S*) = *get-conflict-wl S*›
apply (*solves* ‹*cases S*; *auto simp*: *mark-garbage-wl-no-learned-reset-def*›)+
done

lemma [*simp*]: ‹*get-occs* (*incr-wasted-st b S*) = *get-occs S*›
‹*learned-clss-count* (*incr-wasted-st b S*) = *learned-clss-count S*›
by (*auto simp*: *incr-wasted-st-def*)

lemma *log-clause-heur-log-clause2-occs*:
assumes ‹(*S*,*T*) ∈ *twl-st-heur-restart-occs′ r u*› ‹(*C*,*C′*) ∈ *nat-rel*›
shows ‹*log-clause-heur S C* ≤⇓*unit-rel* (*log-clause2 T C′*)›
proof −
have [*refine0*]: ‹(*0*,*0*)∈*nat-rel*›
by *auto*
have *length*: ‹⇓ *nat-rel* ((*RETURN* ∘ (λ*c*. *length* (*get-clauses-wl T* ∝ *c*))) *C′*) ≤ *SPEC* (λ*c*. (*c*, *length*
(*get-clauses-wl T* ∝ *C′*)) ∈ {(*a*,*b*). *a*=*b* ∧ *a* = *length* (*get-clauses-wl T* ∝ *C*)})›
by (*use assms* **in** *auto*)
show *?thesis*
unfolding *log-clause-heur-def log-clause2-def comp-def uncurry-def mop-arena-length-st-def*
mop-access-lit-in-clauses-heur-def
apply (*refine-vcg mop-arena-lit*[**where** *vdom* = ‹*set* (*get-vdom S*)› **and** *N* = ‹*get-clauses-wl T*›,
THEN order-trans]
mop-arena-length[**where** *vdom* = ‹*set* (*get-vdom S*)›, *THEN fref-to-Down-curry*, *THEN order-trans*,

unfolded prod.simps])
apply *assumption*
subgoal using *assms* **by** (*auto simp*: *twl-st-heur-restart-occs-def twl-st-heur-restart-def*)
apply (*rule length*)
subgoal by (*use assms* **in** ‹*auto simp*: *twl-st-heur-restart-occs-def twl-st-heur-restart-def dest*: *arena-lifting*(*10*)›)
subgoal by *auto*
subgoal using *assms* **by** (*auto simp*: *twl-st-heur-restart-occs-def twl-st-heur-restart-def*)
apply *assumption*
subgoal by (*use assms* **in** *auto*)
apply (*rule refl*)
subgoal by *auto*
by *auto*
qed

lemma *log-del-clause-heur-log-clause*:
assumes ‹(*S*,*T*) ∈ *twl-st-heur-restart-occs′ r u*› ‹(*C*,*C′*) ∈ *nat-rel*› ‹*C* ∈# *dom-m* (*get-clauses-wl T*)›
shows ‹*log-del-clause-heur S C* ≤ *SPEC* (λ*c*. (*c*, *log-clause T C′*) ∈ *unit-rel*)›
unfolding *log-del-clause-heur-alt-def*
apply (*rule log-clause-heur-log-clause2-occs*[*THEN order-trans*, *OF assms*(*1*,*2*)])
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule log-clause2-log-clause*[*THEN fref-to-Down-curry*])
using *assms* **by** *auto*

lemma *mark-garbage-heur-as-subsumed*:
assumes
T: ‹(*S*,*T*)∈ *twl-st-heur-restart-occs′ r u*› **and**
D: ‹*C* ∈# *dom-m* (*get-clauses-wl T*)› **and**
‹(*C′*,*C*)∈*nat-rel*› **and**
eq: ‹*set-mset* (*all-init-atms-st* (*mark-garbage-wl-no-learned-reset C′ T*)) = *set-mset* (*all-init-atms-st T*)›
shows ‹*mark-garbage-heur-as-subsumed C S* ≤ *SPEC* (λ*c*. (*c*, *mark-garbage-wl-no-learned-reset C′ T*) ∈ {(*U*,*V*). (*U*,*V*)∈*twl-st-heur-restart-occs′ r u* ∧ *get-occs U* = *get-occs S* ∧ *get-aivdom U* = *get-aivdom S*})›
proof −
have *H*: ‹*A* = *B* ⟹ *x* ∈ *A* ⟹ *x* ∈ *B*› **for** *A B x*
by *auto*
have *H′*: ‹*A* = *B* ⟹ *A x* ⟹ *B x*› **for** *A B x*
by *auto*

note *cong* = *trail-pol-cong heuristic-rel-cong*
option-lookup-clause-rel-cong isa-vmtf-cong
vdom-m-cong[*THEN H*] *isasat-input-nempty-cong*[*THEN iffD1*]
isasat-input-bounded-cong[*THEN iffD1*]
cach-refinement-empty-cong[*THEN H′*]
phase-saving-cong[*THEN H′*]
$\mathcal{L}_{all}$*-cong*[*THEN H*]
D_0*-cong*[*THEN H*]
map-fun-rel-D_0*-cong*
vdom-m-cong[*symmetric*] $\mathcal{L}_{all}$*-cong isasat-input-nempty-cong*
heuristic-rel-cong
have *CC′*: ‹*C′* = *C*›
using *assms* **by** *auto*
note *cong1* = *cong*[*OF eq*[*unfolded CC′*, *symmetric*]]

```
  have valid: ‹valid-occs (get-occs S) (get-aivdom S)› and
    arena: ‹valid-arena (get-clauses-wl-heur S) (get-clauses-wl T) (set (get-vdom S))›
    using T by (auto simp: twl-st-heur-restart-occs-def)

  have pre: ‹arena-is-valid-clause-idx (get-clauses-wl-heur S) C›
    unfolding arena-is-valid-clause-idx-def
   by (use T arena D in ‹auto simp: arena-is-valid-clause-idx-and-access-def intro!: exI[of - ‹get-clauses-wl
T›] exI[of - ‹set (get-vdom S)›]›)
  show ?thesis
    using pre
    unfolding twl-st-heur-restart-occs-def IsaSAT-Restart.all-init-atms-alt-def
      mark-garbage-heur-as-subsumed-def mop-arena-length-def nres-monad3
   apply (refine-vcg log-del-clause-heur-log-clause[where r=r and u=u and C'=C, THEN order-trans]
T)
    subgoal
      using arena D unfolding arena-is-valid-clause-vdom-def by auto
    subgoal by simp
    subgoal using arena D by simp
    subgoal using arena D unfolding mark-garbage-pre-def by simp
    subgoal
      using arena red-in-dom-number-of-learned-ge1[of C ‹get-clauses-wl T›] assms assms
        red-in-dom-number-of-learned-ge1-twl-st-heur-restart-occs[of S T r u C]
      by (cases ‹arena-status (get-clauses-wl-heur S) C›)
       (auto simp add: arena-lifting)
    subgoal
      using assms pre
      unfolding twl-st-heur-restart-occs-def IsaSAT-Restart.all-init-atms-alt-def
      mark-garbage-heur-as-subsumed-def mop-arena-length-def nres-monad3
      apply (clarsimp-all simp add: cong1 valid-arena-extra-information-mark-to-delete'
        aivdom-inv-dec-remove-clause arena-lifting
        simp del: isasat-input-bounded-def isasat-input-nempty-def
        simp flip: learned-clss-count-def
        )
      apply (intro conjI impI)
      subgoal
        by (auto dest: in-vdom-m-fmdropD simp add: incr-wasted-st-def cong1 intro!: heuristic-relI)
      subgoal
        by (auto simp add: incr-wasted-st-def cong1 intro!: heuristic-relI)
      subgoal
        by (auto simp add: incr-wasted-st-def cong1 intro!: heuristic-relI)
      subgoal
        by (auto dest: in-vdom-m-fmdropD simp add: incr-wasted-st-def cong1 intro!: heuristic-relI)
      subgoal
        by (auto simp add: incr-wasted-st-def cong1 intro!: heuristic-relI)
      subgoal
       by (auto simp flip: learned-clss-count-def simp add: incr-wasted-st-def cong1 intro!: heuristic-relI)
      done
    done
qed

lemma twl-st-heur-restart-occs'-with-occs:
  ‹a∈{(U,V). (U,V)∈twl-st-heur-restart-occs' r u ∧ get-occs U = get-occs T ∧ get-aivdom U = get-aivdom
S} ⟹
   a ∈ twl-st-heur-restart-occs' r u›
  by auto
```

lemma *isa-strengthen-clause-wl2*:
assumes
T: ‹(*T*, *S*) ∈ *twl-st-heur-restart-occs′ r u*› **and**
CC′: ‹(*C*,*C′*)∈*nat-rel*›**and**
DD′: ‹(*D*,*D′*)∈*nat-rel*› **and**
LL′: ‹(*L*,*L′*)∈*nat-lit-lit-rel*›
shows ‹*isa-strengthen-clause-wl2 C D L T*
≤ ⇓ {(*U*,*V*). (*U*,*V*)∈*twl-st-heur-restart-occs′ r u* ∧ *get-occs U* = *get-occs T* ∧ *get-aivdom U* = *get-aivdom T*}
(*strengthen-clause-wl C′ D′ L′ S*)›
proof −
have *arena*: ‹((*get-clauses-wl-heur T*, *C*), *get-clauses-wl S*, *C′*)
∈ {(*N*, *N′*). *valid-arena N N′* (*set* (*get-vdom T*))} ×*f nat-rel*›
‹((*get-clauses-wl-heur T*, *D*), *get-clauses-wl S*, *D′*)
∈ {(*N*, *N′*). *valid-arena N N′* (*set* (*get-vdom T*))} ×*f nat-rel*›
using *T CC′ DD′* **unfolding** *twl-st-heur-restart-occs-def*
by *auto*
have *C′*: ‹*C′* ∈# *dom-m* (*get-clauses-wl S*)› **and** *D′*: ‹*D′* ∈# *dom-m* (*get-clauses-wl S*)› **and**
C′-in-dom: ‹(*get-clauses-wl S*, *C′*) = (*x1*, *x2*) ⟹ *x2* ∈# *dom-m x1*› **and**
D′-in-dom: ‹(*get-clauses-wl S*, *D′*) = (*x1*, *x2*) ⟹ *x2* ∈# *dom-m x1*› **and**
C′-vdom: ‹*C′* ∈ *set* (*get-vdom T*)› **and**
D′-vdom: ‹*D′* ∈ *set* (*get-vdom T*)› **and**
C′-dist: ‹*distinct* (*get-clauses-wl S* ∝ *C′*)› **and**
le2: ‹*2* < *length* (*get-clauses-wl S* ∝ *C′*)›
if *pre*: ‹*subsume-or-strengthen-wl-pre C′* (*STRENGTHENED-BY L′ D′*) *S*›
for *x1 x2*
proof −
show *C′*: ‹*C′* ∈# *dom-m* (*get-clauses-wl S*)› **and** *D′*: ‹*D′* ∈# *dom-m* (*get-clauses-wl S*)›
using *pre* **unfolding** *subsume-or-strengthen-wl-pre-def subsume-or-strengthen-pre-def* **apply** −
by (*normalize-goal*+; *auto*)+
then show ‹*x2* ∈# *dom-m x1*› **if** ‹(*get-clauses-wl S*, *C′*) = (*x1*, *x2*)›
using *that* **by** *auto*
show ‹*x2* ∈# *dom-m x1*› **if** ‹(*get-clauses-wl S*, *D′*) = (*x1*, *x2*)›
using *D′ that* **by** *auto*
have
ai: ‹*aivdom-inv-dec* (*get-aivdom T*) (*dom-m* (*get-clauses-wl S*))›
using *T* **unfolding** *twl-st-heur-restart-occs-def* **by** *auto*
then have *H*: ‹*C′* ∈ *set* (*get-vdom T*)› **if** ‹*C′* ∈# *dom-m* (*get-clauses-wl S*)› **for** *C′*
using *ai that*
by (*smt* (*verit*, *ccfv-threshold*) *UnE aivdom-inv-dec-alt-def2 in-multiset-in-set mset-subset-eqD subsetD*)
show ‹*C′* ∈ *set* (*get-vdom T*)› **and** ‹*D′* ∈ *set* (*get-vdom T*)›
using *H*[*OF C′*] *H*[*OF D′*] *CC′ DD′*
by *auto*
show ‹*distinct* (*get-clauses-wl S* ∝ *C′*)› ‹*2* < *length* (*get-clauses-wl S* ∝ *C′*)›
using *CC′ pre* **unfolding** *subsume-or-strengthen-wl-pre-def subsume-or-strengthen-pre-def subsumption.simps* **apply** −
by (*solves* ‹*normalize-goal*+; *simp*›)+
qed
have [*refine*]: ‹(*Sa*, *U*) ∈ {(*U*, *V*).
(*U*, *V*) ∈ *twl-st-heur-restart-occs′ r u* ∧
get-occs U = *get-occs T* ∧ *get-aivdom U* = *get-aivdom T*} ⟹
(*set-stats-wl-heur* (*incr-forward-strengethening* (*get-stats-heur Sa*)) *Sa*, *U*)
∈ {(*U*, *V*).
(*U*, *V*) ∈ *twl-st-heur-restart-occs′ r u* ∧

get-occs U = get-occs T ∧ get-aivdom U = get-aivdom T}› **for** *Sa U*
by (*auto simp*: *twl-st-heur-restart-occs-def*)

have *H*: ‹(*x*, *y'*) ∈ *R* ⟹ *y*=*y'* ⟹ (*x*, *y*) ∈ *R*› **for** *x y y' R*
by *auto*
show *?thesis*
unfolding *isa-strengthen-clause-wl2-def*
apply (*rule ref-two-step*[*OF - strengthen-clause-wl-alt-def*])
unfolding *if-False Let-def*[*of* ‹*remove1 - -*›]
log-new-clause-heur-alt-def
Let-def[*of* ‹*mark-clause-for-unit-as-changed -*›]
apply (*refine-vcg mop-arena-length*[*of* ‹*set* (*get-vdom T*)›, *THEN fref-to-Down-curry*, *unfolded comp-def*]
remove-lit-from-clause-st T mop-arena-status2[*of - -* ‹*set* (*get-vdom T*)›]
mop-arena-promote-st-spec[**where** *r*=*r* **and** *u*=*u*] *mark-garbage-heur-as-subsumed*)
subgoal by (*rule C'-in-dom*)
subgoal by (*rule arena*)
subgoal by (*rule D'-in-dom*)
subgoal by (*rule arena*)
subgoal using *CC'* **by** *auto*
subgoal using *CC' C'-vdom* **by** *auto*
subgoal using *arena* **by** *auto*
subgoal using *DD'* **by** *auto*
subgoal using *DD' D'-vdom* **by** *auto*
subgoal using *arena* **by** *auto*
subgoal using *LL'* **by** *auto*
subgoal using *CC'* **by** *auto*
subgoal using *C' CC'* **by** *auto*
subgoal using *CC' C'-dist* **by** *fast*
subgoal using *CC' le2* **by** *fast*
apply (*rule log-clause-heur-log-clause2-occs*[**where** *r*=*r* **and** *u*=*u*])
subgoal by *auto*
subgoal using *CC'* **by** *auto*
subgoal by *auto*
apply (*rule add-clauses-to-subsumed-wl-twl-st-heur-restart-occs*[**where** *r*=*r* **and** *u*=*u*])
subgoal by *simp*
subgoal using *DD' D'* **by** *auto*
apply (*rule twl-st-heur-restart-occs'-with-occs*, *assumption*)
subgoal using *CC' C'* **by** *auto*
subgoal by *auto*
subgoal using *DD' CC' C' D' arena* **by** (*clarsimp simp add*: *arena-lifting*)
apply (*rule H*, *assumption*)
subgoal using *DD' CC'* **by** *auto*
subgoal using *DD' CC' C' D' arena* **by** (*auto simp*: *arena-lifting*)
apply (*rule twl-st-heur-restart-occs'-with-occs*, *assumption*)
subgoal using *DD' D'* **by** *simp*
subgoal using *DD'* **by** *simp*
subgoal premises *p*
using *p*(*18*,*21*) **by** *simp*
subgoal by *simp*
done
qed

lemma *isa-subsume-or-strengthen-wl-subsume-or-strengthen-wl*:
assumes
T: ‹(*T*, *S*) ∈ *twl-st-heur-restart-occs' r u*› **and**

x: ‹$(x2a,\ x2) \in Id$›
shows ‹*isa-subsume-or-strengthen-wl C x2a T*
$\leq \Downarrow \{(U,V).\ (U,V) \in$ *twl-st-heur-restart-occs'* $r\ u \wedge$ *get-occs* $U =$ *get-occs* $T \wedge$ *get-aivdom* $U =$ *get-aivdom* $T\}$
(*subsume-or-strengthen-wl C x2 S*)›
proof −

have H: ‹$A = B \Longrightarrow x \in A \Longrightarrow x \in B$› **for** $A\ B\ x$
by *auto*
have H': ‹$A = B \Longrightarrow A\ x \Longrightarrow B\ x$› **for** $A\ B\ x$
by *auto*

note *cong* = *trail-pol-cong heuristic-rel-cong*
option-lookup-clause-rel-cong isa-vmtf-cong
vdom-m-cong[*THEN H*] *isasat-input-nempty-cong*[*THEN iffD1*]
isasat-input-bounded-cong[*THEN iffD1*]
cach-refinement-empty-cong[*THEN H'*]
phase-saving-cong[*THEN H'*]
$\mathcal{L}_{all}$*-cong*[*THEN H*]
D_0*-cong*[*THEN H*]
map-fun-rel-D_0*-cong*
vdom-m-cong[*symmetric*] $\mathcal{L}_{all}$*-cong isasat-input-nempty-cong*

have *atms-eq*[*symmetric*]: ‹$C \in\#$ *dom-m* (*get-clauses-wl S*) $\Longrightarrow$
irred (*get-clauses-wl S*) $C \Longrightarrow$
set-mset (*all-init-atms* (*fmdrop C* (*get-clauses-wl S*))
(*add-mset* (*mset* (*get-clauses-wl S* $\propto$ *C*))
(*IsaSAT-Setup.get-unkept-unit-init-clss-wl S* + *IsaSAT-Setup.get-kept-unit-init-clss-wl S* +
get-subsumed-init-clauses-wl S +
get-init-clauses0-wl S))) =
set-mset (*all-init-atms* (*get-clauses-wl S*)
(*IsaSAT-Setup.get-unkept-unit-init-clss-wl S* + *IsaSAT-Setup.get-kept-unit-init-clss-wl S* +
get-subsumed-init-clauses-wl S +
get-init-clauses0-wl S))›
‹$C \in\#$ *dom-m* (*get-clauses-wl S*) $\Longrightarrow$
$\neg$*irred* (*get-clauses-wl S*) $C \Longrightarrow$
set-mset (*all-init-atms* (*fmdrop C* (*get-clauses-wl S*))
((*IsaSAT-Setup.get-unkept-unit-init-clss-wl S* + *IsaSAT-Setup.get-kept-unit-init-clss-wl S* +
get-subsumed-init-clauses-wl S +
get-init-clauses0-wl S))) =
set-mset (*all-init-atms* (*get-clauses-wl S*)
(*IsaSAT-Setup.get-unkept-unit-init-clss-wl S* + *IsaSAT-Setup.get-kept-unit-init-clss-wl S* +
get-subsumed-init-clauses-wl S +
get-init-clauses0-wl S))›
by (*simp-all add*: *all-init-atms-fmdrop-add-mset-unit*)
note *cong1* = *cong*[*OF this*(*1*)] *cong*[*OF this*(*2*)]

have H: ‹$x2a = x2$›
using x **by** *auto*
have *C-vdom*: ‹$C \in$ *set* (*get-vdom T*)› **and**
C-dom: ‹$C \in\#$ *dom-m* (*get-clauses-wl S*)› **and**
valid: ‹*valid-arena* (*get-clauses-wl-heur T*) (*get-clauses-wl S*) (*set* (*get-vdom T*))› **and**
C'-vdom: ‹$x2 =$ *SUBSUMED-BY* $C' \Longrightarrow C' \in$ *set* (*get-vdom T*)› **and**
C'-dom: ‹$x2 =$ *SUBSUMED-BY* $C' \Longrightarrow C' \in\#$ *dom-m* (*get-clauses-wl S*)› **and**
mark-garbage: ‹*mark-garbage-heur2 C T*
$\leq$ *SPEC* ($\lambda c.$ (*c*, *mark-garbage-wl2 C S*) $\in \{(U,V).\ (U,V) \in$ *twl-st-heur-restart-occs'* $r\ u \wedge$ *get-occs*

```
U=get-occs T ∧ get-aivdom U = get-aivdom T})›
    if ‹isa-subsume-or-strengthen-wl-pre C x2 T› and
      pre2: ‹subsume-or-strengthen-wl-pre C x2 S›
    for C'
  proof −
    show C: ‹C ∈# dom-m (get-clauses-wl S)›
      using pre2 T unfolding isa-subsume-or-strengthen-wl-pre-def subsume-or-strengthen-wl-pre-def
        subsume-or-strengthen-pre-def
      apply − by normalize-goal+ simp
    show valid: ‹valid-arena (get-clauses-wl-heur T) (get-clauses-wl S) (set (get-vdom T))›
      using T unfolding twl-st-heur-restart-occs-def by auto
    have
      ai: ‹aivdom-inv-dec (get-aivdom T) (dom-m (get-clauses-wl S))›
      using T unfolding twl-st-heur-restart-occs-def by auto
    then have H: ‹C' ∈ set (get-vdom T)› if ‹C' ∈# dom-m (get-clauses-wl S)› for C'
      using ai that
        by (smt (verit, ccfv-threshold) UnE aivdom-inv-dec-alt-def2 in-multiset-in-set mset-subset-eqD
subsetD)
    show ‹C ∈ set (get-vdom T)›
      by (rule H[OF C])
    have [simp]: ‹(all-init-atms (fmdrop C (get-clauses-wl S))
      (IsaSAT-Setup.get-unkept-unit-init-clss-wl S +
    IsaSAT-Setup.get-kept-unit-init-clss-wl S +
    get-subsumed-init-clauses-wl (mark-garbage-wl2 C S) +
      get-init-clauses0-wl S)) =
      (all-init-atms ((get-clauses-wl S))
      (IsaSAT-Setup.get-unkept-unit-init-clss-wl S +
    IsaSAT-Setup.get-kept-unit-init-clss-wl S +
    get-subsumed-init-clauses-wl (S) +
      get-init-clauses0-wl (mark-garbage-wl2 C S)))›
      by (cases ‹irred (get-clauses-wl S) C›)
       (use C in ‹auto simp: all-init-atms-def all-init-lits-def image-mset-remove1-mset-if
        simp del: all-init-atms-def[symmetric]›)
    have [simp]: ‹valid-arena (extra-information-mark-to-delete (get-clauses-wl-heur T) C)
      (fmdrop C (get-clauses-wl S)) (set (get-vdom T))›
      using valid C valid-arena-extra-information-mark-to-delete' by presburger
    have [simp]: ‹arena-status (get-clauses-wl-heur T) C = IRRED ⟹
    clss-size-corr-restart ((get-clauses-wl S))
  (IsaSAT-Setup.get-unkept-unit-init-clss-wl S) {#}
  (IsaSAT-Setup.get-kept-unit-init-clss-wl S)
  (IsaSAT-Setup.get-kept-unit-learned-clss-wl S)
  (get-subsumed-init-clauses-wl (S)) {#}
      (get-init-clauses0-wl S) {#} (get-learned-count T) ⟹
    clss-size-corr-restart (fmdrop C (get-clauses-wl S))
  (IsaSAT-Setup.get-unkept-unit-init-clss-wl S) {#}
  (IsaSAT-Setup.get-kept-unit-init-clss-wl S)
  (IsaSAT-Setup.get-kept-unit-learned-clss-wl S)
  (get-subsumed-init-clauses-wl (mark-garbage-wl2 C S)) {#}
      (get-init-clauses0-wl S) {#} (get-learned-count T)›
    ‹arena-status (get-clauses-wl-heur T) C ≠ IRRED ⟹
    clss-size-corr-restart ((get-clauses-wl S))
  (IsaSAT-Setup.get-unkept-unit-init-clss-wl S) {#}
  (IsaSAT-Setup.get-kept-unit-init-clss-wl S)
  (IsaSAT-Setup.get-kept-unit-learned-clss-wl S)
  (get-subsumed-init-clauses-wl (S)) {#}
      (get-init-clauses0-wl S) {#} (get-learned-count T) ⟹
```

```
    clss-size-corr-restart (fmdrop C (get-clauses-wl S))
  (IsaSAT-Setup.get-unkept-unit-init-clss-wl S) {#}
  (IsaSAT-Setup.get-kept-unit-init-clss-wl S)
  (IsaSAT-Setup.get-kept-unit-learned-clss-wl S)
  (get-subsumed-init-clauses-wl (mark-garbage-wl2 C S)) {#}
  (get-init-clauses0-wl S) {#} (clss-size-decr-lcount (get-learned-count T))›
    using C arena-lifting(24)[OF valid C] by (auto simp add: clss-size-corr-restart-def clss-size-def
      clss-size-decr-lcount-def size-remove1-mset-If split: prod.splits)
  show ‹mark-garbage-heur2 C T ≤ SPEC(λc. (c, mark-garbage-wl2 C S) ∈ {(U,V). (U,V)∈twl-st-heur-restart-occs'
r u ∧ get-occs U=get-occs T ∧ get-aivdom U = get-aivdom T})›
    unfolding mark-garbage-heur2-def nres-monad3
    apply refine-vcg
    subgoal
      using T C unfolding mark-garbage-pre-def arena-is-valid-clause-idx-def
      by (auto simp add: twl-st-heur-restart-occs-def aivdom-inv-dec-remove-clause cong1
        valid-arena-extra-information-mark-to-delete' arena-lifting clss-size-corr-restart-intro
        simp flip: learned-clss-count-def learned-clss-count-def
        simp del: isasat-input-nempty-def
        dest: in-vdom-m-fmdropD)
    subgoal by (rule red-in-dom-number-of-learned-ge1-twl-st-heur-restart-occs[OF T C])
    subgoal
      using T
      by (auto simp add: twl-st-heur-restart-occs-def aivdom-inv-dec-remove-clause cong1
        valid-arena-extra-information-mark-to-delete' arena-lifting clss-size-corr-restart-intro
        simp flip: learned-clss-count-def learned-clss-count-def
        simp del: isasat-input-nempty-def
        dest: in-vdom-m-fmdropD)
    done
  assume ‹x2 = SUBSUMED-BY C'›
  then show C': ‹C' ∈# dom-m (get-clauses-wl S)›
    using pre2 T unfolding isa-subsume-or-strengthen-wl-pre-def subsume-or-strengthen-wl-pre-def
      subsume-or-strengthen-pre-def
    apply − by normalize-goal+ simp
  show ‹C' ∈ set (get-vdom T)›
    by (rule H[OF C'])
qed
have [refine, intro!]: ‹(Sa, U)      ∈ {(U, V).
  (U, V) ∈ twl-st-heur-restart-occs' r u ∧
  get-occs U = get-occs T ∧ get-aivdom U = get-aivdom T} ⟹
  (set-stats-wl-heur (incr-forward-subsumed (get-stats-heur Sa)) Sa, U)
  ∈ {(U, V).
  (U, V) ∈ twl-st-heur-restart-occs' r u ∧
  get-occs U = get-occs T ∧ get-aivdom U = get-aivdom T}› for Sa U
  by (auto simp: twl-st-heur-restart-occs-def)
show ?thesis
  apply (rule order-trans)
  defer
  apply (rule ref-two-step'[OF subsume-or-strengthen-wl-alt-def])
  unfolding isa-subsume-or-strengthen-wl-def Let-def[of ‹If - - -›] Let-def[of ‹log-clause - -›]
    case-wl-split state-wl-recompose H
  apply (refine-vcg subsumption-cases-lhs mop-arena-status2[where vdom = ‹set (get-vdom T)›]
    mark-garbage mop-arena-promote-st-spec[where T=‹mark-garbage-wl2 C S› and r=r and u=u]
    isa-strengthen-clause-wl2[where r=r and u=u]
    log-del-clause-heur-log-clause[where r=r and u=u and T=S and C'=C] T)
  subgoal using T unfolding isa-subsume-or-strengthen-wl-pre-def by fast
  subgoal by auto
```

subgoal by *auto*
subgoal by (*rule C-vdom*)
subgoal by (*rule valid*)
subgoal by *auto*
subgoal by (*rule C′-vdom*)
subgoal by (*rule valid*)
subgoal by *auto*
subgoal using *C′-dom C-dom* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *valid C′-dom C-dom* **by** (*auto simp add*: *arena-lifting*)
subgoal using *valid C′-dom C-dom* **by** (*auto simp add*: *arena-lifting*)
subgoal using *valid C′-dom C-dom* **by** (*auto simp add*: *arena-lifting*)
subgoal using *valid C′-dom C-dom* **by** (*auto simp add*: *arena-lifting*)
subgoal by *auto*
subgoal using *T* **by** *auto*
done
qed

lemma *isa-forward-subsumption-one-forward-subsumption-wl-one*:
assumes
SS′: ‹(*S*, *S′*) ∈ *twl-st-heur-restart-occs′ r u*› **and**
CC′: ‹(*C*,*C′*)∈*nat-rel*› **and**
DD′: ‹(*D*,*D′*)∈*clause-hash*› **and**
‹(*L*,*L′*)∈*Id*› **and**
occs: ‹(*get-occs S*, *occs*) ∈ *occurrence-list-ref*›
shows ‹*isa-forward-subsumption-one-wl C D L S* ≤
⇓{((*U*, *changed*, *E*), (*S′*, *changed′*, *occs′*, *E′*)). (*get-occs U*, *occs′*) ∈ *occurrence-list-ref* ∧
get-aivdom U = *get-aivdom S* ∧ *changed′* = (*changed* ≠ *NONE*) ∧
((*U*, *E*), (*S′*, *E′*)) ∈ *twl-st-heur-restart-occs′ r u* ×$_r$ *clause-hash*}
(*forward-subsumption-one-wl2 C′ cands L′ occs D′ S′*)›
proof −
have *valid*: ‹*valid-occs* (*get-occs S*) (*get-aivdom S*)› **and**
dom-m-incl: ‹*set-mset* (*dom-m* (*get-clauses-wl S′*)) ⊆ *set* (*get-vdom S*)›
using *SS′* **by** (*auto simp*: *twl-st-heur-restart-occs-def vdom-m-def*)
have *C-vdom*: ‹*C* ∈*set* (*get-vdom S*)›
if ‹*forward-subsumption-one-wl2-pre C cands L S′*›
proof −
have ‹*C* ∈# *dom-m* (*get-clauses-wl S′*)›
using *that* **unfolding** *forward-subsumption-one-wl2-pre-def forward-subsumption-one-wl-pre-def*
forward-subsumption-one-pre-def **by** − (*normalize-goal+*; *simp*)
then show *?thesis*
using *dom-m-incl* **by** *fast*
qed

have *lits*: ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-init-atms-st S′*) (*mset* (*get-clauses-wl S′* ∝ *C*))› **and**
dist: ‹*distinct-mset* (*mset* (*get-clauses-wl S′* ∝ *C*))› **and**
tauto: ‹¬ *tautology* (*mset* (*get-clauses-wl S′* ∝ *C*))›
if ‹*forward-subsumption-one-wl2-pre C′ cands L′ S′*›
using *that CC′* **unfolding** *forward-subsumption-one-wl2-pre-def forward-subsumption-one-wl-pre-def*
forward-subsumption-one-pre-def
apply −
subgoal
apply *normalize-goal+*
apply (*frule literals-are-*$\mathcal{L}_{in}$*′-all-init-atms-alt-def*)
apply (*rule literals-are-in-*$\mathcal{L}_{in}$*-nth*)

```
    apply (simp add: literals-are-\mathcal{L}_{in}-def literals-are-\mathcal{L}_{in}'-def)
    apply (simp add: literals-are-in-\mathcal{L}_{in}-nth literals-are-\mathcal{L}_{in}-cong
      literals-are-\mathcal{L}_{in}'-literals-are-\mathcal{L}_{in}-iff)
    done
  subgoal
    apply normalize-goal+
    unfolding twl-struct-invs-def twl-st-inv-alt-def
    by (simp add:mset-take-mset-drop-mset')
  subgoal
    apply normalize-goal+
    unfolding twl-list-invs-def
    by (simp add:mset-take-mset-drop-mset')
 done

 have forward-subsumption-one-wl2-alt-def:
 ‹forward-subsumption-one-wl2 = (λC cands L occs D S. do {
 ASSERT (forward-subsumption-one-wl2-pre C cands L S);
 ASSERT (atm-of L ∈ fst occs);
 let ys = occ-list occs L;
 let n = length ys;
 (-, s) ←
   WHILE_T forward-subsumption-one-wl2-inv S C cands ys  (λ(i, s). i < n ∧ s = NONE)
   (λ(i, s). do {
     C' ← mop-occ-list-at occs L i;
     b ← RETURN (C' ∈# dom-m (get-clauses-wl S));
     if ¬b
     then RETURN (i+1, s)
     else do  {
       s ← subsume-clauses-match2 C' C S D;
      RETURN (i+1, s)
     }
   })
       (0, NONE);
 D ← (if s ≠ NONE then mop-ch-remove-all (mset (get-clauses-wl S ∝ C)) D else RETURN D);
 occs ← (if is-strengthened s then maybe-push-to-occs-list2 C S occs else RETURN occs);
 S ← subsume-or-strengthen-wl C s S;
 RETURN (S, s ≠ NONE, occs, D)
 })›
  unfolding forward-subsumption-one-wl2-def by auto
 have eq: ‹L' = L› ‹C' = C›
   using assms by auto
 have [refine]: ‹((0, NONE), 0, NONE) ∈ nat-rel ×_r Id›
   by auto
 have H[simp]: ‹forward-subsumption-one-wl2-pre C cands L S' ⟹ atm-of L ∈ fst occs ⟹
   (occ-list occs L) = (get-occs S ! nat-of-lit L)›
   using occs
   by (cases L)
    (auto simp: forward-subsumption-one-wl2-pre-def occ-list-def occurrence-list-ref-def map-fun-rel-def
     dest: bspec[of - - L])
 show ?thesis
  unfolding isa-forward-subsumption-one-wl-def forward-subsumption-one-wl2-alt-def eq Let-def mop-cocc-list-length-def
nres-monad3
     cocc-list-length-def
     bind-to-let-conv[of ‹length (get-occs - ! -)›]
  apply (refine-vcg mop-cocc-list-at-mop-occ-list-at occs mop-arena-status2[where arena=‹get-clauses-wl-heur
S› and vdom=‹set (get-vdom S)› and
```

N=‹*get-clauses-wl S'*›] *isa-subsume-clauses-match2-subsume-clauses-match2 SS' mop-cch-remove-all-clauses-mop-ch-CC' DD'*
isa-maybe-push-to-occs-list-st-push-to-occs-list2
isa-subsume-or-strengthen-wl-subsume-or-strengthen-wl[**where** *r*=*r* **and** *u*=*u*])
subgoal using *assms* **unfolding** *isa-forward-subsumption-one-wl-pre-def* **by** *fast*
subgoal using *occs* **by** (*cases L*) (*auto simp*: *occurrence-list-ref-def map-fun-rel-def dest*: *bspec*[*of* - - ‹*L*›])
subgoal by (*auto simp*: *cocc-list-length-pre-def*)
subgoal using *assms H* **unfolding** *isa-forward-subsumption-one-wl2-inv-def* **apply** − **by** (*rule exI*[*of* - *S'*]) *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *valid occs* **by** (*auto simp*: *forward-subsumption-one-wl2-pre-def occurrence-list-ref-def all-init-atms-st-alt-def*
all-occurrences-add-mset2 intro: *valid-occs-in-vdomI*)
subgoal using *SS'* **by** (*auto simp*: *twl-st-heur-restart-occs-def*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *DD'* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal unfolding *forward-subsumption-one-wl2-pre-def forward-subsumption-one-wl-pre-def*
forward-subsumption-one-pre-def
by *normalize-goal+ auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *C-vdom* **by** *fast*
subgoal
using *assms*(*1*,*2*,*3*,*4*) *simple-clss-size-upper-div2*[*of* ‹*all-init-atms-st S'*› ‹*mset* (*get-clauses-wl S'* ∝ *C*)›, *OF - lits dist tauto*]
by (*auto simp del*: *isasat-input-bounded-def simp*: *clause-not-marked-to-delete-def*
simp: *twl-st-heur-restart-occs-def IsaSAT-Restart.all-init-atms-alt-def*)
subgoal using *SS' occs* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto 4 3 simp*: *forward-subsumption-one-wl2-pre-def isa-forward-subsumption-one-wl-pre-def*
forward-subsumption-one-wl-pre-def)
done
qed

lemma *isa-try-to-forward-subsume-wl2-try-to-forward-subsume-wl2*:
assumes
SS': ‹(*S*, *S'*) ∈ *twl-st-heur-restart-occs' r u*› **and**
CC': ‹(*C*,*C'*)∈*nat-rel*› **and**
DD': ‹(*D*,*D'*)∈*clause-hash*› **and**
occs: ‹(*get-occs S*, *occs*) ∈ *occurrence-list-ref*› **and**
shrunken: ‹*shrunken* = *mset shrunken'*› **and**
cands: ‹*Suc* (*length shrunken'*) ≤ *length* (*get-tvdom S*)›
shows ‹*isa-try-to-forward-subsume-wl2 C D shrunken' S* ≤
⇓{((*D*, *shrunken*, *U*), (*occs*, *D'*, *shrunken'*, *S'*)). (*D*,*D'*)∈*clause-hash* ∧ (*get-occs U*, *occs*) ∈ *occur-*

rence-list-ref ∧
(*U*, *S′*) ∈ *twl-st-heur-restart-occs′ r u* ∧ *get-aivdom U* = *get-aivdom S* ∧ *shrunken′* = *mset shrunken*}
(*try-to-forward-subsume-wl2 C′ occs cands D′ shrunken S′*)›
proof −
have *le*: ‹*length* (*get-clauses-wl S′* ∝ *C*) ≤ *Suc* (*unat32-max div 2*)›
if ‹*try-to-forward-subsume-wl2-pre C′ cands shrunken S′*›
proof −
have *lits*: ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-init-atms-st S′*) (*mset* (*get-clauses-wl S′* ∝ *C′*))› **and**
dist: ‹*distinct-mset* (*mset* (*get-clauses-wl S′* ∝ *C′*))› **and**
tauto: ‹¬ *tautology* (*mset* (*get-clauses-wl S′* ∝ *C′*))›
using *that* **unfolding** *try-to-forward-subsume-wl2-pre-def try-to-forward-subsume-wl-pre-def*
try-to-forward-subsume-pre-def
apply −
subgoal
apply *normalize-goal+*
apply (*frule literals-are-*$\mathcal{L}_{in}$*′-all-init-atms-alt-def*)
apply (*rule literals-are-in-*$\mathcal{L}_{in}$*-nth*)
apply (*simp add*: *literals-are-*$\mathcal{L}_{in}$*-def literals-are-*$\mathcal{L}_{in}$*′-def*)
apply (*simp add*: *literals-are-in-*$\mathcal{L}_{in}$*-nth literals-are-*$\mathcal{L}_{in}$*-cong*
literals-are-$\mathcal{L}_{in}$*′-literals-are-*$\mathcal{L}_{in}$*-iff*)
done
subgoal
apply *normalize-goal+*
unfolding *twl-struct-invs-def twl-st-inv-alt-def*
by (*simp add*:*mset-take-mset-drop-mset′*)
subgoal
apply *normalize-goal+*
unfolding *twl-list-invs-def*
by (*simp add*:*mset-take-mset-drop-mset′*)
done
have ‹*isasat-input-bounded* (*all-init-atms-st S′*)›
using *SS′* **unfolding** *twl-st-heur-restart-occs-def IsaSAT-Restart.all-init-atms-alt-def*
by (*simp del*: *isasat-input-bounded-def*)
then show *?thesis*
using *simple-clss-size-upper-div2*[*of* ‹*all-init-atms-st S′*› ‹*mset* (*get-clauses-wl S′* ∝ *C′*)›, *OF - lits dist tauto*] *CC′*
by *auto*
qed
have *loop-rel*: ‹(*ebreak*, *ebreaka*) ∈ *bool-rel* ⟹
try-to-forward-subsume-wl2-inv S′ cands C′ (*0*, *False*, *ebreaka*, *occs*, *D′*, *S′*) ⟹
((*0*, *NONE*, *ebreak*, *D*, *S*), *0*, *False*, *ebreaka*, *occs*, *D′*, *S′*) ∈
{((*m*, *b*, *brk*, *D*, *U*),(*m′*, *b′*, *brk′*, *occs*, *D′*, *V*)). (*m*,*m′*)∈*nat-rel* ∧ *b′*=(*b*≠*NONE*) ∧ (*brk*,*brk′*)∈*bool-rel* ∧
(*D*,*D′*)∈*clause-hash* ∧ (*get-occs U*, *occs*) ∈ *occurrence-list-ref* ∧ (*U*, *V*) ∈ *twl-st-heur-restart-occs′* (*length* (*get-clauses-wl-heur S*)) (*learned-clss-count S*) ∧
get-aivdom U = *get-aivdom S*}›
for *ebreak ebreaka*
using *assms* **by** *auto*
have [*refine*]: ‹*RETURN* (*is-strengthened x1f*) ≤ ⇓ *bool-rel* (*RES UNIV*)› **for** *x1f*
by (*auto simp*:)

have [*refine*]: ‹*isa-try-to-forward-subsume-wl2-break S* ≤ *SPEC* (λ-. *True*)› **for** *S*
unfolding *isa-try-to-forward-subsume-wl2-break-def* **by** *auto*
show *?thesis*
supply [[*goals-limit=1*]]

unfolding *isa-try-to-forward-subsume-wl2-def*
try-to-forward-subsume-wl2-def mop-arena-length-st-def Let-def[*of* ‹*if - then mark-clause-for-unit-as-unchanged - else -*›]
apply (*refine-vcg mop-arena-lit*[*of* ‹*get-clauses-wl-heur S*› - ‹*set* (*get-vdom S*)› **for** *S*]
mop-arena-length[*THEN fref-to-Down-curry*, *of* - - - - ‹*set* (*get-vdom S*)›, *unfolded comp-def*] *loop-rel*
isa-forward-subsumption-one-forward-subsumption-wl-one[**where** *r*=‹*length* (*get-clauses-wl-heur S*)› **and** *u*=‹(*learned-clss-count S*)›]
mop-cch-remove-all-clauses-mop-ch-remove-all-clauses[**where** *r*=‹*length* (*get-clauses-wl-heur S*)› **and** *u*=‹(*learned-clss-count S*)›])
subgoal
using *assms* **unfolding** *isa-try-to-forward-subsume-wl-pre-def* **apply** −
by (*rule exI*[*of* - *S'*], *rule exI*[*of* - *r*], *rule exI*[*of* - *u*], *rule exI*[*of* - *cands*], *rule exI*[*of* - *occs*])
auto
subgoal using *SS' CC'* **unfolding** *isa-try-to-forward-subsume-wl-pre-def try-to-forward-subsume-wl2-pre-def try-to-forward-subsume-wl-pre-def*
try-to-forward-subsume-pre-def
by − (*normalize-goal*+; *auto*)
subgoal using *SS' CC'* **unfolding** *twl-st-heur-restart-occs-def* **by** *simp*
subgoal using *le CC'* **by** *auto*
subgoal for *n ebreak ebreaka x x'* **using** *CC' SS'* **unfolding** *isa-try-to-forward-subsume-wl-inv-def case-prod-beta* **apply** −
by (*rule exI*[*of* - *S'*], *rule exI*[*of* - ‹*snd* (*snd* (*snd* (*snd* (*snd* (*x'*)))))›],
rule exI[*of* - *cands*], *rule exI*[*of* - ‹*fst* ((*snd* (*snd* (*snd* (*x'*)))))›], *rule exI*[*of* - ‹*fst* (*snd* (*snd* (*snd* (*snd* (*x'*)))))›]; *cases x'*)
auto
subgoal by *simp*
subgoal by *auto*
subgoal for *n ebreak ebreaka x x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h*
by (*auto simp*: *twl-st-heur-restart-occs-def*)
subgoal using *CC'* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *CC'* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *CC'* **by** *auto*
subgoal by *auto*
subgoal using *CC'* **by** *auto*
subgoal by *auto*
subgoal using *cands* **by** *auto*
subgoal using *SS' CC' shrunken* **by** *auto*
done
qed

definition *clause-not-marked-to-delete-init-pre* **where**
‹*clause-not-marked-to-delete-init-pre* =
(λ(*S*, *C*). *C* ∈ *vdom-m* (*all-init-atms-st S*) (*get-watched-wl S*) (*get-clauses-wl S*))›

lemma *clause-not-marked-to-delete-rel-occs*:

‹(S,T)∈twl-st-heur-restart-occs ⟹ (C,C′)∈nat-rel ⟹ C ∈ set(get-vdom S) ⟹
(clause-not-marked-to-delete-heur S C, clause-not-marked-to-delete T C′) ∈ bool-rel›
by (*cases S*, *cases T*)
(*use arena-dom-status-iff*[*of* ‹*get-clauses-wl-heur S*› ‹*get-clauses-wl T*›
‹*set* (*get-vdom S*)› ‹*C*›] *in-dom-in-vdom* **in**
‹*auto 5 5 simp*: *clause-not-marked-to-delete-def twl-st-heur-restart-occs-def Let-def
*clause-not-marked-to-delete-heur-def all-init-atms-st-def
clause-not-marked-to-delete-init-pre-def ac-simps; *blast*›)

lemma *mop-polarity-pol-mop-polarity-wl*:
‹(S,T)∈twl-st-heur-restart-occs ⟹ (L,L′)∈nat-lit-lit-rel⟹L′∈# $\mathcal{L}_{all}$ (all-init-atms-st T)⟹
mop-polarity-pol (get-trail-wl-heur S) L ≤ ⇓Id (mop-polarity-wl T L′)›
unfolding *mop-polarity-pol-def mop-polarity-wl-def*
apply *refine-vcg*
subgoal
by (*rule polarity-pol-pre*[*of* - ‹*get-trail-wl T*› ‹*all-init-atms-st T*›])
(*auto simp add*: $\mathcal{L}_{all}$*-all-atms twl-st-heur-restart-occs-def IsaSAT-Restart.all-init-atms-alt-def*)
subgoal
by (*auto intro*!: *polarity-pol-polarity*[*of* ‹(*all-init-atms-st T*)›, *unfolded option-rel-id-simp*, *THEN
fref-to-Down-unRET-uncurry-Id*] *simp*: $\mathcal{L}_{all}$*-all-atms IsaSAT-Restart.all-init-atms-alt-def twl-st-heur-restart-occs-def*)
done

lemma *isa-all-lit-clause-unset-all-lit-clause-unset*:
assumes ‹(S, T) ∈ *twl-st-heur-restart-occs′ r u*› ‹(C,C′)∈*nat-rel*› **and**
‹*isa-all-lit-clause-unset-pre C S*›
shows
‹*isa-all-lit-clause-unset C S* ≤ ⇓ *bool-rel* (*all-lit-clause-unset T C′*)›
proof −
have [*refine*]: ‹((0,*True*, *added*), (0,*True*))∈{((a,b,c), (d,e)). ((a,b), (d,e))∈*nat-rel* ×$_f$ *bool-rel*}› **for**
added
by *auto*
have *lits*: ‹*literals-are-in-*$\mathcal{L}_{in}$ (*all-init-atms-st T*) (*mset* (*get-clauses-wl T* ∝ *C*))› **and**
dist: ‹*distinct-mset* (*mset* (*get-clauses-wl T* ∝ *C*))› **and**
tauto: ‹¬ *tautology* (*mset* (*get-clauses-wl T* ∝ *C*))›
if ‹*C* ∈# *dom-m* (*get-clauses-wl T*)› ‹*forward-subsumption-all-wl-pre T*›
using *that* **unfolding** *forward-subsumption-all-wl-pre-def forward-subsumption-all-pre-def*
apply −
subgoal
apply *normalize-goal+*
apply (*frule literals-are-*$\mathcal{L}_{in}$*′-all-init-atms-alt-def*)
apply (*rule literals-are-in-*$\mathcal{L}_{in}$*-nth*)
apply (*simp add*: *literals-are-*$\mathcal{L}_{in}$*-def literals-are-*$\mathcal{L}_{in}$*′-def*)
apply (*simp add*: *literals-are-in-*$\mathcal{L}_{in}$*-nth literals-are-*$\mathcal{L}_{in}$*-cong
literals-are-$\mathcal{L}_{in}$*′-literals-are-*$\mathcal{L}_{in}$*-iff*)
done
subgoal
apply *normalize-goal+*
unfolding *twl-struct-invs-def twl-st-inv-alt-def*
by (*simp add*:*mset-take-mset-drop-mset′*)
subgoal
apply *normalize-goal+*
unfolding *twl-list-invs-def*
by (*simp add*:*mset-take-mset-drop-mset′*)
done
have ‹*heuristic-rel* (*all-init-atms-st T*) (*get-heur S*)›

using *assms* **unfolding** *twl-st-heur-restart-occs-def IsaSAT-Restart.all-init-atms-alt-def* **by** *fast*
then have *is-marked-added-heur-pre-stats*: ‹$L \in\# \mathcal{L}_{all}$ (*all-init-atms-st T*) $\Longrightarrow$*is-marked-added-heur-pre* (*get-heur S*) (*atm-of L*)› **for** *L*
by (*auto simp*: *is-marked-added-heur-pre-stats-def heuristic-rel-def heuristic-rel-stats-def phase-saving-def is-marked-added-heur-pre-def*
dest!: *multi-member-split*)
show *?thesis*
unfolding *isa-all-lit-clause-unset-def all-lit-clause-unset-def mop-clause-not-marked-to-delete-heur-def*
nres-monad3 clause-not-marked-to-delete-def[*symmetric*] *mop-arena-length-st-def*
mop-is-marked-added-heur-st-def mop-is-marked-added-heur-def
apply (*refine-vcg clause-not-marked-to-delete-rel-occs mop-arena-lit*[**where** *vdom*=‹*set* (*get-vdom S*)›]
mop-polarity-pol-mop-polarity-wl
mop-arena-length[*THEN fref-to-Down-curry*, *unfolded comp-def*, *of* - - ‹*get-clauses-wl-heur S*› - ‹*set* (*get-vdom S*)› **for** *S*])
subgoal using *assms* **by** *fast*
subgoal using *assms* **unfolding** *clause-not-marked-to-delete-heur-pre-def isa-all-lit-clause-unset-pre-def*
by (*auto simp*: *arena-is-valid-clause-vdom-def twl-st-heur-restart-occs-def*
intro!: *exI*[*of* - ‹*get-clauses-wl T*›]
exI[*of* - ‹*set* (*get-vdom S*)›])
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **unfolding** *clause-not-marked-to-delete-init-pre-def*
by (*auto simp*: *isa-all-lit-clause-unset-pre-def*)
subgoal using *assms* **by** *auto*
subgoal by (*auto simp*: *clause-not-marked-to-delete-def*)
subgoal using *assms* **unfolding** *twl-st-heur-restart-occs-def* **by** *auto*
subgoal by *auto*
subgoal
using *assms*(*1*,*2*) *simple-clss-size-upper-div2*[*of* ‹*all-init-atms-st T*› ‹*mset* (*get-clauses-wl T* $\propto$ *C*)›, *OF* - *lits dist tauto*]
by (*auto simp del*: *isasat-input-bounded-def simp*: *clause-not-marked-to-delete-def*
simp: *twl-st-heur-restart-occs-def IsaSAT-Restart.all-init-atms-alt-def*)
subgoal by (*auto simp*: *unat32-max-def*)
subgoal using *assms* **unfolding** *twl-st-heur-restart-occs-def* **by** *auto*
subgoal using *assms* **by** *fast*
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal by (*rule is-marked-added-heur-pre-stats*) *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

lemma *valid-occs-empty*:
assumes ‹(*occs*, *empty-occs-list A*) ∈ *occurrence-list-ref*›
shows ‹*valid-occs occs vdom*› **and** ‹*cocc-content-set occs* = {}›
proof −
show ‹*cocc-content-set occs* = {}›
using *assms*
apply (*auto simp*: *valid-occs-def empty-occs-list-def occurrence-list-ref-def*
map-fun-rel-def cocc-content-set-def in-set-conv-nth)
by (*smt* (*verit*, *del-insts*) *fst-conv image-iff*)
then show ‹*valid-occs occs vdom*›
using *in-cocc-content-iff* **by** (*fastforce simp*: *valid-occs-def empty-occs-list-def occurrence-list-ref-def*

map-fun-rel-def cocc-content-set-def)
qed

lemma *twl-st-heur-restart-occs'-avdom-nth-vdom*:
‹(*S*, *S'*) ∈ *twl-st-heur-restart-occs' r u* ⟹ *i* < *length* (*get-avdom S*) ⟹
get-avdom S ! *i* ∈ *set* (*get-vdom S*)› **and**
twl-st-heur-restart-occs'-ivdom-nth-vdom:
‹(*S*, *S'*) ∈ *twl-st-heur-restart-occs' r u* ⟹ *i* < *length* (*get-ivdom S*) ⟹
get-ivdom S ! *i* ∈ *set* (*get-vdom S*)›
using *nth-mem*[*of i* ‹*get-avdom S*›] *nth-mem*[*of i* ‹*get-ivdom S*›]
by (*auto simp*: *twl-st-heur-restart-occs-def aivdom-inv-dec-alt-def simp del*: *nth-mem*)

lemma *isa-is-candidate-forward-subsumption*:
assumes *S*: ‹(*S*, *S'*) ∈ *twl-st-heur-restart-occs' r u*› ‹*C* ∈# *dom-m* (*get-clauses-wl S'*)› **and**
pre: ‹*forward-subsumption-all-wl-pre S'*›
shows ‹*isa-is-candidate-forward-subsumption S C* ≤ ⇓ *bool-rel* (*RES UNIV*)›
proof −
have *1*: ‹⇓ *bool-rel* (*RES UNIV*) = *SPEC* (λ-::*bool*. *True*)›
by (*auto*)
have *valid*: ‹*valid-arena* (*get-clauses-wl-heur S*) (*get-clauses-wl S'*) (*set* (*get-vdom S*))› **and**
heur: ‹*heuristic-rel* (*all-init-atms-st S'*) (*get-heur S*)› **and**
trail: ‹(*get-trail-wl-heur S*, *get-trail-wl S'*) ∈ *trail-pol* (*all-init-atms-st S'*)›
using *assms* **unfolding** *twl-st-heur-restart-occs-def IsaSAT-Restart.all-init-atms-alt-def* **by** *fast+*
have *H*: ‹*is-marked-added-heur-pre* (*get-heur S*)
(*atm-of* (*arena-lit* (*get-clauses-wl-heur S*) (*C* + *a*)))› **and**
H': ‹(*atm-of* (*arena-lit* (*get-clauses-wl-heur S*) (*C* + *a*))) ∈# *all-init-atms-st S'*›
if ‹*a* < *arena-length* (*get-clauses-wl-heur S*) *C*› **for** *a*
proof −
have *no-alien*: ‹*set-mset* (*$\mathcal{L}_{all}$* (*all-init-atms-st S'*)) = *set-mset* (*all-lits-st S'*)›
using *literals-are-$\mathcal{L}_{in}$'-literals-are-$\mathcal{L}_{in}$-iff*(*3*)[*of S'*, *THEN* *$\mathcal{L}_{all}$-cong*] *pre*
unfolding *forward-subsumption-all-wl-pre-def*
forward-subsumption-all-pre-def **apply** −
by *normalize-goal+*
(*simp add*: *$\mathcal{L}_{all}$-all-atms*)

have ‹*a* < *length* (*get-clauses-wl S'* ∝ *C*)›
using *valid assms that* **by** (*auto simp*: *arena-lifting*)
then have ‹*atm-of* (*get-clauses-wl S'* ∝ *C* ! *a*) ∈# *all-atms-st S'*›
using *assms that* **by** (*auto simp*: *arena-lifting all-lits-def ran-m-def all-lits-of-mm-add-mset image-Un atm-of-all-lits-of-m*(*2*)
all-atms-st-def all-atms-def simp del: *all-atms-def*[*symmetric*] *dest*!: *multi-member-split*[*of C*])
then show ‹(*atm-of* (*arena-lit* (*get-clauses-wl-heur S*) (*C* + *a*))) ∈# *all-init-atms-st S'*›
using *valid assms that no-alien* **apply** (*auto simp*: *arena-lifting*)
using *$\mathcal{L}_{all}$-all-atms in-$\mathcal{L}_{all}$-atm-of-$\mathcal{A}_{in}$* **by** *blast*
then show ‹*is-marked-added-heur-pre* (*get-heur S*) (*atm-of* (*arena-lit* (*get-clauses-wl-heur S*) (*C* + *a*)))›
using *heur* **unfolding** *heuristic-rel-def*
by (*auto simp*: *is-marked-added-heur-pre-def is-marked-added-heur-pre-stats-def*
heuristic-rel-stats-def phase-saving-def atms-of-$\mathcal{L}_{all}$-$\mathcal{A}_{in}$)
qed
show *?thesis*
using *valid*
unfolding *isa-is-candidate-forward-subsumption-def*
mop-arena-lbd-st-def mop-arena-lbd-def mop-arena-status-st-def nres-monad3
mop-arena-status-def bind-to-let-conv mop-arena-length-st-def
mop-arena-length-def 1 mop-arena-lit-def mop-arena-lit2-def

mop-is-marked-added-heur-def mop-polarity-st-heur-def mop-polarity-pol-def
apply (*refine-vcg*
WHILET-rule[**where** $R = \langle measure\ (\lambda(i,\text{-}).\ length\ (get\text{-}clauses\text{-}wl\ S' \propto C) - i)\rangle$ **and** $I = \langle\lambda(i,\text{-}).\ i \leq length\ (get\text{-}clauses\text{-}wl\ S' \propto C)\rangle$])
subgoal
using *assms* **by** (*auto simp*: *arena-act-pre-def arena-is-valid-clause-idx-def*)
subgoal
using *assms* **by** (*auto simp*: *get-clause-LBD-pre-def arena-is-valid-clause-idx-def*)
subgoal
using *assms* **by** (*auto simp*: *arena-is-valid-clause-idx-def*)
subgoal
using *assms* **by** (*auto simp*: *arena-is-valid-clause-vdom-def*)
subgoal using *arena-lifting*[*OF valid*, *of C*] *assms* **by** *auto*
subgoal
by (*auto intro*!: *RETURN-SPEC-refine*)
subgoal by *auto*
subgoal by *auto*
subgoal using *assms* **by** (*auto simp*: *arena-lit-pre-def arena-is-valid-clause-idx-and-access-def*
arena-lifting
intro!: *exI*[*of* - $\langle get\text{-}clauses\text{-}wl\ S'\rangle$] *exI*[*of* - *C*])
subgoal using *assms* **by** (*auto intro*: *H*)
subgoal using *assms* **by** (*auto simp*: *arena-lifting*)
subgoal using *assms* **by** (*auto simp*: *arena-lifting*)
subgoal by *auto*
done
qed

lemma *valid-occs-push-to-tvdom*[*intro*!]:
$\langle valid\text{-}occs\ occs\ aivdom \Longrightarrow valid\text{-}occs\ occs\ (push\text{-}to\text{-}tvdom\ C\ aivdom)\rangle$
unfolding *valid-occs-def* **by** *auto*

lemma *valid-occs-empty-vdom*[*intro*!]:
$\langle valid\text{-}occs\ occs\ vdom \Longrightarrow valid\text{-}occs\ occs\ (empty\text{-}tvdom\ vdom)\rangle$
by (*auto simp*: *valid-occs-def*)

lemma *valid-arena-vdom-le-strong*:
assumes $\langle valid\text{-}arena\ arena\ N\ ovdm\rangle$
shows $\langle card\ ovdm \leq length\ arena - 2\rangle$
proof −
have *incl*: $\langle ovdm \subseteq \{MIN\text{-}HEADER\text{-}SIZE..< length\ arena\}\rangle$
apply *auto*
using *assms valid-arena-in-vdom-le-arena* **by** *blast*+
from *card-mono*[*OF* - *this*] **show** $\langle card\ ovdm \leq length\ arena - 2\rangle$ **by** *auto*
qed

lemma
assumes *SS'*: $\langle (S,\ S') \in twl\text{-}st\text{-}heur\text{-}restart\text{-}occs'\ r\ u\rangle$
shows
twl-st-heur-restart-occs'-avdom-vdom: $\langle x1 < length\ (get\text{-}avdom\ S) \Longrightarrow get\text{-}avdom\ S\ !\ x1 \in set\ (get\text{-}vdom\ S)\rangle$ **and**
twl-st-heur-restart-occs'-ivdom-vdom: $\langle x1 < length\ (get\text{-}ivdom\ S) \Longrightarrow get\text{-}ivdom\ S\ !\ x1 \in set\ (get\text{-}vdom\ S)\rangle$ **and**
twl-st-heur-restart-occs'-length-vdom-clauses: $\langle length\ (get\text{-}vdom\ S) \leq length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S)\rangle$
and
twl-st-heur-restart-occs'-length-tvdom-clauses: $\langle length\ (get\text{-}tvdom\ S) \leq length\ (get\text{-}clauses\text{-}wl\text{-}heur$

S)› **and**
twl-st-heur-restart-occs'-length-tvdom-clauses-notin: ‹*C* ∈ *set* (*get-vdom S*) ⟹ *Suc* (*length* (*get-tvdom S*)) < *length* (*get-clauses-wl-heur S*)›
‹*C* ∈ *set* (*get-vdom S*) ⟹ (*length* (*get-tvdom S*)) < *length* (*get-clauses-wl-heur S*)› **and**
twl-st-heur-restart-occs'-length-avdom-ivdom: ‹*length* (*get-avdom S* @ *get-ivdom S*) ≤ *length* (*get-vdom S*)› **and**
twl-st-heur-restart-occs'-length-avdom-ivdom-clauses: ‹*length* (*get-avdom S* @ *get-ivdom S*) ≤ *length* (*get-clauses-wl-heur S*)›
proof −
have *arena*: ‹*valid-arena* (*get-clauses-wl-heur S*) (*get-clauses-wl S'*) (*set* (*get-vdom S*))› **and**
aivdom: ‹*aivdom-inv-dec* (*get-aivdom S*) (*dom-m* (*get-clauses-wl S'*))›
using *SS'* **unfolding** *twl-st-heur-restart-occs-def* **by** *fast+*
show ‹*x1* < *length* (*get-avdom S*) ⟹ *get-avdom S* ! *x1* ∈ *set* (*get-vdom S*)›
using *aivdom nth-mem*[*of x1* ‹*get-avdom S*›] **unfolding** *aivdom-inv-dec-alt-def*
by (*auto simp add*: *distinct-card simp del*: *nth-mem*)
show ‹*x1* < *length* (*get-ivdom S*) ⟹ *get-ivdom S* ! *x1* ∈ *set* (*get-vdom S*)›
using *aivdom nth-mem*[*of x1* ‹*get-ivdom S*›] **unfolding** *aivdom-inv-dec-alt-def*
by (*auto simp add*: *distinct-card simp del*: *nth-mem*)
show *1*: ‹*length* (*get-vdom S*) ≤ *length* (*get-clauses-wl-heur S*)›
using *valid-arena-vdom-le-strong*[*OF arena*] *aivdom* **unfolding** *aivdom-inv-dec-alt-def*
by (*simp add*: *distinct-card*)
show ‹*length* (*get-tvdom S*) ≤ *length* (*get-clauses-wl-heur S*)›
using *valid-arena-vdom-le*(*2*)[*OF arena*] *aivdom* **unfolding** *aivdom-inv-dec-alt-def*
by (*auto simp add*: *distinct-card dest*!: *size-mset-mono*)
have *WTF*: ‹*a* ≥*1* ⟹ *a* ≤ *b* − *2* ⟹ *Suc a* < *b*› **for** *a b* :: *nat*
by *auto*

have ‹*mset* (*get-avdom S* @ *get-ivdom S*) ⊆# *mset* (*get-vdom S*)›
unfolding *aivdom-inv-dec-alt-def2*
apply (*auto simp del*: *size-mset simp flip*: *size-mset*)
by (*smt* (*z3*) *List.finite-set Un-iff aivdom aivdom-inv-dec-alt-def2 mset-set-Union mset-set-set mset-set-subset-iff subset-iff*)
from *size-mset-mono*[*OF this*] **show** *2*: ‹*length* (*get-avdom S* @ *get-ivdom S*) ≤ *length* (*get-vdom S*)›
using *aivdom distinct-mset-mono*[*of* ‹*mset* (*get-avdom S*)› ‹*mset* (*get-vdom S*)›] *distinct-mset-mono*[*of* ‹*mset* (*get-ivdom S*)› ‹*mset* (*get-vdom S*)›]
unfolding *aivdom-inv-dec-alt-def2*
by (*auto simp del*: *size-mset simp flip*: *size-mset*)

show ‹*length* (*get-avdom S* @ *get-ivdom S*) ≤ *length* (*get-clauses-wl-heur S*)›
using *1 2* **by** *linarith*
assume ‹*C* ∈ *set* (*get-vdom S*)›
then have ‹*get-vdom S* ≠ []›
by *auto*
then show ‹*C* ∈ *set* (*get-vdom S*) ⟹ *Suc* (*length* (*get-tvdom S*)) < *length* (*get-clauses-wl-heur S*)›
‹*C* ∈ *set* (*get-vdom S*) ⟹ (*length* (*get-tvdom S*)) < *length* (*get-clauses-wl-heur S*)›
using *valid-arena-vdom-le-strong*[*OF arena*] *aivdom WTF*[*of* ‹*length* (*get-vdom S*)› ‹*length* (*get-clauses-wl-heur S*)›]
unfolding *aivdom-inv-dec-alt-def*
by (*auto simp add*: *distinct-card dest*!: *size-mset-mono*)
qed

lemma *isa-populate-occs*:
assumes *SS'*: ‹(*S*, *S'*) ∈ *twl-st-heur-restart-ana'* *r u*›
shows ‹*isa-populate-occs S* ≤ ⇓{(*U*,(*occs*, *cands'*)). (*U*, *S'*) ∈ *twl-st-heur-restart-occs'* *r u* ∧ (*get-tvdom U*, *cands'*)∈*Id* ∧ *get-vdom U* = *get-vdom S* ∧ (*get-occs U*, *occs*) ∈ *occurrence-list-ref*} (*populate-occs S'*)›

proof −
have *SS''*: ‹(*S*,*S'*)∈*twl-st-heur-restart-occs'* *r u*› **and**
aivdom: ‹*aivdom-inv-dec* (*get-aivdom S*) (*dom-m* (*get-clauses-wl S'*))› **and**
occs: ‹(*get-occs S*, *empty-occs-list* (*all-init-atms-st S'*)) ∈ *occurrence-list-ref*›
using *SS'* **unfolding** *twl-st-heur-restart-occs-def IsaSAT-Restart.all-init-atms-alt-def*
twl-st-heur-restart-ana-def case-wl-split state-wl-recompose twl-st-heur-restart-def
by (*auto simp add*: *valid-occs-empty*)
have [*refine*]: ‹*RETURN* (*get-occs S*) ≤ ⇓ *occurrence-list-ref* (*mop-occ-list-create* (*set-mset* (*all-init-atms-st S'*)))›
using *valid-occs-empty*[*OF occs*] *occs*
by (*auto simp*: *mop-occ-list-create-def empty-occs-list-def*)

have [*refine0*]: ‹*empty-tvdom-st S* ≤ *SPEC* (λ*c*. (*c*, []) ∈ {(*V*,*cands*). *cands*=[] ∧
(*V*,*S'*)∈*twl-st-heur-restart-occs'* (*length* (*get-clauses-wl-heur S*)) (*learned-clss-count S*) ∧
get-tvdom V = [] ∧ *get-occs V* = *get-occs S* ∧ *get-clauses-wl-heur V* = *get-clauses-wl-heur S* ∧
get-vdom V = *get-vdom S* ∧ *get-ivdom V* = *get-ivdom S*∧ *get-avdom V* = *get-avdom S*})›
(**is** ‹- ≤ *SPEC* (λ*c*. (*c*, -) ∈ *?empty*)›)
using *SS''* **unfolding** *empty-tvdom-st-def*
by (*auto simp add*: *twl-st-heur-restart-occs-def*)

have [*refine*]: ‹(*get-occs T*, *occs*) ∈ *occurrence-list-ref* ⟹ (*T*, *cands*) ∈ *?empty* ⟹
((*0*, *T*), *0*, *occs*, []) ∈ {((*w*, *U*), (*w'*, *occs*, *cands'*)).
get-avdom U = *get-avdom S* ∧
get-ivdom U = *get-ivdom S* ∧
get-vdom U = *get-vdom S* ∧
(*w*,*w'*)∈*nat-rel* ∧ (*get-tvdom U*, *cands'*)∈*Id* ∧ (*get-occs U*, *occs*)∈*occurrence-list-ref* ∧
(*U*, *S'*)∈*twl-st-heur-restart-occs'* (*length* (*get-clauses-wl-heur S*)) (*learned-clss-count S*)}› (**is** ‹- ⟹ - ⟹ - ∈ *?loop*›) **for** *occs xs cands T*
by *simp*

have *sorted*: ‹*sort-cands-by-length x2b*
≤ ⇓ {(*U*, *cands*). (*U*, *S'*) ∈ *twl-st-heur-restart-occs'* (*length* (*get-clauses-wl-heur S*)) (*learned-clss-count S*) ∧ *get-avdom U* = *get-avdom x2b* ∧
get-ivdom U = *get-ivdom x2b* ∧ *get-vdom U* = *get-vdom x2b* ∧ *get-occs U* = *get-occs x2b* ∧
get-tvdom U = *cands*}
(*SPEC*
(λ*cands'*.
mset cands' = *mset x2a* ∧
sorted-wrt (λ*a b*. *length* (*get-clauses-wl S'* ∝ *a*) ≤ *length* (*get-clauses-wl S'* ∝ *b*))
cands'))›
if
H: ‹*forward-subsumption-all-wl-pre S'*›
‹(*get-avdom S* @ *get-ivdom S*, *xs*) ∈ *Id*›
‹(*get-occs S*, *occs*) ∈ *occurrence-list-ref*›
‹(*x*, *x'*) ∈ *?loop*›
‹*x2* = (*x1a*, *x2a*)›
‹*x'* = (*x1*, *x2*)›
‹*x* = (*x1b*, *x2b*)›
‹(*S*$_0$, *cands*) ∈ *?empty*›
‹∀ *C*∈*set x2a*. *C* ∈# *dom-m* (*get-clauses-wl S'*) ∧ *C* ∈ *set xs* ∧ *2* < *length* (*get-clauses-wl S'* ∝ *C*)›
for *xs occs x x' x1 x2 x1a x2a x1b x2b cands* *S*$_0$
proof −
have *K*: ‹*sorted-wrt* (λ*a b*. *length* (*get-clauses-wl S'* ∝ *a*) ≤ *length* (*get-clauses-wl S'* ∝ *b*)) *tvdom*›
if ‹*sorted-wrt* (λ*a b*. *arena-length* (*get-clauses-wl-heur x2b*) *a* ≤ *arena-length* (*get-clauses-wl-heur x2b*) *b*)
(*tvdom*)› **and**

```
    ‹mset tvdom = mset (get-tvdom x2b)›
  for tvdom
  apply (rule sorted-wrt-mono-rel[of - ‹(λa b. arena-length (get-clauses-wl-heur x2b) a
    ≤ arena-length (get-clauses-wl-heur x2b) b)›
    ‹(λa b. length (get-clauses-wl S' ∝ a) ≤ length (get-clauses-wl S' ∝ b))›])
  subgoal for a b
  using H(1−7,9) mset-eq-setD[OF that(2)] by (auto simp: twl-st-heur-restart-occs-def IsaSAT-Restart.all-init-atms-d
      arena-lifting)
  subgoal using that(1) . done
  show ?thesis
    using that unfolding sort-cands-by-length-def
    apply refine-vcg
    subgoal
      by (auto intro!: ASSERT-leI
        simp: arena-lifting twl-st-heur-restart-occs-def
        arena-is-valid-clause-idx-def
        intro!: exI[of - ‹get-clauses-wl S'›] exI[of - ‹set (get-vdom S)›]
        dest: mset-eq-setD)
    subgoal for tvdom
      apply (subst RETURN-RES-refine-iff)
      apply (rule-tac x=tvdom in bexI)
      defer
    subgoal
      by (auto simp: twl-st-heur-restart-occs-def RETURN-RES-refine-iff valid-occs-def
        aivdom-inv-dec-alt-def simp del: distinct-mset-mset-distinct
        simp flip: distinct-mset-mset-distinct dest: mset-eq-setD
        dest!: K)
    subgoal
      apply (clarsimp simp: twl-st-heur-restart-occs-def
        aivdom-inv-dec-alt-def simp del: distinct-mset-mset-distinct
        simp flip: distinct-mset-mset-distinct dest: mset-eq-setD)
      apply (simp add: valid-occs-def)
      done
    done
  done
qed

  have [simp]: ‹¬ x1 < length (get-avdom S) ⟹ x1 − length (get-avdom S) < length (get-ivdom S)
⟷ x1 < length (get-avdom S) + length (get-ivdom S)› for x1
  by linarith
  show ?thesis
    supply [[goals-limit=1]]
    unfolding isa-populate-occs-def populate-occs-def mop-arena-length-st-def
      push-to-tvdom-st-def Let-def[of ‹length (get-avdom S)›]
     apply (refine-vcg isa-all-lit-clause-unset-all-lit-clause-unset[where r=‹length (get-clauses-wl-heur
S)› and u=‹(learned-clss-count S)›]
      mop-arena-length[where vdom=‹set(get-vdom S)›, THEN fref-to-Down-curry, unfolded comp-def]
    isa-push-to-occs-list-st-push-to-occs-list2[where r=‹length (get-clauses-wl-heur S)› and u=‹learned-clss-count
S›]
    isa-is-candidate-forward-subsumption[where r=‹length (get-clauses-wl-heur S)› and u=‹learned-clss-count
S›])
    subgoal using twl-st-heur-restart-occs'-length-avdom-ivdom-clauses[OF SS'] by auto
    subgoal using aivdom unfolding aivdom-inv-dec-alt-def by (auto dest: distinct-mset-mono)
    subgoal by auto
    apply assumption
    subgoal for xs occs Sa x x' unfolding isa-populate-occs-inv-def case-prod-beta
```

by (*rule exI*[*of* - *S'*], *rule exI*[*of* - ‹*S'*›], *rule exI*[*of* - ‹*fst* (*snd* (*x'*))›], *cases x'*) (*use SS''* **in** *clarsimp*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *access-avdom-at-pre-def*)
subgoal by (*auto simp*: *access-ivdom-at-pre-def*)
subgoal for *xs occs T x x' x1 x2 x1a x2a x1b x2b*
using
twl-st-heur-restart-occs'-avdom-vdom[*of x2b S'* - ‹*learned-clss-count S*› *x1*]
twl-st-heur-restart-occs'-ivdom-vdom[*of x2b S'* - ‹*learned-clss-count S*› ‹*x1* − *length* (*get-avdom S*)›]
by (*auto simp*: *nth-append*)
subgoal by *auto*
subgoal by (*auto simp*: *nth-append*)
subgoal
using *SS''*
unfolding *isa-all-lit-clause-unset-pre-def* **apply** −
by (*rule exI*[*of* - *S'*], *rule exI*[*of* - ‹*length* (*get-clauses-wl-heur S*)›] , *rule exI*[*of* - ‹*learned-clss-count S*›])
(*simp add*: *twl-st-heur-restart-occs'-avdom-nth-vdom twl-st-heur-restart-occs'-ivdom-nth-vdom SS''*
eq-commute[*of* ‹*get-avdom S* @ *get-ivdom S*›])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal premises *p* **using** *p*(*2*−) **by** (*clarsimp simp add*: *twl-st-heur-restart-occs-def nth-append*)
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *nth-append*)
subgoal by *auto*
subgoal by (*auto simp*: *unat32-max-def nth-append*)
subgoal by *auto*
apply (*solves auto*)
subgoal by (*auto simp*: *nth-append*)
subgoal by *auto*
subgoal by *auto*
subgoal for *xs occs T x x' x1 x2 x1a x2a x1b x2b all-undef all-undefa n cand canda*
using *twl-st-heur-restart-occs'-length-vdom-clauses*[*of x2b S'* - ‹*learned-clss-count S*›] **by** *simp*
subgoal for *xs occs T x x' x1 x2 x1a x2a x1b x2b all-undef all-undefa n cand canda*
using *twl-st-heur-restart-occs'-length-tvdom-clauses-notin*[*of x2b S'* - ‹*learned-clss-count S*› ‹*xs* ! *x1*›]
twl-st-heur-restart-occs'-avdom-vdom[*of x2b S'* - ‹*learned-clss-count S*› *x1*]
twl-st-heur-restart-occs'-ivdom-vdom[*of x2b S'* - ‹*learned-clss-count S*› ‹*x1* − *length* (*get-avdom S*)›]
by (*auto simp*: *nth-append*)
subgoal premises *p* **using** *p*(*2*−) **by** (*clarsimp intro*!: *aivdom-push-to-tvdom simp*: *nth-append twl-st-heur-restart-occs-def*)
(*intro conjI impI aivdom-push-to-tvdom*; *clarsimp*)
subgoal by *auto*
apply (*rule sorted*; *assumption*)
subgoal for *xs occs x x' x1 x2 x1a x2a x1b x2b V sorted-cands*
using *SS'* **by** (*auto simp*: *twl-st-heur-restart-ana-def*)
done
qed

lemma *empty-occs2-replicate-empty*: ‹*empty-occs2 occs* ≤ *SPEC* (λ*c*. (*c*, *replicate* (*length occs*) []) ∈

Id)›
proof −
show *?thesis*
unfolding *empty-occs2-def*
apply (*refine-vcg WHILET-rule*[**where** *R* = ‹*measure* (λ(*i*,-). *length occs* −*i*)› **and**
I = ‹ λ(*i*, *occs*′). *i* ≤ *length occs* ∧ *length occs* = *length occs*′ ∧ *take i occs*′ = *replicate i* []›])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *take-Suc-conv-app-nth replicate-Suc-conv-snoc list-update-append simp del*: *replicate-Suc*)
subgoal by *auto*
subgoal by *auto*
done
qed

lemma *empty-occs2-st-empty-occs-st*: ‹*empty-occs2-st S* ≤ ⇓*Id* (*empty-occs-st S*)›
unfolding *empty-occs2-st-def empty-occs-st-def*
by (*refine-vcg empty-occs2-replicate-empty*) *auto*

lemma *empty-occs-st*:
‹(*S*, *S*′) ∈ *twl-st-heur-restart-occs*′ *r u* ⟹
fst occs = *set-mset* (*all-init-atms-st S*′) ⟹
(*get-occs S*, *occs*) ∈ *occurrence-list-ref* ⟹
empty-occs-st S≤*SPEC*(λ*c*. (*c*,*S*′)∈*twl-st-heur-restart-ana*′ *r u*)›
unfolding *empty-occs-st-def twl-st-heur-restart-occs-def twl-st-heur-restart-ana-def IsaSAT-Restart.all-init-atms-alt-def twl-st-heur-restart-def occurrence-list-ref-def empty-occs-list-def*
by (*auto*, *auto simp*: *map-fun-rel-def*)

lemma *empty-occs2-st*:
‹(*S*, *S*′) ∈ *twl-st-heur-restart-occs*′ *r u* ⟹
fst occs = *set-mset* (*all-init-atms-st S*′) ⟹
(*get-occs S*, *occs*) ∈ *occurrence-list-ref* ⟹
empty-occs2-st S≤*SPEC*(λ*c*. (*c*,*S*′)∈*twl-st-heur-restart-ana*′ *r u*)›
apply (*rule order-trans*[*OF empty-occs2-st-empty-occs-st*])
apply (*subst Down-id-eq*)
by (*rule empty-occs-st*)

lemma [*intro*!]: ‹*heuristic-rel A heur* ⟹ *heuristic-rel A* (*reset-added-heur heur*)›
by (*auto simp*: *heuristic-rel-def heuristic-rel-stats-def reset-added-heur-def reset-added-heur-stats-def schedule-next-pure-lits-stats-def phase-saving-def*)

lemma *mop-mark-added-heur-st-it*:
assumes ‹(*S*,*T*) ∈ *twl-st-heur-restart-occs*′ *r u*› **and** ‹*A* ∈# *all-init-atms-st T*›
shows ‹*mop-mark-added-heur-st A S* ≤ *SPEC* (λ*c*. (*c*, *T*) ∈ {(*U*, *V*). (*U*, *V*) ∈ *twl-st-heur-restart-occs*′ *r u* ∧ (*get-clauses-wl-heur U*) = *get-clauses-wl-heur S* ∧
learned-clss-count U = *learned-clss-count S* ∧ *get-vdom U* = *get-vdom S* ∧ *get-occs U* = *get-occs S*})›

proof −
have *heur*: ‹*heuristic-rel* (*all-init-atms-st T*) (*get-heur S*)›
using *assms*(*1*)
by (*auto simp*: *twl-st-heur-restart-occs-def IsaSAT-Restart.all-init-atms-alt-def*)

show *?thesis*
unfolding *mop-mark-added-heur-st-def mop-mark-added-heur-def*
apply *refine-vcg*
subgoal
using *heur assms*(*2*)
unfolding *mark-added-heur-pre-def mark-added-heur-pre-stats-def*
by (*auto simp*: *heuristic-rel-def heuristic-rel-stats-def*
phase-saving-def $\mathcal{L}_{all}$*-all-atms-all-lits atms-of-def* $\mathcal{L}_{all}$*-add-mset*
dest!: *multi-member-split*)
subgoal by (*use assms* **in** ‹*auto simp add*: *twl-st-heur-restart-occs-def*›)
done
qed
lemma *mark-added-clause-heur2-id*:
assumes ‹(*S*,*T*) ∈ *twl-st-heur-restart-occs′ r u*› **and** ‹*C* ∈# *dom-m* (*get-clauses-wl T*)› **and**
lits: ‹*literals-are-*$\mathcal{L}_{in}$*′ T*›
shows ‹*mark-added-clause-heur2 S C*
≤ ⇓{(*U*, *V*). (*U*, *V*) ∈ *twl-st-heur-restart-occs′ r u* ∧ (*get-clauses-wl-heur U*) = *get-clauses-wl-heur S* ∧
learned-clss-count U = *learned-clss-count S* ∧ *get-vdom U* = *get-vdom S* ∧
get-occs U = *get-occs S*} (*RETURN T*)› (**is** ‹- ≤⇓*?R* -›)
proof −
have *1*: ‹*mark-added-clause2 T C* ≤ ⇓*Id* (*RETURN T*)›
unfolding *mark-added-clause2-def mop-clauses-at-def nres-monad3*
apply (*refine-vcg WHILEIT-rule*[**where** *R* = ‹*measure* (λ(*i*,-). *length* (*get-clauses-wl T* ∝ *C*) −*i*)›])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*use assms* **in** *auto*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
have *eq*: ‹*set-mset* (*all-init-atms-st T*) = *set-mset* (*all-atms-st T*)›
using *lits* **by** (*simp add*: *literals-are-*$\mathcal{L}_{in}$*′-all-init-atms-alt-def*)
have [*refine*]: ‹*y′* ∈# *dom-m x′* ⟹
((*x*, *y*), *x′*, *y′*) ∈ {(*N*, *N′*). *valid-arena N N′* (*set* (*get-vdom S*))} ×$_f$ *nat-rel* ⟹
mop-arena-length x y ≤ *SPEC* (λ*y*. (*y*, *length* (*x′* ∝ *y′*)) ∈ {(*a*,*b*). (*a*,*b*)∈*nat-rel* ∧ *a* = *length* (*x′* ∝ *y′*)})› **for** *x y x′ y′*
apply (*rule mop-arena-length*[*THEN fref-to-Down-curry*, *of* - - - - ‹*set* (*get-vdom S*)›, *unfolded comp-def conc-fun-RETURN prod.simps*, *THEN order-trans*])
apply *assumption*
apply *assumption*
by *auto*
have [*refine*]: ‹((*0*, *S*), *0*, *T*) ∈ *nat-rel* ×$_r$ *?R*›
using *assms* **by** *auto*
have *2*: ‹*mark-added-clause-heur2 S C* ≤ ⇓*?R* (*mark-added-clause2 T C*)›
unfolding *mark-added-clause-heur2-def mop-arena-length-st-def mop-access-lit-in-clauses-heur-def*

mark-added-clause2-def
apply (*refine-vcg mop-mark-added-heur-st-it*[**where** *r*=*r* **and** *u*=*u*])
subgoal by (*use assms* **in** *auto*)
subgoal by (*use assms* **in** ‹*auto simp*: *twl-st-heur-restart-occs-def*›)
subgoal using *assms* **by** (*auto simp*: *twl-st-heur-restart-occs-def dest*: *arena-lifting*(*10*))
subgoal by *auto*
subgoal by *auto*
apply (*rule-tac vdom* = ‹*set* (*get-vdom* (*x2a*))› **in** *mop-arena-lit2*)
subgoal by (*use assms* **in** ‹*auto simp*: *twl-st-heur-restart-occs-def*›)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*use assms* **in** ‹*auto simp*: *eq all-atms-st-def all-atms-def all-lits-def ran-m-def*
all-lits-of-mm-add-mset image-Un atm-of-all-lits-of-m(*2*)
dest!: *multi-member-split*›)
subgoal by *auto*
subgoal by *auto*
done
show *?thesis*
unfolding *mop-arena-length-st-def mop-access-lit-in-clauses-heur-def*
apply (*rule order-trans*[*OF 2*])
apply (*rule ref-two-step'*)
apply (*rule 1*[*unfolded Down-id-eq*])
done
qed

lemma *isa-forward-reset-added-and-stats*:
‹(*S*, *S'*) ∈ *twl-st-heur-restart-occs'* *r* *u* ⟹
(*isa-forward-reset-added-and-stats S*, *S'*) ∈ *twl-st-heur-restart-occs'* *r* *u*›
by (*auto simp add*: *IsaSAT-Restart.all-init-atms-alt-def twl-st-heur-restart-occs-def incr-forward-rounds-st-def*
isa-forward-reset-added-and-stats-def)

lemma [*intro*!]: ‹*heuristic-rel* *A* (*get-heur S*) ⟹ *heuristic-rel* *A* (*schedule-next-subsume delta* (*get-heur S*))›
by (*auto simp*: *schedule-next-subsume-def heuristic-rel-def heuristic-rel-stats-def*
schedule-next-subsume-stats-def)

lemma *schedule-next-subsume-st*:
‹(*S*, *S'*) ∈ *twl-st-heur-restart-occs'* *r* *u* ⟹
(*schedule-next-subsume-st delta S*, *S'*) ∈ *twl-st-heur-restart-occs'* *r* *u*›
by (*auto simp add*: *IsaSAT-Restart.all-init-atms-alt-def twl-st-heur-restart-occs-def incr-forward-rounds-st-def*
schedule-next-subsume-st-def)

lemma
get-occs-schedule-next-subsume-st[*simp*]: ‹*get-occs* (*schedule-next-subsume-st delta S*) = *get-occs S*› **and**
get-vdom-schedule-next-subsume-st[*simp*]: ‹*get-vdom* (*schedule-next-subsume-st delta S*) = *get-vdom S*›
by (*auto simp*: *schedule-next-subsume-st-def*)

lemma *forward-subsumption-finalize*:
assumes
‹(*S*, *S'*) ∈ *twl-st-heur-restart-occs'* *r* *u*›
‹*fst occs* = *set-mset* (*all-init-atms-st S'*)›
‹(*get-occs S*, *occs*) ∈ *occurrence-list-ref*› **and**
shrunken: ‹∀ *C*∈#*mset shrunken*. *C* ∈ *set* (*get-vdom S*)› **and**
lits: ‹*literals-are-*$\mathcal{L}_{in}$' *S'*›

shows ‹*forward-subsumption-finalize shrunken S*≤*SPEC*(λ*c*. (*c*,*S'*)∈*twl-st-heur-restart-ana' r u*)› (**is** ‹- ≤ *?C*›)
proof −
have
1: ‹(*RETURN S'*) ≥ *do* {
let S = *S*;
let - = *True*;
(-, *S*) ← *WHILE*$_T^{\lambda(i,S).\ i \le \text{length shrunken} \wedge S=S'}$(λ(*i*,*S*). *i* < *length shrunken*) (λ(*i*,*S*). *do* {
ASSERT (*i* < *length shrunken*);
let C = *shrunken* ! *i*;
b ← *RES* (*UNIV* :: *bool set*);
let S = *S'*;
RETURN (*i*+*1*, *S*)
}) (*0*, *S'*);
ASSERT (*S* = *S'*);
RETURN S
}›
apply (*refine-vcg WHILEIT-rule*[**where** *R* = ‹*measure* (λ(*i*,-). *length shrunken* − *i*)›])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
have [*refine*]: ‹*isasat-current-progress c* (*isa-forward-reset-added-and-stats* (*schedule-next-subsume-st delta S*)) ≤ *SPEC* (λ*c*. (*c*, *True*) ∈ *UNIV*)› **for** *c delta*
unfolding *isasat-current-progress-def* **by** *auto*
have [*refine*]: ‹((*0*, *isa-forward-reset-added-and-stats* (*schedule-next-subsume-st delta S*)), *0*, *S'*) ∈ *nat-rel* ×$_r$ {(*U*,*U'*). (*U*,*U'*) ∈ *twl-st-heur-restart-occs' r u* ∧ *get-vdom U* = *get-vdom S* ∧
get-occs U = *get-occs S*}› (**is** ‹- ∈ - ×$_r$ *?state*›) **for** *delta*
using *assms isa-forward-reset-added-and-stats*[*of* ‹*schedule-next-subsume-st delta S*› *S' r u*]
schedule-next-subsume-st[*of S S' r u*]
by (*auto simp*: *isa-forward-reset-added-and-stats-def*)
have [*refine*]: ‹*RETURN* (*clause-not-marked-to-delete-heur x2a* (*shrunken* ! *x1a*))
≤ ⇓ {(*b*,*b'*). *b*=*b'* ∧ *b'*=(*shrunken*!*x1a*∈#*dom-m* (*get-clauses-wl S'*))} (*RES UNIV*)›
if ‹(*x2a*, *S'*) ∈ *?state*› ‹*x1a* < *length shrunken*›
for *x2a x1a*
using *that shrunken arena-dom-status-iff*(*1*)[*of* ‹*get-clauses-wl-heur x2a*› ‹*get-clauses-wl S'*› ‹*set* (*get-vdom x2a*)› ‹*shrunken* ! *x1a*›]
by (*auto simp*: *clause-not-marked-to-delete-heur-def IsaSAT-Restart.all-init-atms-alt-def*
twl-st-heur-restart-occs-def conc-fun-RES)
show *?thesis*
unfolding *forward-subsumption-finalize-def*
mop-clause-not-marked-to-delete-heur-def nres-monad3
conc-fun-RETURN[*symmetric*] *Let-def*[*of* ‹*isasat-current-progress* - -›]
apply (*rule ref-two-step*[*OF* - *1*])
apply (*refine-vcg clause-not-marked-to-delete-rel*[*THEN fref-to-Down-unRET-uncurry*]
mark-added-clause-heur2-id[*unfolded conc-fun-RETURN*, *of* - *S' r u*, *THEN order-trans*]
empty-occs2-st[**where** *occs* = *occs*])
subgoal by *auto*
subgoal by *auto*
subgoal using *shrunken*

by (*auto simp*: *clause-not-marked-to-delete-heur-pre-def twl-st-heur-restart-occs-def*
arena-is-valid-clause-vdom-def
intro!: *exI*[*of* - ‹*get-clauses-wl S'*›] *exI*[*of* - ‹*set* (*get-vdom S*)›])
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *lits* **by** *auto*
apply *assumption*
subgoal by *auto*
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** (*auto simp*: *isa-forward-reset-added-and-stats-def*)
subgoal using *assms* **by** *auto*
done
qed

lemma *all-init-lits-of-wl-Pos-Neg-def*:
‹*set-mset* (*all-init-lits-of-wl S'*) = *Pos* ‘ *set-mset* (*all-init-atms-st S'*) ∪ *Neg* ‘ *set-mset* (*all-init-atms-st S'*)›
apply (*auto simp*: *all-init-atms-st-alt-def image-image*)
using *literal.exhaust-sel* **apply** *blast*
apply (*simp add*: *in-all-lits-of-wl-ain-atms-of-iff*)
apply (*simp add*: *in-all-lits-of-wl-ain-atms-of-iff*)
done

lemma *get-conflict-wl-is-None-heur-get-conflict-wl-is-None-restart-occs*:
‹(*RETURN o get-conflict-wl-is-None-heur*, *RETURN o get-conflict-wl-is-None*) ∈
twl-st-heur-restart-occs →$_f$ ⟨*Id*⟩*nres-rel*›
unfolding *get-conflict-wl-is-None-heur-def get-conflict-wl-is-None-def comp-def*
apply (*intro frefI nres-relI*) **apply** *refine-rcg*
by (*auto simp*: *twl-st-heur-restart-occs-def get-conflict-wl-is-None-heur-def get-conflict-wl-is-None-def*
option-lookup-clause-rel-def
split: *option.splits*)

lemma *mop-cch-add-all-clause-mop-ch-add-all*:
assumes ‹(*S*, *S'*) ∈ *twl-st-heur-restart-occs'* *r u*› ‹(*C*,*C'*)∈*nat-rel*›
‹(*D*, *D'*)∈*clause-hash*› **and** ‹*C'*∈#*dom-m* (*get-clauses-wl S'*)›
shows ‹*mop-cch-add-all-clause S C D* ≤ ⇓*clause-hash* (*mop-ch-add-all* (*mset* (*get-clauses-wl S'* ∝ *C'*)) *D'*)›
proof −
have *1*: ‹*mop-ch-add-all-clause S' C' D'* ≤⇓*Id* (*mop-ch-add-all* (*mset* (*get-clauses-wl S'* ∝ *C'*)) *D'*)›
unfolding *mop-ch-add-all-clause-def mop-ch-add-all-def mop-clauses-at-def mop-ch-add-def*
apply (*refine-vcg*
WHILET-rule[**where** *I*=‹λ(*i*, *D*). *i* ≤ *length* (*get-clauses-wl S'* ∝ *C'*) ∧
D=*ch-add-all* (*mset* (*take i* (*get-clauses-wl S'* ∝ *C'*))) *D'*› **and**
R = ‹*measure* (λ(*i*,-). *length* (*get-clauses-wl S'* ∝ *C'*) − *i*)›])
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *ch-add-all-def*)
subgoal by *auto*
subgoal unfolding *ch-add-pre-def ch-add-all-pre-def ch-add-all-def*
by (*auto simp*: *take-Suc-conv-app-nth distinct-in-set-take-iff*)
(*meson disjunct-not-in nth-mem-mset*)
subgoal by *auto*

subgoal by (*auto simp*: *ch-add-all-def take-Suc-conv-app-nth ch-add-def case-prod-beta*)
subgoal by *auto*
subgoal by *auto*
done
have [*refine*]: ‹((*0*, *D*), *0*, *D'*) ∈ *nat-rel* ×$_r$ *clause-hash*›
using *assms* **by** *auto*
show *?thesis*
apply (*rule ref-two-step*[*OF - 1*[*unfolded Down-id-eq*]])
unfolding *mop-cch-add-all-clause-def mop-ch-add-all-clause-def mop-arena-length-st-def*
mark-literal-for-unit-deletion-def Let-def[*of* ()]
apply (*refine-vcg mop-arena-length*[**where** *vdom* = ‹*set* (*get-vdom S*)›,*THEN fref-to-Down-curry*, *unfolded comp-def*, *of* ‹*get-clauses-wl S'*› *C'*]
mop-arena-lit[**where** *vdom*=‹*set* (*get-vdom S*)›] *mop-cch-add-mop-cch-add*)
subgoal using *assms* **by** *auto*
subgoal using *assms* **unfolding** *twl-st-heur-restart-occs-def* **by** *auto*
subgoal
using *arena-lifting*(*10*)[*of* ‹*get-clauses-wl-heur S*› ‹*get-clauses-wl S'*› ‹*set* (*get-vdom S*)› *C*] *assms*
unfolding *twl-st-heur-restart-occs-def* **by** (*auto simp*: *arena-lifting*)
subgoal by *auto*
subgoal by *auto*
subgoal using *assms* **unfolding** *twl-st-heur-restart-occs-def* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done
qed

lemma *get-occs-incr-forward-rounds-st*[*simp*]: ‹*get-occs* (*incr-forward-rounds-st S*) = *get-occs S*›
‹*get-occs* (*isa-forward-reset-added-and-stats S*) = *get-occs S*›
‹*get-clauses-wl-heur* (*incr-forward-tried-st S*) = (*get-clauses-wl-heur S*)›
‹*learned-clss-count* (*incr-forward-tried-st S*) = *learned-clss-count S*›
‹*get-aivdom* (*incr-forward-tried-st S*) = *get-aivdom S*›
‹*get-occs* (*incr-forward-tried-st S*) = *get-occs S*›
by (*auto simp*: *incr-forward-rounds-st-def isa-forward-reset-added-and-stats-def*
incr-forward-tried-st-def)

lemma *isa-forward-subsumption-all-forward-subsumption-wl-all2*:
assumes ‹(*S*, *S'*) ∈ *twl-st-heur-restart-ana' r u*›
shows ‹*isa-forward-subsumption-all S* ≤
⇓(*twl-st-heur-restart-ana' r u*) (*forward-subsumption-all-wl2 S'*)›
proof −
have *SS''*: ‹(*S*,*S'*)∈*twl-st-heur-restart-occs' r u*›
using *assms* **unfolding** *twl-st-heur-restart-occs-def IsaSAT-Restart.all-init-atms-alt-def*
twl-st-heur-restart-ana-def case-wl-split state-wl-recompose twl-st-heur-restart-def
by (*auto simp add*: *valid-occs-empty*)

have [*refine*]: ‹ (*get-occs x2a*, *occs*) ∈ *occurrence-list-ref* ∧ (*D*, *D'*) ∈*clause-hash* ∧ (*x2a*, *S'*) ∈ *twl-st-heur-restart-occs'* (*length* (*get-clauses-wl-heur S*)) (*learned-clss-count S*) ⟹
((*0*, *D*, [], *x2a*), *0*, *occs*, *D'*, *S'*, *2*, {#}) ∈
{((*m*, *D*, *shrunken*, *U*), (*n*, *occs*, *D'*, *U'*, -, *shrunken'*)). (*m*,*n*)∈*nat-rel* ∧ (*D*,*D'*)∈*clause-hash* ∧
(*get-occs U*, *occs*) ∈ *occurrence-list-ref* ∧
shrunken' = *mset shrunken* ∧
(*U*, *U'*) ∈ *twl-st-heur-restart-occs'* (*length* (*get-clauses-wl-heur S*)) (*learned-clss-count S*)∧ *get-aivdom U* = *get-aivdom x2a*}› **for** *x2a occs D D'*

by *auto*
have *H*: ‹(*S*, *S'*) ∈ *twl-st-heur-restart-occs'* *r u* ⟹
∀ *L*∈*fst* ‘ *D*$_1$ (*set-mset* (*all-init-atms-st S'*)). *L* < *length* (*get-watched-wl-heur S*)› **for** *S S'* *r u*
apply (*intro ballI impI*)
unfolding *twl-st-heur-restart-occs-def map-fun-rel-def IsaSAT-Restart.all-init-atms-alt-def*
in-pair-collect-simp $\mathcal{L}_{all}$*-all-init-atms*(2) *all-init-lits-of-wl-Pos-Neg-def*[*symmetric*]
by *normalize-goal+ auto*

have *H2*: ‹(*Sa*, *U*) ∈ *twl-st-heur-restart-occs* ⟹
(*incr-forward-tried-st Sa*, *U*) ∈ *twl-st-heur-restart-occs*› **for** *Sa U*
by (*auto simp*: *twl-st-heur-restart-occs-def incr-forward-tried-st-def*)
have *H3*: ‹*C* ∈ *set x1h* ⟹
(*Sa*, *S'*) ∈ *twl-st-heur-restart-occs* ⟹
mset x1h ⊆# *mset* (*get-tvdom Sa*) ⟹
get-vdom Sa = *get-vdom S* ⟹
C ∈ *set* (*get-vdom S*)›
for *C x1h Sa*
using *multi-member-split*[*of C* ‹*mset x1h*›] *multi-member-split*[*of C* ‹*mset* (*get-tvdom Sa*)›]
unfolding *twl-st-heur-restart-occs-def IsaSAT-Restart.all-init-atms-alt-def*
aivdom-inv-dec-alt-def
by (*auto dest*: *mset-subset-eqD dest*!: *mset-subset-eq-insertD*)

show *?thesis*
supply [[*goals-limit=1*]]
unfolding *isa-forward-subsumption-all-def*
forward-subsumption-all-wl2-def Let-def
apply (*refine-vcg ref-two-step*[*OF isa-populate-occs populate-occs-populate-occs-spec*,
unfolded Down-id-eq, *of* - - ‹*length* (*get-clauses-wl-heur S*)› ‹(*learned-clss-count S*)›]
forward-subsumption-finalize[**where** *r*=‹*length* (*get-clauses-wl-heur S*)› **and** *u*=‹(*learned-clss-count S*)›]
mop-cch-create-mop-cch-create
mop-cch-add-all-clause-mop-ch-add-all[**where** *r*=‹*length* (*get-clauses-wl-heur S*)› **and** *u*=‹(*learned-clss-count S*)›]
isa-try-to-forward-subsume-wl2-try-to-forward-subsume-wl2[**where** *r*=‹*length* (*get-clauses-wl-heur S*)› **and** *u*=‹(*learned-clss-count S*)›]
isa-forward-reset-added-and-stats[**where** *r*=‹*length* (*get-clauses-wl-heur S*)› **and** *u*=‹(*learned-clss-count S*)›])
subgoal using *assms* **unfolding** *isa-forward-subsumption-pre-all-def* **by** *blast*
subgoal using *assms* **by** (*auto simp*: *twl-st-heur-restart-ana-def*)
subgoal unfolding *forward-subsumption-all-wl-pre-def* **by** *blast*
subgoal by (*auto simp*: *isasat-fast-relaxed-def*)
subgoal by (*rule H*) *auto*
subgoal by *auto*
subgoal by *auto*
subgoal for *Sa x' x1 x2 D Da x x'a*
using *SS''* **unfolding** *isa-forward-subsumption-all-wl-inv-def* **apply** (*case-tac x*, *hypsubst*, *unfold prod.simps*)
by (*rule exI*[*of* - *S'*], *rule exI*[*of* - *S'*], *rule exI*[*of* - ‹*fst* (*snd* (*snd* (*snd x'a*)))›],
rule-tac x= ‹*fst* (*snd* (*snd* (*x'a*)))› **in** *exI*, *rule-tac x*= ‹*fst* (*snd x'a*)› **in** *exI*,
rule-tac x=‹*fst* (*snd* (*snd* (*snd* (*snd x'a*))))› **in** *exI*)
auto
subgoal by (*subst get-conflict-wl-is-None-heur-get-conflict-wl-is-None-restart-occs*[*THEN fref-to-Down-unRET-Id*])
(*auto simp*: *get-conflict-wl-is-None-def*)
subgoal by *auto*
subgoal by (*auto simp*: *access-tvdom-at-pre-def*)
subgoal by *auto*

subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *simp*
subgoal by (*clarsimp dest*!: *size-mset-mono simp add*: *forward-subsumption-all-wl2-inv-def*)
subgoal by (*auto simp add*:*H2*)
subgoal by (*clarsimp simp add*: *H3*)
apply (*solves auto*)
apply *assumption*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *assms* **by** (*auto simp*: *twl-st-heur-restart-ana-def*)
done
qed

lemma *isa-forward-subsumption-all-forward-subsumption-wl-all*:
assumes ‹(*S*, *S'*) ∈ *twl-st-heur-restart-ana' r u*›
shows ‹*isa-forward-subsumption-all S* ≤
⇓(*twl-st-heur-restart-ana' r u*) (*forward-subsumption-all-wl S'*)›
apply (*rule ref-two-step*)
apply (*rule isa-forward-subsumption-all-forward-subsumption-wl-all2*)
apply (*rule assms*)
apply (*rule forward-subsumption-all-wl2-forward-subsumption-all-wl*[*unfolded Down-id-eq*])
done

lemma *isa-forward-subsume-forward-subsume-wl*:
assumes ‹(*S*, *S'*) ∈ *twl-st-heur-restart-ana' r u*›
shows ‹*isa-forward-subsume S* ≤
⇓(*twl-st-heur-restart-ana' r u*) (*forward-subsume-wl S'*)›
proof −
have [*refine*]: ‹*RETURN* (*should-subsume-st S*) ≤ ⇓ *bool-rel* (*forward-subsume-wl-needed S'*)›
unfolding *forward-subsume-wl-needed-def* **by** *auto*
show *?thesis*
using *assms*
unfolding *forward-subsume-wl-def isa-forward-subsume-def*
by (*refine-vcg isa-forward-subsumption-all-forward-subsumption-wl-all*) *auto*
qed

end
theory *IsaSAT-Simplify-Forward-Subsumption-LLVM*
imports
IsaSAT-Simplify-Forward-Subsumption-Defs
IsaSAT-Setup-LLVM
IsaSAT-Trail-LLVM
IsaSAT-Proofs-LLVM
IsaSAT-Arena-Sorting-LLVM
IsaSAT-Show-LLVM
IsaSAT-LBD-LLVM
begin

lemma *incr-forward-subsumed-st-alt-def*: ‹*incr-forward-subsumed-st S* = (
let (*stats*, *S*) = *extract-stats-wl-heur S*; *stats* = *incr-forward-subsumed stats in*
update-stats-wl-heur stats S

)› **and**
incr-forward-strengthened-st-alt-def: ‹*incr-forward-strengthened-st S* = (
let (*stats*, *S*) = *extract-stats-wl-heur S*; *stats* = *incr-forward-strengethening stats in*
update-stats-wl-heur stats S
)› **and**
incr-forward-tried-st-alt-def: ‹*incr-forward-tried-st S* = (
let (*stats*, *S*) = *extract-stats-wl-heur S*; *stats* = *incr-forward-tried stats in*
update-stats-wl-heur stats S
)› **and**
incr-forward-rounds-st-alt-def: ‹*incr-forward-rounds-st S* = (
let (*stats*, *S*) = *extract-stats-wl-heur S*; *stats* = *incr-forward-rounds stats in*
update-stats-wl-heur stats S
)› **and**
isa-forward-reset-added-and-stats-alt-def: ‹*isa-forward-reset-added-and-stats S* = (
let (*stats*, *S*) = *extract-stats-wl-heur S*;
(*heur*, *S*) = *extract-heur-wl-heur S*;
stats = *incr-forward-rounds stats*;
heur = *reset-added-heur heur in*
update-stats-wl-heur stats (*update-heur-wl-heur heur S*))›
by (*auto simp*: *isa-push-to-occs-list-st-def state-extractors incr-forward-subsumed-st-def*
incr-forward-strengthened-st-def incr-forward-tried-st-def incr-forward-rounds-st-def
isa-forward-reset-added-and-stats-def
split: *isasat-int-splits*)

sepref-def *incr-forward-strengthened-st-impl*
is ‹*RETURN o incr-forward-strengthened-st*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
unfolding *incr-forward-strengthened-st-alt-def*
by *sepref*

sepref-def *incr-forward-tried-st-impl*
is ‹*RETURN o incr-forward-tried-st*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
unfolding *incr-forward-tried-st-alt-def*
by *sepref*

sepref-def *incr-forward-rounds-st-impl*
is ‹*RETURN o incr-forward-rounds-st*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
unfolding *incr-forward-rounds-st-alt-def*
by *sepref*

sepref-def *incr-forward-subsumed-st-impl*
is ‹*RETURN o incr-forward-subsumed-st*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
unfolding *incr-forward-subsumed-st-alt-def*
by *sepref*

sepref-def *isa-forward-reset-added-and-stats-impl*
is ‹*RETURN o isa-forward-reset-added-and-stats*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
unfolding *isa-forward-reset-added-and-stats-alt-def*
by *sepref*

sepref-register *incr-forward-subsumed-st incr-forward-strengthened-st incr-forward-rounds-st incr-forward-tried-st*
isa-forward-reset-added-and-stats

definition *clause-size-sort-clauses-raw* :: ‹*arena* ⇒ *vdom* ⇒ *nat* ⇒ *nat* ⇒ *nat list nres*› **where**
‹*clause-size-sort-clauses-raw arena N* = *pslice-sort-spec idx-clause-cdom clause-size-less arena N*›

definition *clause-size-sort-clauses-avdom* :: ‹*arena* ⇒ *vdom* ⇒ *nat list nres*› **where**
‹*clause-size-sort-clauses-avdom arena N* = *clause-size-sort-clauses-raw arena N 0* (*length N*)›

lemmas *Size-Ordering-introsort*[*sepref-fr-rules*] =
Size-Ordering-it.introsort-param-impl-correct[*unfolded clause-size-sort-clauses-raw-def*[*symmetric*] *PR-CONST-def*]

sepref-register *clause-size-sort-clauses-raw*
sepref-def *clause-size-sort-clauses-avdom-impl*
is ‹*uncurry clause-size-sort-clauses-avdom*›
:: ‹*arena-fast-assn*^{k} $*_a$ *vdom-fast-assn*^{d} $\to_a$ *vdom-fast-assn*›
supply[[*goals-limit=1*]]
unfolding *clause-size-sort-clauses-avdom-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

definition *clause-size-sort-clauses* :: ‹*arena* ⇒ *aivdom2* ⇒ *aivdom2 nres*› **where**
‹*clause-size-sort-clauses arena N* = *map-tvdom-aivdom-int* (*clause-size-sort-clauses-avdom arena*) *N*›

sepref-def *clause-size-sort-clauses-impl*
is ‹*uncurry clause-size-sort-clauses*›
:: ‹*arena-fast-assn*^{k} $*_a$ *aivdom-int-assn*^{d} $\to_a$ *aivdom-int-assn*›
unfolding *clause-size-sort-clauses-def map-tvdom-aivdom-int-def*
by *sepref*

lemma
map-vdom-aivdom-int2:
‹(*uncurry* (λ*arena*. *map-vdom-aivdom-int* (*f arena*)), *uncurry* (λ*arena*. *map-vdom-aivdom* (*f arena*)))
∈ *Id* $\times_r$ *aivdom-rel* $\to_f$ ⟨*aivdom-rel*⟩*nres-rel*›
apply (*intro frefI nres-relI*)
subgoal for *x y*
using *map-vdom-aivdom-int*[*of* ‹*f* (*fst x*)›]
apply (*cases x*; *cases y*)
apply (*auto intro*!: *frefI nres-relI simp*: *fref-def nres-rel-def*)
done
done

lemma *get-aivdom-eq-aivdom-iff*:
‹*IsaSAT-VDom.get-aivdom b* = (*x1*, *a*, *aa*, *ba*) ⟷ *b* = *AIvdom* (*x1*, *a*, *aa*, *ba*)›
by (*cases b*) *auto*
hide-const (**open**) *NEMonad.ASSERT NEMonad.RETURN NEMonad.ASSERT NEMonad.SPEC*

definition *sort-cands-by-length2* :: ‹- ⇒ *isasat-aivdom* ⇒ *isasat-aivdom nres*› **where**
‹*sort-cands-by-length2 arena avdom* = *do* {
ASSERT (∀ *i*∈*set* (*get-tvdom-aivdom avdom*). *arena-is-valid-clause-idx arena i*);
SPEC (λ*cands*′.
mset (*get-tvdom-aivdom cands*′) = *mset* (*get-tvdom-aivdom avdom*) ∧
(*get-avdom-aivdom cands*′) = (*get-avdom-aivdom avdom*) ∧
(*get-ivdom-aivdom cands*′) = (*get-ivdom-aivdom avdom*) ∧
(*get-vdom-aivdom cands*′) = (*get-vdom-aivdom avdom*) ∧
sorted-wrt (λ*a b*. *arena-length arena a* ≤ *arena-length arena b*) (*get-tvdom-aivdom cands*′))
}›

lemma *quicksort-clauses-by-score-sort*:
‹(*uncurry clause-size-sort-clauses*, *uncurry sort-cands-by-length2*) ∈
Id ×$_r$ *aivdom-rel* →$_f$ ⟨*aivdom-rel*⟩*nres-rel*›
apply (*intro fun-relI nres-relI frefI*)
subgoal for *arena arena'*
unfolding *uncurry-def sort-cands-by-length2-def map-tvdom-aivdom-int-def*
clause-size-sort-clauses-def clause-size-sort-clauses-avdom-def
clause-size-sort-clauses-raw-def pslice-sort-spec-def nres-monad3
apply (*refine-vcg*)
subgoal for *x1 x2 x1a x2a x1b x2b x1c x2c*
by (*cases x2*) (*auto simp*: *idx-clause-cdom-def code-hider-rel-def*)
subgoal for *x1 x2 x1a x2a x1b x2b x1c x2c*
apply (*rule specify-left*)
apply (*rule order-trans*)
apply (*rule slice-sort-spec-refine-sort*)
apply (*auto simp*:
pslice-sort-spec-def le-ASSERT-iff idx-cdom-def slice-rel-def br-def uncurry-def
conc-fun-RES sort-spec-def map-vdom-aivdom-int-def
code-hider-rel-def
split:*prod.splits*
simp del: *slice-complete*
intro!: *ASSERT-leI*
)
subgoal for *x1d x*
using *slice-complete*[*of x*]
apply (*rule-tac x* = ‹*AIvdom* (*x1d*, *x1b*, *x1c*, *x*)› **in** *exI*)
apply (*case-tac x2*; *auto simp*: *clause-size-less-def slice-complete*
le-by-lt-def)
unfolding *le-by-lt-def*
apply (*auto simp*: *clause-size-less-def*
intro!: *arg-cong*[*of* ‹(λ*a b*. ¬ *arena-length x1 b* < *arena-length x1 a*)› ‹(λ*a b*. *arena-length x1 a* ≤ *arena-length x1 b*)› ‹λ*a*. *sorted-wrt a x*›, *THEN iffD1*]*ext*
)
done
done
done
done

context
notes [*fcomp-norm-unfold*] = *aivdom-assn-alt-def*[*symmetric*] *aivdom-assn-def*[*symmetric*]
begin

lemma *clause-size-sort-clauses-impl-sort-cands-by-length2*[*sepref-fr-rules*]:
‹(*uncurry clause-size-sort-clauses-impl*, *uncurry sort-cands-by-length2*)
∈ (*al-assn arena-el-impl-assn*)^k ×$_a$ *aivdom-assn*^{d} →$_a$ *aivdom-assn*›
(**is** ‹*?c* ∈ [*?pre*]$_a$ *?im* → *?f*›)
proof −
have *H*: ‹*?c*
∈ [*comp-PRE* (*Id* ×$_f$ *aivdom-rel*) (λ-. *True*) (λ*x y*. *True*)
(λ*x*. *nofail* (*uncurry sort-cands-by-length2 x*))]$_a$ *?im* → *?f*›
(**is** ‹- ∈ [*?pre'*]$_a$ *?im'* → -›)
using *hfref-compI-PRE*[*OF clause-size-sort-clauses-impl.refine*,
OF quicksort-clauses-by-score-sort, *unfolded fcomp-norm-unfold*] **by** *blast*
have *pre*: ‹*?pre' x*› **if** ‹*?pre x*› **for** *x*

using *that*
by (*case-tac x*, *case-tac ‹snd x›*)
(*auto simp*: *comp-PRE-def code-hider-rel-def*)
show *?thesis*
apply (*rule hfref-weaken-pre*[*OF*])
defer
using *H* **apply** *assumption*
using *pre* **..**
qed

end

lemma *sort-cands-by-length-alt-def*:
‹*sort-cands-by-length* S_0 = *do* {
let (*aivdom*, *S*) = *extract-vdom-wl-heur* S_0;
ASSERT (*aivdom* = *get-aivdom* S_0);
let (*arena*, *S*) = *extract-arena-wl-heur S*;
ASSERT (*arena* = *get-clauses-wl-heur* S_0);
aivdom ← *sort-cands-by-length2 arena aivdom*;
let S = *update-arena-wl-heur arena S*;
let S = *update-vdom-wl-heur aivdom S*;
RETURN S
}›
apply (*auto simp*: *sort-cands-by-length-def sort-cands-by-length2-def state-extractors Let-def RES-RETURN-RES image-iff*
intro!: *bind-cong*[*OF refl*]
split: *isasat-int-splits*)
apply (*case-tac xb*)
apply *auto*
done

sepref-def *sort-cands-by-length-impl*
is *sort-cands-by-length*
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
unfolding *sort-cands-by-length-alt-def*
by *sepref*

sepref-register *mop-is-marked-added-heur-st*

sepref-def *isa-all-lit-clause-unset-impl*
is ‹*uncurry isa-all-lit-clause-unset*›
:: ‹*sint64-nat-assn*^{k} *$*$_a$ *isasat-bounded-assn*^{k} →$_a$ *bool1-assn*›
supply [[*goals-limit=1*]]
unfolding *isa-all-lit-clause-unset-def*
mop-access-lit-in-clauses-heur-def[*symmetric*] *mop-polarity-st-heur-def*[*symmetric*]
tri-bool-eq-def[*symmetric*] *UNSET-def*[*symmetric*]
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *rdomp-aivdom-assn-length-avdomD*: ‹*rdomp aivdom-assn x* ⟹ *length* (*get-avdom-aivdom x*) < *max-snat 64*›
unfolding *isasat-bounded-assn-def*
apply (*cases x*)
apply (*auto simp*: *isasat-bounded-assn-def snat64-max-def max-snat-def length-avdom-def*
aivdom-assn-def code-hider-assn-def hr-comp-def code-hider-rel-def

split: *isasat-int-splits*
dest: *al-assn-boundD*[*of sint64-nat-assn*] *mod-starD*)
done

lemma *rdomp-isasat-bounded-assn-length-avdomD*:
‹*rdomp isasat-bounded-assn x* ⟹ *length-avdom x* < *max-snat 64*›
using *rdomp-aivdom-assn-length-avdomD*[*of* ‹*get-aivdom x*›] **apply** −
unfolding *isasat-bounded-assn-def rdomp-def*
apply *normalize-goal*+
by (*cases x*)
(*force simp*: *isasat-bounded-assn-def length-avdom-def*
split: *isasat-int-splits*
dest!: *rdomp-aivdom-assn-length-avdomD mod-starD*)

sepref-register *isa-all-lit-clause-unset isa-push-to-occs-list-st*
find-best-subsumption-candidate find-best-subsumption-candidate-and-push sort-cands-by-length

sepref-def *find-best-subsumption-candidate-code*
is ‹*uncurry find-best-subsumption-candidate*›
:: ‹*sint64-nat-assn*^{k} *∗*$_a$ *isasat-bounded-assn*^{k} →$_a$ *unat-lit-assn*›
supply [[*goals-limit=1*]]
unfolding *find-best-subsumption-candidate-def*
mop-access-lit-in-clauses-heur-def[*symmetric*]
tri-bool-eq-def[*symmetric*] *UNSET-def*[*symmetric*]
length-occs-def[*symmetric*]
get-occs-list-at-def[*symmetric*]
length-occs-at-def[*symmetric*]
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *isa-push-to-occs-list-st-alt-def*:
‹*isa-push-to-occs-list-st C S* = *do* {
L ← *find-best-subsumption-candidate C S*;
let (*occs*, *T*) = *extract-occs-wl-heur S*;
ASSERT (*length* (*occs* ! *nat-of-lit L*) < *length* (*get-clauses-wl-heur S*));
occs ← *mop-cocc-list-append C occs L*;
RETURN (*update-occs-wl-heur occs T*)
}›
by (*auto simp*: *isa-push-to-occs-list-st-def state-extractors*
split: *isasat-int-splits*)

sepref-register *mop-cocc-list-append*
sepref-def *mop-cocc-list-append-impl*
is ‹*uncurry2 mop-cocc-list-append*›
:: ‹[*λ*((*C*,*occs*), *L*). *Suc* (*length* (*occs* ! *nat-of-lit L*)) < *max-snat 64*]$_a$
sint64-nat-assn^k *∗*$_a$ *occs-assn*^{d} *∗*$_a$ *unat-lit-assn*^{k} → *occs-assn*›
unfolding *mop-cocc-list-append-def cocc-list-append-pre-def cocc-list-append-def*
fold-op-list-list-push-back
by *sepref*

lemma *empty-tvdom-st-alt-def*:
‹*empty-tvdom-st S* = *do* {
let (*aivdom*, *S*) = *extract-vdom-wl-heur S*;
let aivdom = *empty-tvdom aivdom*;
RETURN (*update-vdom-wl-heur aivdom S*)

}›
by (*auto simp*: *isa-push-to-occs-list-st-def state-extractors empty-tvdom-st-def*
split: *isasat-int-splits*)

sepref-def *empty-tvdom-st-impl*
is *empty-tvdom-st*
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *empty-tvdom-st-alt-def*
by *sepref*

sepref-register *empty-tvdom-st*

sepref-def *isa-push-to-occs-list-st-impl*
is ‹*uncurry isa-push-to-occs-list-st*›
:: ‹[λ(-, *S*). *isasat-fast-relaxed S*]$_a$ *sint64-nat-assn*^{k} *$*$_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
unfolding *isa-push-to-occs-list-st-alt-def isasat-fast-relaxed-def*
by *sepref*

sepref-def *find-best-subsumption-candidate-and-push-code*
is ‹*uncurry find-best-subsumption-candidate-and-push*›
:: ‹*sint64-nat-assn*^{k} *$*$_a$ *isasat-bounded-assn*^{k} →$_a$ *unat-lit-assn* ×$_a$ *bool1-assn*›
supply [[*goals-limit=1*]]
unfolding *find-best-subsumption-candidate-and-push-def*
mop-access-lit-in-clauses-heur-def[*symmetric*]
tri-bool-eq-def[*symmetric*] *UNSET-def*[*symmetric*]
length-occs-def[*symmetric*]
get-occs-list-at-def[*symmetric*]
mop-is-marked-added-heur-st-def[*symmetric*]
length-occs-at-def[*symmetric*]
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *isa-maybe-push-to-occs-list-st-alt-def*:
‹*isa-maybe-push-to-occs-list-st C S* = *do* {
(*L*, *push*) ← *find-best-subsumption-candidate-and-push C S*;
if push then do {
let (*occs*, *T*) = *extract-occs-wl-heur S*;
ASSERT (*length* (*occs* ! *nat-of-lit L*) < *length* (*get-clauses-wl-heur S*));
occs ← *mop-cocc-list-append C occs L*;
RETURN (*update-occs-wl-heur occs T*)
} *else RETURN S*
}›
unfolding *isa-maybe-push-to-occs-list-st-def Let-def*
by (*auto simp*: *state-extractors cong*: *if-cong split*: *isasat-int-splits*)

sepref-def *isa-maybe-push-to-occs-list-st-impl*
is ‹*uncurry isa-maybe-push-to-occs-list-st*›
:: ‹[λ(-, *S*). *isasat-fast-relaxed S*]$_a$ *sint64-nat-assn*^{k} *$*$_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
unfolding *isa-maybe-push-to-occs-list-st-alt-def isasat-fast-relaxed-def*
by *sepref*

lemmas [*sepref-fr-rules*] = *arena-get-lbd.mop-refine*

sepref-def *isa-is-candidate-forward-subsumption-impl*
is ‹*uncurry isa-is-candidate-forward-subsumption*›
:: ‹*isasat-bounded-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\rightarrow_a$ *bool1-assn*›
unfolding *isa-is-candidate-forward-subsumption-def*
mop-access-lit-in-clauses-heur-def[*symmetric*]
mop-is-marked-added-heur-st-def[*symmetric*]
mop-arena-lbd-st-def[*symmetric*]
mop-arena-length-st-def[*symmetric*]
mop-arena-status-st-def[*symmetric*]
UNSET-def[*symmetric*] *tri-bool-eq-def*[*symmetric*]
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *push-to-tvdom-st-alt-def*:
‹*push-to-tvdom-st C S* = *do* {
let (*av*, *T*) = *extract-vdom-wl-heur S*;
ASSERT (*length* (*get-vdom-aivdom av*) ≤ *length* (*get-clauses-wl-heur S*));
ASSERT (*length* (*get-tvdom-aivdom av*) < *length* (*get-clauses-wl-heur S*));
let av = *push-to-tvdom C av*;
RETURN (*update-vdom-wl-heur av T*)
}›
by (*auto simp*: *isa-push-to-occs-list-st-def state-extractors push-to-tvdom-st-def*
split: *isasat-int-splits*)

sepref-def *push-to-tvdom-st-impl*
is ‹*uncurry push-to-tvdom-st*›
:: ‹[λ(-, *S*). *isasat-fast-relaxed S*]$_a$ *sint64-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
supply [[*goals-limit*=*1*]]
unfolding *push-to-tvdom-st-alt-def isasat-fast-relaxed-def*
by *sepref*

lemma *isa-populate-occs-inv-isasat-fast-relaxedI*:
‹*isa-populate-occs-inv x cands* (*a1'*, *a2'*) ⟹ *isasat-fast-relaxed x* ⟹ *isasat-fast-relaxed a2'*›
by (*auto simp*: *isa-populate-occs-inv-def isasat-fast-relaxed-def*)

sepref-def *isa-populate-occs-code*
is *isa-populate-occs*
:: ‹[*isasat-fast-relaxed*]$_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
supply [[*goals-limit*=*1*]]
supply [*dest*] = *rdomp-isasat-bounded-assn-length-avdomD isasat-bounded-assn-length-arenaD*
supply [*intro*] = *isa-populate-occs-inv-isasat-fast-relaxedI*
unfolding *isa-populate-occs-def access-avdom-at-def*[*symmetric*] *length-avdom-def*[*symmetric*] *length-ivdom-def*[*symmet*
al-fold-custom-empty[**where** *'l*=*64*] *Let-def*[*of* ‹*get-avdom - @ get-ivdom -*›] *Let-def*[*of* ‹*get-occs -*›]
Let-def[*of* ‹*get-tvdom -*›] *nth-append length-append access-ivdom-at-def*[*symmetric*]
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-register *isa-forward-subsumption-all isa-populate-occs*

sepref-register *mop-cch-create mop-cch-add-all-clause mop-cch-add mop-cch-in*

abbreviation *cch-assn* **where**
‹*cch-assn* ≡ *IICF-Array.array-assn bool1-assn*›

sepref-def *mop-cch-create-impl*

is *mop-cch-create*
:: ‹*sint64-nat-assn*^{k} $\rightarrow_a$ *cch-assn*›
unfolding *mop-cch-create-def op-list-replicate-def*[*symmetric*] *array-fold-custom-replicate*
by *sepref*

sepref-def *mop-cch-add-impl*
is ‹*uncurry mop-cch-add*›
:: ‹*unat-lit-assn*^{k} $*_a$ *cch-assn*^{d} $\rightarrow_a$ *cch-assn*›
unfolding *mop-cch-add-def cch-add-def cch-add-pre-def*
by *sepref*

sepref-def *mop-cch-in-impl*
is ‹*uncurry mop-cch-in*›
:: ‹*unat-lit-assn*^{k} $*_a$ *cch-assn*^{k} $\rightarrow_a$ *bool1-assn*›
unfolding *mop-cch-in-def cch-in-def cch-in-pre-def*
by *sepref*

sepref-def *mop-cch-add-all-clause-impl*
is ‹*uncurry2 mop-cch-add-all-clause*›
:: ‹*isasat-bounded-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *cch-assn*^{d} $\rightarrow_a$ *cch-assn*›
unfolding *mop-cch-add-all-clause-def*
mop-access-lit-in-clauses-heur-def[*symmetric*]
supply [*dest*] = *isasat-bounded-assn-length-arenaD*
supply [[*goals-limit=1*]]
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-register *isa-try-to-forward-subsume-wl2*

sepref-def *isa-try-to-forward-subsume-wl2-break-impl*
is *isa-try-to-forward-subsume-wl2-break*
:: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *bool1-assn*›
unfolding *isa-try-to-forward-subsume-wl2-break-def*
by *sepref*

sepref-register *isa-try-to-forward-subsume-wl2-break*

definition *subsumption-rel* :: ‹(*'c* × *nat*) *set*
⇒ (*'b* × *'d literal*) *set* ⇒ (*'c* × *nat*) *set* ⇒ ((*3 word* × *'b* × -) × *'d subsumption*) *set*› **where**
subsumption-rel-internal-def: ‹*subsumption-rel R1 R2 R3* = {((*tag*, *x*, *y*),*b*).
case b of NONE ⇒ *tag* = *0*
| *SUBSUMED-BY x'* ⇒ (*y*, *x'*) ∈ *R1* ∧ *tag* = *1*
| *STRENGTHENED-BY x' y'* ⇒ (*x*, *x'*) ∈ *R2* ∧ (*y*, *y'*) ∈ *R3* ∧ *tag* = *2*}›

lemma *subsumption-rel-def*: ‹⟨*R1*,*R2*,*R3*⟩*subsumption-rel* = {((*tag*, *x*, *y*),*b*).
case b of NONE ⇒ *tag* = *0*
| *SUBSUMED-BY x'* ⇒ (*y*, *x'*) ∈ *R1* ∧ *tag* = *1*
| *STRENGTHENED-BY x' y'* ⇒ (*x*, *x'*) ∈ *R2* ∧ (*y*, *y'*) ∈ *R3* ∧ *tag* = *2*}›
unfolding *subsumption-rel-internal-def relAPP-def* **by** *auto*

definition *is-NONE* **where**

‹*is-NONE x* ⟷ *NONE = x*›
lemma *is-subsumption*:
‹(λ(*tag*, -). *tag = 0*, *is-NONE*) ∈ ⟨*R1*,*R2*,*R3*⟩*subsumption-rel* → *bool-rel*›
‹(λ(*tag*, -). *tag = 1*, *is-subsumed*) ∈ ⟨*R1*,*R2*,*R3*⟩*subsumption-rel* → *bool-rel*›
‹(λ(*tag*, -). *tag = 2*, *is-strengthened*) ∈ ⟨*R1*,*R2*,*R3*⟩*subsumption-rel* → *bool-rel*›
‹((*0*, *0*, *0*), *NONE*) ∈ ⟨*R1*,*R2*,*R3*⟩*subsumption-rel*›
‹(λ*C*. (*1*, *0*, *C*), *SUBSUMED-BY*) ∈ *R1* → ⟨*R1*,*R2*,*R3*⟩*subsumption-rel*›
‹(λ*C D*. (*2*, *C*, *D*), *STRENGTHENED-BY*) ∈ *R2* → *R3* → ⟨*R1*,*R2*,*R3*⟩*subsumption-rel*›
‹(λ(*tag*, *C*, *D*). *D*, *subsumed-by*) ∈ [*is-subsumed*]$_f$ ⟨*R1*,*R2*,*R3*⟩*subsumption-rel* → *R1*›
‹(λ(*tag*, *C*, *D*). *D*, *strengthened-by*) ∈ [*is-strengthened*]$_f$ ⟨*R1*,*R2*,*R3*⟩*subsumption-rel* → *R3*›
‹(λ(*tag*, *C*, *D*). *C*, *strengthened-on-lit*) ∈ [*is-strengthened*]$_f$ ⟨*R1*,*R2*,*R3*⟩*subsumption-rel* → *R2*›
unfolding *subsumption-rel-def*
by (*auto simp*: *IS-LEFT-UNIQUE-def single-valued-def is-NONE-def*
intro!: *frefI*
split: *subsumption.splits*)

abbreviation *subsumption-raw-assn* **where**
‹*subsumption-raw-assn* ≡ *word-assn′ TYPE*(*3*) ×$_a$ *word-assn* ×$_a$ *id-assn*›

definition *subsumption-assn* **where**
‹*subsumption-assn = hr-comp subsumption-raw-assn* (⟨*snat-rel′ TYPE*(*64*), *unat-lit-rel*,*snat-rel′ TYPE*(*64*)⟩*subsumptio*

sepref-definition *is-NONE-impl*
is ‹*RETURN o* (λ(*tag*, -). *tag = 0*)›
:: ‹*subsumption-raw-assn*^{k} →$_a$ *bool1-assn*›
by *sepref*

sepref-definition *is-subsumed-impl*
is ‹*RETURN o* (λ(*tag*, -). *tag = 1*)›
:: ‹*subsumption-raw-assn*^{k} →$_a$ *bool1-assn*›
by *sepref*

sepref-definition *is-strengthened-impl*
is ‹*RETURN o* (λ(*tag*, -). *tag = 2*)›
:: ‹*subsumption-raw-assn*^{k} →$_a$ *bool1-assn*›
by *sepref*

sepref-definition *STRENGTHENED-BY-impl*
is ‹*uncurry* (*RETURN oo* (λ*C D*. (*2*, *C*, *D*)))›
:: ‹*word-assn*^{k} *$_a$ *id-assn*^{k} →$_a$ *subsumption-raw-assn*›
by *sepref*

sepref-definition *SUBSUMED-BY-impl*
is ‹*RETURN o* (λ*C*. (*1*, *0*, *C*))›
:: ‹*word-assn*^{k} →$_a$ *subsumption-raw-assn*›
by *sepref*

sepref-definition *NONE-impl*
is ‹*uncurry0* (*RETURN* (*0*, *0*, *0*::*64 word*))›
:: ‹*unit-assn*^{k} →$_a$ *subsumption-raw-assn*›
by *sepref*

sepref-definition *subsumed-by-impl*
is ‹*RETURN o* (λ(*tag*, *C*, *D*). *D*)›

:: ‹*subsumption-raw-assn*^{k} $\rightarrow_a$ *id-assn*›
by *sepref*

sepref-definition *strengthened-on-lit-impl*
is ‹*RETURN o* (λ(*tag*, *C*, *D*). *C*)›
:: ‹*subsumption-raw-assn*^{k} $\rightarrow_a$ *word-assn*›
by *sepref*

sepref-register *is-NONE is-subsumed is-strengthened STRENGTHENED-BY SUBSUMED-BY NONE subsumed-by strengthened-by strengthened-on-lit*

lemmas [*sepref-fr-rules*] =
is-NONE-impl.refine[*FCOMP is-subsumption*(*1*), *of* ‹*snat-rel*′ *TYPE*(*64*)› ‹*unat-lit-rel*› ‹*snat-rel*′ *TYPE*(*64*)›, *unfolded subsumption-assn-def*[*symmetric*] *is-NONE-def*[*symmetric*]]
is-subsumed-impl.refine[*FCOMP is-subsumption*(*2*), *of* ‹*snat-rel*′ *TYPE*(*64*)› ‹*unat-lit-rel*› ‹*snat-rel*′ *TYPE*(*64*)›, *unfolded subsumption-assn-def*[*symmetric*]]
is-strengthened-impl.refine[*FCOMP is-subsumption*(*3*), *of* ‹*snat-rel*′ *TYPE*(*64*)› ‹*unat-lit-rel*› ‹*snat-rel*′ *TYPE*(*64*)›, *unfolded subsumption-assn-def*[*symmetric*]]
SUBSUMED-BY-impl.refine[*FCOMP is-subsumption*(*5*), *of* ‹*snat-assn*′ *TYPE*(*64*)› *unat-lit-rel* ‹*snat-rel*′ *TYPE*(*64*)›, *unfolded the-pure-pure subsumption-assn-def*[*symmetric*]]
STRENGTHENED-BY-impl.refine[*FCOMP is-subsumption*(*6*), *of unat-lit-assn* ‹*snat-assn*′ *TYPE*(*64*)› ‹*snat-rel*′ *TYPE*(*64*)›, *unfolded the-pure-pure subsumption-assn-def*[*symmetric*]]
NONE-impl.refine[*FCOMP is-subsumption*(*4*), *of* ‹*snat-rel*′ *TYPE*(*64*)› *unat-lit-rel* ‹*snat-rel*′ *TYPE*(*64*)›, *unfolded the-pure-pure subsumption-assn-def*[*symmetric*]]
subsumed-by-impl.refine[*FCOMP is-subsumption*(*7*), *of* ‹*snat-assn*′ *TYPE*(*64*)› *unat-lit-rel* ‹*snat-rel*′ *TYPE*(*64*)›, *unfolded the-pure-pure subsumption-assn-def*[*symmetric*]]
subsumed-by-impl.refine[*FCOMP is-subsumption*(*8*), *of* ‹*snat-assn*′ *TYPE*(*64*)› ‹*snat-rel*′ *TYPE*(*64*)› *unat-lit-rel*, *unfolded the-pure-pure subsumption-assn-def*[*symmetric*]]
strengthened-on-lit-impl.refine[*FCOMP is-subsumption*(*9*), *of unat-lit-assn* ‹*snat-rel*′ *TYPE*(*64*)› ‹*snat-rel*′ *TYPE*(*64*)›, *unfolded the-pure-pure subsumption-assn-def*[*symmetric*]]

lemma *fold-is-NONE*: ‹*x* = *NONE* $\longleftrightarrow$ *is-NONE x*› ‹*NONE* = *x* $\longleftrightarrow$ *is-NONE x*›
by (*auto simp*: *is-NONE-def*)

lemma *isa-subsume-clauses-match2-alt-def*:
‹*isa-subsume-clauses-match2 C*′ *C N D* = *do* {
ASSERT (*isa-subsume-clauses-match2-pre C*′ *C N D*);
n ← *mop-arena-length-st N C*′;
ASSERT (*n* ≤ *length* (*get-clauses-wl-heur N*));
(*i*, *st*) ← *WHILE*$_T$ $^{\lambda(i,s).\ True}$ (λ(*i*, *st*). *i* < *n*∧ *st* ≠ *NONE*)
(λ(*i*, *st*). *do* {
ASSERT (*i* < *n*);
L ← *mop-arena-lit2* (*get-clauses-wl-heur N*) *C*′ *i*;
lin ← *mop-cch-in L D*;
if lin
then RETURN (*i*+*1*, *st*)
else do {
lin ← *mop-cch-in* (−*L*) *D*;
if lin
then if is-subsumed st
then do {*mop-free st*; *RETURN* (*i*+*1*, *STRENGTHENED-BY L C*′)}
else do {*mop-free st*; *RETURN* (*i*+*1*, *NONE*)}
else do {*mop-free st*; *RETURN* (*i*+*1*, *NONE*)}
}})
(*0*, *SUBSUMED-BY C*′);
RETURN st

}›
unfolding *isa-subsume-clauses-match2-def mop-free-def bind-to-let-conv Let-def*
by *auto*

schematic-goal *mk-free-lbd-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE subsumption-assn ?fr*›
unfolding *subsumption-assn-def* **by** *synthesize-free+*

lemma [*safe-constraint-rules*]: ‹*CONSTRAINT is-pure subsumption-assn*›
unfolding *subsumption-assn-def* **by** *auto*

sepref-register *isa-subsume-clauses-match2*
sepref-def *isa-subsume-clauses-match2-impl*
is ‹*uncurry3 isa-subsume-clauses-match2*›
:: ‹$sint64\text{-}nat\text{-}assn^k$ $*_a$ $sint64\text{-}nat\text{-}assn^k$ $*_a$ $isasat\text{-}bounded\text{-}assn^k$ $*_a$ $cch\text{-}assn^k$ $\rightarrow_a$ *subsumption-assn*›
unfolding *isa-subsume-clauses-match2-alt-def fold-is-NONE*
mop-access-lit-in-clauses-heur-def[*symmetric*]
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *mop-cch-remove-one-impl*
is ‹*uncurry mop-cch-remove-one*›
:: ‹$unat\text{-}lit\text{-}assn^k$ $*_a$ $cch\text{-}assn^d$ $\rightarrow_a$ *cch-assn*›
unfolding *mop-cch-remove-one-def*
by *sepref*

sepref-register *mop-cch-remove-one mop-arena-status-st mop-arena-promote-st*

sepref-def *swap-lits-impl* **is** ‹*uncurry3 mop-arena-swap*›
:: ‹$sint64\text{-}nat\text{-}assn^k$ $*_a$ $sint64\text{-}nat\text{-}assn^k$ $*_a$ $sint64\text{-}nat\text{-}assn^k$ $*_a$ $arena\text{-}fast\text{-}assn^d$ $\rightarrow_a$ *arena-fast-assn*›
unfolding *mop-arena-swap-def swap-lits-pre-def*
unfolding *gen-swap*
by *sepref*

sepref-def *mop-cch-remove-all-clauses-impl*
is ‹*uncurry2 mop-cch-remove-all-clauses*›
:: ‹$isasat\text{-}bounded\text{-}assn^k$ $*_a$ $sint64\text{-}nat\text{-}assn^k$ $*_a$ $cch\text{-}assn^d$ $\rightarrow_a$ *cch-assn*›
unfolding *mop-cch-remove-all-clauses-def mop-access-lit-in-clauses-heur-def*[*symmetric*]
mop-arena-length-st-def[*symmetric*]
supply [*dest*] = *isasat-bounded-assn-length-arenaD*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-register *ASize*

lemma *arena-is-valid-clause-idxD*:
assumes ‹*arena-is-valid-clause-idx a b*›
‹*rdomp* (*al-assn arena-el-impl-assn*) *a*›
‹$j \leq$ *arena-length a b*›
shows ‹$j - 2 <$ *max-unat 32*›
proof −
obtain *N vdom* **where**
‹*valid-arena a N vdom*› **and**
‹$b \in\#$ *dom-m N*›
using *assms*(*1*) **unfolding** *arena-is-valid-clause-idx-def*

```
    by auto
  then have eq: ‹length (N ∝ b) = arena-length a b› and
    le: ‹b < length a› and
    size: ‹is-Size (a ! (b − SIZE-SHIFT))›
    by (auto simp: arena-lifting)

  have ‹i<length a ⟹ rdomp arena-el-impl-assn (a ! i)› for i
    using rdomp-al-dest'[OF assms(2)]
    by auto
  from this[of ‹b − SIZE-SHIFT›] have ‹rdomp arena-el-impl-assn (a ! (b − SIZE-SHIFT))›
    using le by auto
  then have ‹length (N ∝ b) ≤ unat32-max + 2›
    using size eq unfolding rdomp-pure
    apply (auto simp: rdomp-def arena-el-impl-rel-def is-Size-def
      comp-def pure-def unat-rel-def unat.rel-def br-def arena-el-rel-def
      arena-length-def unat32-max-def)
     subgoal for x
      using unat-lt-max-unat[of x]
      apply (auto simp: max-unat-def)
      done
    done
  then show ?thesis
    using assms POS-SHIFT-def
    unfolding isa-update-pos-pre-def
    by (auto simp: arena-is-valid-clause-idx-def arena-lifting eq
      unat32-max-def max-unat-def)
qed

lemma arena-is-valid-clause-idxD2: ‹arena-is-valid-clause-idx b a ⟹ a − Suc 0 < length b›
  ‹arena-is-valid-clause-idx b a ⟹ MAX-LENGTH-SHORT-CLAUSE < arena-length b a ⟹ 3 ≤ a›
  ‹arena-is-valid-clause-idx b a ⟹ MAX-LENGTH-SHORT-CLAUSE < arena-length b a ⟹ a − 3 <
length b›
  using arena-lengthI(2) less-imp-diff-less apply blast
 apply (auto simp: arena-is-valid-clause-idx-def header-size-def arena-lifting split: if-splits dest: arena-lifting(1)[of
- - - a])
  apply (metis arena-header-size-def arena-lifting(1) valid-arena-header-size)
  using valid-arena-def by fastforce

sepref-def mop-arena-shorten-impl
  is ‹uncurry2 mop-arena-shorten›
  :: ‹sint64-nat-assn^k *_a sint64-nat-assn^k *_a arena-fast-assn^d →_a arena-fast-assn›
 unfolding mop-arena-shorten-def arena-shorten-def arena-shorten-pre-def SIZE-SHIFT-def POS-SHIFT-def
    arena-is-valid-clause-idxD2
  supply [intro] = arena-lengthI arena-is-valid-clause-idxD arena-is-valid-clause-idxD2
  apply (rewrite at ‹APos ⌑› unat-const-fold[where 'a=32])
  apply (annot-snat-const ‹TYPE(64)›)
  apply (rewrite at ‹If - (-[- := ASize ⌑, - := -]) -› annot-snat-unat-downcast[where 'l=32])
  apply (rewrite at ‹If - - (-[- := ASize ⌑])› annot-snat-unat-downcast[where 'l=32])
  by sepref

sepref-def remove-lit-from-clause-impl
  is ‹uncurry2 remove-lit-from-clause›
  :: ‹arena-fast-assn^d *_a sint64-nat-assn^k *_a unat-lit-assn^k →_a arena-fast-assn›
  unfolding remove-lit-from-clause-def if-not-swap
  apply (annot-snat-const ‹TYPE(64)›)
  by sepref
```

lemma *remove-lit-from-clause-st-alt-def*: ‹*remove-lit-from-clause-st S C L* = *do* {
  let (*N*, *S*) = *extract-arena-wl-heur S*;
  N ← *remove-lit-from-clause N C L*;
  RETURN (*update-arena-wl-heur N S*)
}›
  by (*auto simp*: *remove-lit-from-clause-st-def state-extractors push-to-tvdom-st-def*
      split: *isasat-int-splits*)

sepref-def *remove-lit-from-clause-st-impl*
  is ‹*uncurry2 remove-lit-from-clause-st*›
  :: ‹*isasat-bounded-assn*^{d} *$*_a$ *sint64-nat-assn*^{k} *$*_a$ *unat-lit-assn*^{k} →$_a$ *isasat-bounded-assn*›
  unfolding *remove-lit-from-clause-st-alt-def*
  by *sepref*

sepref-register *remove-lit-from-clause-st*

lemma *mark-garbage-heur-as-subsumed-alt-def*:
  ‹*mark-garbage-heur-as-subsumed C S*$_0$ = (*do*{
    ASSERT (*arena-is-valid-clause-vdom* (*get-clauses-wl-heur S*$_0$) *C*);
    - ← *log-del-clause-heur S*$_0$ *C*;
    ASSERT (*mark-garbage-pre* (*get-clauses-wl-heur S*$_0$, *C*));
    size ← *mop-arena-length* (*get-clauses-wl-heur S*$_0$) *C*;
    let (*N'*, *S*) = *extract-arena-wl-heur S*$_0$;
    ASSERT (*N'* = *get-clauses-wl-heur S*$_0$);
    let st = *arena-status N' C* = *IRRED*;
    let N' = *extra-information-mark-to-delete* (*N'*) *C*;
    let (*lcount*, *S*) = *extract-lcount-wl-heur S*;
    ASSERT(¬*st* ⟶ *clss-size-lcount lcount* ≥ *1*);
    let lcount = (*if st then lcount else* (*clss-size-decr-lcount lcount*));
    let (*stats*, *S*) = *extract-stats-wl-heur S*;
    let stats = (*if st then decr-irred-clss stats else stats*);
    let S = *update-arena-wl-heur N' S*;
    let S = *update-lcount-wl-heur lcount S*;
    let S = *update-stats-wl-heur stats S*;
    let S = *incr-wasted-st* (*of-nat size*) *S*;
    RETURN S
  })›
  by (*auto simp*: *mark-garbage-heur-as-subsumed-def Let-def state-extractors push-to-tvdom-st-def*
    intro!: *bind-cong*[*OF refl*]
    split: *isasat-int-splits*)

sepref-def *mark-garbage-heur-as-subsumed-impl*
  is ‹*uncurry mark-garbage-heur-as-subsumed*›
  :: ‹*sint64-nat-assn*^{k} *$*_a$ *isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
  supply [[*goals-limit=1*]]
  supply *of-nat-snat*[*sepref-import-param*]
  unfolding *mark-garbage-heur-as-subsumed-alt-def*
    mop-arena-length-st-def[*symmetric*]
  by *sepref*

sepref-def *isa-strengthen-clause-wl2-impl*
  is ‹*uncurry3 isa-strengthen-clause-wl2*›
 :: ‹*sint64-nat-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} *$*_a$ *unat-lit-assn*^{k} *$*_a$ *isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
  unfolding *isa-strengthen-clause-wl2-def mop-arena-status-st-def*[*symmetric*]
    mop-arena-length-st-def[*symmetric*]

apply (*subst incr-forward-strengthened-st-def*[*symmetric*])+
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *subsumption-cases-split*:
‹(*case s of SUBSUMED-BY s* ⇒ *f s* | *STRENGTHENED-BY x y* ⇒ *g x y* | *NONE* ⇒ *h*) =
(*if is-NONE s then h else if is-subsumed s then f* (*subsumed-by s*) *else do* {*ASSERT* (*is-strengthened s*); *g* (*strengthened-on-lit s*) (*strengthened-by s*)})›
by (*auto simp*: *is-NONE-def split*: *subsumption.splits*)

sepref-register *isa-strengthen-clause-wl2 isa-subsume-or-strengthen-wl*
sepref-def *isa-subsume-or-strengthen-wl-impl*
is ‹*uncurry2 isa-subsume-or-strengthen-wl*›
:: ‹*sint64-nat-assn*^{k} *∗*$_a$ *subsumption-assn*^{k} *∗*$_a$ *isasat-bounded-assn*^{d} *→*$_a$ *isasat-bounded-assn*›
unfolding *isa-subsume-or-strengthen-wl-def subsumption-cases-split mop-arena-status-st-def*[*symmetric*]
incr-forward-subsumed-st-def[*symmetric*]
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *isa-forward-subsumption-one-wl-impl*
is ‹*uncurry3 isa-forward-subsumption-one-wl*›
:: ‹[*λ*((-, -), *S*). *isasat-fast-relaxed S*]$_a$ *sint64-nat-assn*^{k} *∗*$_a$ *cch-assn*^{d} *∗*$_a$ *unat-lit-assn*^{k} *∗*$_a$ *isasat-bounded-assn*^{d} *→ isasat-bounded-assn ×*$_a$ *subsumption-assn ×*$_a$ *cch-assn*›
supply [*dest*] = *rdomp-isasat-bounded-assn-length-avdomD isasat-bounded-assn-length-arenaD*
supply [[*goals-limit=1*]]
unfolding *isa-forward-subsumption-one-wl-def get-occs-list-at-def*[*symmetric*] *fold-is-NONE*
mop-access-lit-in-clauses-heur-def[*symmetric*] *length-occs-at-def*[*symmetric*] *mop-arena-status-st-def*[*symmetric*]
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *isa-try-to-forward-subsume-wl-invI*:
‹*isa-try-to-forward-subsume-wl-inv S C* (*i*, *changed*, *break*, *D*, *T*)⟹ *isasat-fast-relaxed S* ⟹ *isasat-fast-relaxed T*›
unfolding *isa-try-to-forward-subsume-wl-inv-def prod.simps*
by *normalize-goal+* (*auto simp add*: *isasat-fast-relaxed-def*)

lemma *isasat-bounded-assn-get-vdomD*: ‹*rdomp isasat-bounded-assn a* ⟹ *length* (*get-tvdom a*) < *max-snat 64*›
using *al-assn-boundD*[*of sint64-nat-assn*, **where** *'l*=‹*64*›, *of* ‹*get-tvdom a*›]
apply −
unfolding *rdomp-def*
apply *normalize-goal+*
apply (*cases a*, *case-tac xa*; *cases* ‹*get-aivdom a*›)
apply (*auto 7 5 simp*: *isasat-bounded-assn-def snat64-max-def max-snat-def aivdom-assn-def*
code-hider-assn-def hr-comp-def code-hider-rel-def import-param-3 pred-lift-def
split: *isasat-int-splits*
dest!: *mod-starD al-assn-boundD*[*of sint64-nat-assn*, **where** *'l*=‹*64*›])
apply *auto*
done

sepref-def *isa-try-to-forward-subsume-wl2-impl*
is ‹*uncurry3 isa-try-to-forward-subsume-wl2*›
:: ‹[*λ*(((-, -), -), *S*). *isasat-fast-relaxed S*]$_a$ *sint64-nat-assn*^{k} *∗*$_a$ *cch-assn*^{d} *∗*$_a$ (*al-assn' TYPE*(*64*) *sint64-nat-assn*)^d *∗*$_a$ *isasat-bounded-assn*^{d} *→*
cch-assn ×$_a$ *al-assn' TYPE*(*64*) *sint64-nat-assn ×*$_a$ *isasat-bounded-assn*›

unfolding *isa-try-to-forward-subsume-wl2-def*
mop-access-lit-in-clauses-heur-def[*symmetric*] *Let-def*[*of* ‹*is-strengthened* -›] *fold-is-NONE*
Let-def[*of* ‹*if is-strengthened - then - else* -›]
supply [[*goals-limit=1*]]
supply [*intro*] = *isa-try-to-forward-subsume-wl-invI*
supply [*dest*] = *isasat-bounded-assn-get-vdomD*
apply (*rewrite at* ‹*if* - *then mark-clause-for-unit-as-unchanged* ⌑ *else* -› *unat-const-fold*[**where** *'a*=‹*64*›])
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *empty-occs2-st-alt-def*:
‹*empty-occs2-st S* = *do* {
let (*occs*, *S*) = *extract-occs-wl-heur S*;
occs ← *empty-occs2 occs*;
RETURN (*update-occs-wl-heur occs S*)
}›
by (*auto simp*: *empty-occs2-st-def Let-def state-extractors*
intro!: *bind-cong*[*OF refl*]
split: *isasat-int-splits*)

sepref-def *empty-occs2-impl*
is ‹*empty-occs2*›
:: ‹*occs-assn*^{d} →$_a$ *occs-assn*›
unfolding *empty-occs2-def fold-op-list-list-take op-list-list-len-def*[*symmetric*]
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *empty-occs2-st-impl*
is ‹*empty-occs2-st*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
unfolding *empty-occs2-st-alt-def*
by *sepref*

lemma *isa-forward-subsumption-all-wl-invI*:
‹*isa-forward-subsumption-all-wl-inv R S* (*i*, *D*, *shrunken*, *T*) ⟹ *isasat-fast-relaxed R* ⟹ *isasat-fast-relaxed T*›
unfolding *isa-forward-subsumption-all-wl-inv-def prod.simps*
apply *normalize-goal*+
by (*auto simp*: *isasat-fast-relaxed-def*)

sepref-register *empty-occs2-st forward-subsumption-finalize schedule-next-subsume-st*

sepref-def *mark-added-clause-heur2-impl*
is ‹*uncurry mark-added-clause-heur2*›
:: ‹*isasat-bounded-assn*^{d} *$*_a$ *sint64-nat-assn*^{k} →$_a$ *isasat-bounded-assn*›
unfolding *mark-added-clause-heur2-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *schedule-next-subsume-st-alt-def*:
‹*schedule-next-subsume-st b S* = (*let* (*heur*, *S*) = *extract-heur-wl-heur S*;
heur = *schedule-next-subsume b heur in*
update-heur-wl-heur heur S)›
by (*auto simp*: *schedule-next-subsume-st-def Let-def state-extractors*
split: *isasat-int-splits*)

sepref-def *schedule-next-subsume-st*
  is ‹*uncurry* (*RETURN oo schedule-next-subsume-st*)›
  :: ‹*word64-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
  supply [[*goals-limit=1*]]
  unfolding *schedule-next-subsume-st-alt-def*
  by *sepref*

sepref-def *forward-subsumption-finalize*
  is ‹*uncurry forward-subsumption-finalize*›
  :: ‹(*al-assn*$'$ *TYPE*(*64*) *sint64-nat-assn*)^k $*_a$ *isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
  unfolding *forward-subsumption-finalize-def*
  supply [[*goals-limit=1*]]
  apply (*annot-snat-const* ‹*TYPE*(*64*)›)
  by *sepref*

sepref-def *isa-forward-subsumption-all-impl*
  is *isa-forward-subsumption-all*
  :: ‹[*isasat-fast-relaxed*]$_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *isasat-bounded-assn*›
  supply [[*goals-limit=1*]]
  supply [*intro*] = *isa-forward-subsumption-all-wl-invI*
  unfolding *isa-forward-subsumption-all-def*
    access-tvdom-at-def[*symmetric*] *length-tvdom-def*[*symmetric*]
    length-watchlist-raw-def[*symmetric*]
    al-fold-custom-empty[**where** *'l=64*]
  apply (*annot-snat-const* ‹*TYPE*(*64*)›)
  by *sepref*

lemma *get-subsumption-opts-alt-def*:
  ‹*get-subsumption-opts S* = (*case S of IsaSAT M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs* ⇒ *opts-subsumption opts*)›
  by (*cases S*) (*auto simp*: *get-subsumption-opts-def*)

sepref-def *get-subsumption-opts-impl*
  is ‹*RETURN o get-subsumption-opts*›
  :: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *bool1-assn*›
  unfolding *get-subsumption-opts-alt-def*
  by *sepref*

lemma *next-subsume-schedule-st-def*:
  ‹*next-subsume-schedule-st S* = (*case S of IsaSAT M N D i W ivmtf icount ccach lbd outl stats heur aivdom clss opts arena occs* ⇒ *next-subsume-schedule heur*)›
  by (*cases S*) (*auto simp*: *next-subsume-schedule-st-def*)

sepref-def *next-subsume-schedule-st-impl*
  is ‹*RETURN o next-subsume-schedule-st*›
  :: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *word-assn*›
  unfolding *next-subsume-schedule-st-def*
  by *sepref*

sepref-def *should-subsume-st*
  is ‹*RETURN o should-subsume-st*›
  :: ‹*isasat-bounded-assn*^{k} $\rightarrow_a$ *bool1-assn*›
  unfolding *should-subsume-st-def*
  by *sepref*

sepref-def *isa-forward-subsume-impl*
is *isa-forward-subsume*
:: ‹$[$*isasat-fast-relaxed*$]_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
unfolding *isa-forward-subsume-def*
by *sepref*

end

theory *IsaSAT-Restart-Inprocessing-Defs*
imports *IsaSAT-Setup*
IsaSAT-Simplify-Units-Defs
Watched-Literals.Watched-Literals-Watch-List-Inprocessing
More-Refinement-Libs.WB-More-Refinement-Loops
IsaSAT-Restart-Defs
IsaSAT-Simplify-Binaries-Defs
IsaSAT-Simplify-Pure-Literals-Defs
IsaSAT-Simplify-Forward-Subsumption-Defs
begin

definition *isa-pure-literal-elimination-round-wl* **where**
‹*isa-pure-literal-elimination-round-wl* S_0 = *do* {
ASSERT (*isa-pure-literal-elimination-round-wl-pre* S_0);
$S \leftarrow$ *isa-simplify-clauses-with-units-st-wl2* S_0;
ASSERT (*length* (*get-clauses-wl-heur* S) = *length* (*get-clauses-wl-heur* S_0));
ASSERT (*learned-clss-count* $S \leq$ *learned-clss-count* S_0);
if get-conflict-wl-is-None-heur S
then do {
(*abort*, *occs*) ← *isa-pure-literal-count-occs-wl* (S);
if ¬*abort then isa-pure-literal-deletion-wl occs* S
else RETURN (*0*, S)}
else RETURN (*0*, S)
}›

definition *isa-pure-literal-elimination-wl-pre* :: ‹-› **where**
‹*isa-pure-literal-elimination-wl-pre* S = ($\exists$ T u r.
(S, T) ∈ *twl-st-heur-restart-ana'* r u ∧ *pure-literal-elimination-wl-pre* T)›

definition *isa-pure-literal-elimination-wl-inv* :: ‹-› **where**
‹*isa-pure-literal-elimination-wl-inv* S *max-rounds* = (λ(T,*m*,*abort*). $\exists$ S' T' u r.
(S, S') ∈ *twl-st-heur-restart-ana'* r u ∧
(T, T') ∈ *twl-st-heur-restart-ana'* r u ∧ *pure-literal-elimination-wl-inv* S' *max-rounds* (T', *m*, *abort*))›

definition *isa-pure-literal-elimination-wl* :: ‹*isasat* ⇒ *isasat nres*› **where**
‹*isa-pure-literal-elimination-wl* S_0 = *do* {
ASSERT (*isa-pure-literal-elimination-wl-pre* S_0);
max-rounds ← *RETURN* (*3*::*nat*);
(S, -, -) ← *WHILE*$_T$*isa-pure-literal-elimination-wl-inv* S_0 *max-rounds* (λ(S, *m*, *abort*). *m* < *max-rounds* ∧ ¬*abort*)
(λ(S, *m*, *abort*). *do* {
ASSERT (*m* ≤ *max-rounds*);
ASSERT (*length* (*get-clauses-wl-heur* S) = *length* (*get-clauses-wl-heur* S_0));
ASSERT (*learned-clss-count* S ≤ *learned-clss-count* S_0);
let S = *incr-purelit-rounds-st* S;

```
        (elim, S) ← isa-pure-literal-elimination-round-wl S;
        abort ← RETURN (elim = 0);
        RETURN (S, m+1, abort)
      })
    (S₀, 0, False);
   RETURN (schedule-next-pure-lits-st S)
  }›

definition isa-pure-literal-eliminate :: ‹isasat ⇒ isasat nres› where
  ‹isa-pure-literal-eliminate S = do {
    let b = should-eliminate-pure-st S;
    if b then isa-pure-literal-elimination-wl S else RETURN S
}›

end

theory IsaSAT-Restart-Inprocessing
  imports IsaSAT-Setup
   IsaSAT-Simplify-Units
    Watched-Literals.Watched-Literals-Watch-List-Inprocessing
    More-Refinement-Libs.WB-More-Refinement-Loops
    IsaSAT-Restart
    IsaSAT-Simplify-Binaries
    IsaSAT-Simplify-Pure-Literals
    IsaSAT-Simplify-Forward-Subsumption
    IsaSAT-Restart-Inprocessing-Defs
begin

lemma isa-simplify-clauses-with-unit-st2-isa-simplify-clauses-with-unit-wl:
  assumes ‹(S,S') ∈ twl-st-heur-restart-ana' r u›
  shows
   ‹isa-simplify-clauses-with-units-st-wl2 S ≤ ⇓ (twl-st-heur-restart-ana' r u) (simplify-clauses-with-units-st-wl
S')›
  apply (rule order-trans)
    defer
  apply (rule ref-two-step')
  apply (rule simplify-clauses-with-units-st-wl2-simplify-clauses-with-units-st-wl[unfolded Down-id-eq, of
- S'])
  subgoal by auto
  subgoal
    apply (rule isa-simplify-clauses-with-units-st2-simplify-clauses-with-units-st2[THEN order-trans, of
- S'])
    apply (rule assms)
    subgoal using assms by auto
    done
  done


lemma incr-purelit-rounds-st-twl-st-heur-restart-ana':
  assumes S₀T: ‹(S₀, T) ∈ twl-st-heur-restart-ana' r u›
  shows ‹(incr-purelit-rounds-st S₀, T) ∈twl-st-heur-restart-ana' r u›
  using assms by (auto simp: incr-purelit-rounds-st-def twl-st-heur-restart-def
    twl-st-heur-restart-ana-def)

lemma isa-pure-literal-elimination-round-wl-pure-literal-elimination-round-wl:
  assumes S₀T: ‹(S₀, T) ∈ twl-st-heur-restart-ana' r u›
```

shows ‹*isa-pure-literal-elimination-round-wl* S_0 ≤⇓{((-, U), V). (U,V)∈*twl-st-heur-restart-ana′* r u}
(*pure-literal-elimination-round-wl* T)›
proof −
show *?thesis*
unfolding *isa-pure-literal-elimination-round-wl-def pure-literal-elimination-round-wl-def Let-def*[*of* ‹*incr-purelit-rounds-st* -›]
apply (*refine-rcg isa-pure-literal-deletion-wl-pure-literal-deletion-wl*[**where** r=r **and** u=‹*learned-clss-count* S_0›]
isa-pure-literal-count-occs-wl-pure-literal-count-occs-wl[**where** r=r **and** u=‹*learned-clss-count* S_0›]
isa-simplify-clauses-with-unit-st2-isa-simplify-clauses-with-unit-wl[**where** r=r **and** u=‹*learned-clss-count* S_0›])
subgoal using *assms* **unfolding** *isa-pure-literal-elimination-round-wl-pre-def* **by** *fast*
subgoal using *assms* **by** (*auto simp*: *twl-st-heur-restart-ana-def*)
subgoal using *assms* **by** (*auto simp*: *twl-st-heur-restart-ana-def*)
subgoal using *assms* **by** (*auto simp*: *twl-st-heur-restart-ana-def*)
subgoal
by (*subst get-conflict-wl-is-None-heur-get-conflict-wl-is-None-ana*[*THEN fref-to-Down-unRET-Id*])
(*use assms* **in** ‹*auto simp*: *get-conflict-wl-is-None-def*›)
subgoal by *auto*
apply (*rule order-trans*[*OF*])
apply (*rule isa-pure-literal-deletion-wl-pure-literal-deletion-wl*[**where** r=r **and** u=‹*learned-clss-count* S_0›])
prefer *3*
apply (*rule conc-fun-R-mono*)
subgoal using *assms* **by** *auto*
subgoal by *auto*
subgoal by *auto*
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
done
qed

lemma *schedule-next-pure-lits-st-twl-st-heur-restart-ana′*:
assumes ‹(S,S') ∈ *twl-st-heur-restart-ana′* r u› ‹u ≤ u'›
shows ‹(*schedule-next-pure-lits-st* S, S') ∈ *twl-st-heur-restart-ana′* r u'›
using *assms* **by** (*auto simp*: *schedule-next-pure-lits-st-def twl-st-heur-restart-ana-def*
twl-st-heur-restart-def)

lemma *isa-pure-literal-elimination-wl-pure-literal-elimination-wl*:
assumes ‹(S, S') ∈ *twl-st-heur-restart-ana′* r u›
shows ‹*isa-pure-literal-elimination-wl* S ≤⇓(*twl-st-heur-restart-ana′* r u) (*pure-literal-elimination-wl*
S')›
proof −
have [*refine*]: ‹*RETURN* (*3*::*nat*) ≤ ⇓ {(a,b). a = b} (*RES UNIV*)›
‹*RETURN* (*x1d* = *0*) ≤ ⇓ *bool-rel* (*RES UNIV*)› **for** *x1d*
by (*auto simp*: *RETURN-RES-refine*)
have [*refine*]: ‹((*S*, *0*, *False*), *S′*, *0*, *False*) ∈ *twl-st-heur-restart-ana′* r (*learned-clss-count* S) ×$_r$ *Id*
×$_r$ *Id*›
using *assms* **by** *auto*
show *?thesis*
unfolding *isa-pure-literal-elimination-wl-def pure-literal-elimination-wl-def Let-def*[*of* ‹*incr-purelit-rounds-st*
-›]
apply (*refine-vcg isa-pure-literal-elimination-round-wl-pure-literal-elimination-round-wl*[**where** r=r
and u=‹*learned-clss-count* S›]
schedule-next-pure-lits-st-twl-st-heur-restart-ana′[**where** r=r **and** u=‹*learned-clss-count* S› **and**

u'=u])
  subgoal using *assms* **unfolding** *isa-pure-literal-elimination-wl-pre-def* **by** *fast*
  subgoal for *max-rounds max-roundsa x x'*
    using *assms* **unfolding** *isa-pure-literal-elimination-wl-inv-def case-prod-beta prod-rel-fst-snd-iff*
    by (*rule-tac x*=S' **in** *exI*, *rule-tac x*=‹*fst x'*› **in** *exI*) *auto*
  subgoal by *auto*
  subgoal by *auto*
  subgoal using *assms* **by** (*auto simp*: *twl-st-heur-restart-ana-def*)
  subgoal by *auto*
  subgoal by (*rule incr-purelit-rounds-st-twl-st-heur-restart-ana'*) *auto*
  subgoal by *auto*
  subgoal using *assms* **by** *auto*
  subgoal using *assms* **by** *auto*
  done
qed

lemma *isa-pure-literal-eliminate*:
  assumes ‹(S, S') ∈ *twl-st-heur-restart-ana' r u*›
  shows ‹*isa-pure-literal-eliminate S* ≤⇓(*twl-st-heur-restart-ana' r u*) (*pure-literal-eliminate-wl S'*)›
proof −
  have [*refine*]: ‹*RETURN* (*should-eliminate-pure-st S*) ≤ ⇓ *bool-rel* (*pure-literal-eliminate-wl-needed S'*)›
    unfolding *pure-literal-eliminate-wl-needed-def* **by** *auto*
  show *?thesis*
    unfolding *isa-pure-literal-eliminate-def pure-literal-eliminate-wl-def*
    by (*refine-vcg isa-pure-literal-elimination-wl-pure-literal-elimination-wl*)
     (*use assms* **in** *auto*)
qed

end
theory *IsaSAT-Inprocessing-LLVM*
  imports
    IsaSAT-Restart-Inprocessing-Defs
    IsaSAT-Simplify-Units-LLVM
    IsaSAT-Simplify-Binaries-LLVM
    IsaSAT-Simplify-Forward-Subsumption-LLVM
    IsaSAT-Simplify-Pure-Literals-LLVM
begin

sepref-register *0 1*

sepref-register *mop-arena-update-lit isa-pure-literal-count-occs-wl*
    isa-pure-literal-elimination-round-wl incr-purelit-rounds-st

sepref-def *should-inprocess-st*
  is ‹*RETURN o should-inprocess-st*›
  :: ‹*isasat-bounded-assn*^{k} →$_a$ *bool1-assn*›
  unfolding *should-inprocess-st-def*
  by *sepref*

lemma [*simp*]: ‹*get-clauses-wl-heur* (*incr-purelit-rounds-st S*) = *get-clauses-wl-heur S*›
  ‹*learned-clss-count* (*incr-purelit-rounds-st S*) = *learned-clss-count S*›
  by (*auto simp*: *incr-purelit-rounds-st-def*)

sepref-def *isa-pure-literal-elimination-round-wl-code*
is *isa-pure-literal-elimination-round-wl*
:: ‹$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq snat64\text{-}max \wedge learned\text{-}clss\text{-}count\ S \leq unat64\text{-}max]_a$
isasat-bounded-assn$^d \to$ *word64-assn* $\times_a$ *isasat-bounded-assn*›
unfolding *isa-pure-literal-elimination-round-wl-def*
by *sepref*

lemma *schedule-next-pure-lits-st-alt-def*:
‹*schedule-next-pure-lits-st S* =
(*let* (*heur*, *S*) = *extract-heur-wl-heur S*;
heur = (*schedule-next-pure-lits* (*heur*))*in*
update-heur-wl-heur heur S)›
by (*auto simp*: *schedule-next-pure-lits-st-def state-extractors split*: *isasat-int-splits*
intro!: *ext*)

sepref-def *schedule-next-pure-lits-st-impl*
is ‹*RETURN o schedule-next-pure-lits-st*›
:: ‹*isasat-bounded-assn*$^d \to_a$ *isasat-bounded-assn*›
unfolding *schedule-next-pure-lits-st-alt-def*
by *sepref*

sepref-def *isa-pure-literal-elimination-wl-code*
is *isa-pure-literal-elimination-wl*
:: ‹$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq snat64\text{-}max \wedge learned\text{-}clss\text{-}count\ S \leq unat64\text{-}max]_a$
isasat-bounded-assn$^d \to$ *isasat-bounded-assn*›
unfolding *isa-pure-literal-elimination-wl-def Let-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *isa-pure-literal-eliminate-code*
is *isa-pure-literal-eliminate*
:: ‹$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq snat64\text{-}max \wedge learned\text{-}clss\text{-}count\ S \leq unat64\text{-}max]_a$
isasat-bounded-assn$^d \to$ *isasat-bounded-assn*›
unfolding *isa-pure-literal-eliminate-def*
by *sepref*

lemmas [*llvm-code*] = *is-NONE-impl-def is-subsumed-impl-def is-strengthened-impl-def*
STRENGTHENED-BY-impl-def SUBSUMED-BY-impl-def NONE-impl-def subsumed-by-impl-def
strengthened-on-lit-impl-def

lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
get-occs-list-at-impl-def[*unfolded read-all-st-code-def*]

experiment
begin
export-llvm *isa-simplify-clauses-with-unit-st2-code*
isa-simplify-clauses-with-units-st-wl2-code
isa-deduplicate-binary-clauses-code
isa-forward-subsumption-all-impl
end

end
theory *IsaSAT-Restart-Heuristics-Defs*
imports
IsaSAT-Restart-Reduce-Defs IsaSAT-Restart-Inprocessing-Defs

begin

For simplification in our proofs, our inprocessing contains both inprocessing (currently: deduplication of binary clauses) and removal of unit clauses. We leave the concrete schedule to the inprocessing function.

definition *should-inprocess-or-unit-reduce-st* :: ‹*isasat* ⇒ *bool* ⇒ *bool*› **where**
‹*should-inprocess-or-unit-reduce-st S should-GC* ⟷
(*should-GC* ∧ *units-since-last-GC-st S* > *0*) ∨
should-inprocess-st S ∨
GC-units-required S›

definition *restart-required-heur* :: ‹*isasat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *8 word nres*› **where**
‹*restart-required-heur S last-GC last-Restart n* = *do* {
ASSERT(*learned-clss-count S* ≥ *last-Restart*);
ASSERT(*learned-clss-count S* ≥ *last-GC*);
let opt-red = *opts-reduction-st S*;
let opt-res = *opts-restart-st S*;
let curr-phase = *get-restart-phase S*;
let can-res = (*learned-clss-count S* > *last-Restart*);
let can-GC = (*learned-clss-count S* − *last-GC*> *n*);
let fully-proped = *is-fully-propagated-heur-st S*;
let should-reduce = (*opt-red* ∧ *upper-restart-bound-reached S* ∧ *can-GC*);
should-GC ← *GC-required-heur S n*;
let should-inprocess = *should-inprocess-or-unit-reduce-st S should-GC*;

if (¬*can-res* ∧ ¬*can-GC*) ∨ ¬*opt-res* ∨ ¬*opt-red* ∨ ¬*fully-proped then RETURN FLAG-no-restart*
else if curr-phase = *STABLE-MODE*
then do {
if should-reduce
then if should-inprocess
then RETURN FLAG-Inprocess-restart
else if should-GC then RETURN FLAG-GC-restart else RETURN FLAG-Reduce-restart
else if heuristic-reluctant-triggered2-st S ∧ *can-res*
then RETURN FLAG-restart
else RETURN FLAG-no-restart
}
else do {
let sema = *ema-get-value* (*get-slow-ema-heur S*);
let limit = (*opts-restart-coeff1-st S*) ∗ (*shiftr* (*sema*) *4*);
let fema = *ema-get-value* (*get-fast-ema-heur S*);
let ccount = *get-conflict-count-since-last-restart-heur S*;
let min-reached = (*ccount* > *opts-minimum-between-restart-st S*);
let level = *count-decided-st-heur S*;
let should-restart = ((*opt-res*) ∧
limit > *fema* ∧ *min-reached* ∧ *can-res* ∧
level ≥ *1* *This comment from Marijn Heule seems not to help: term /\ level /= 0 ⟹ restart decision /\ this was taken from Lingeling IIRC term /\ of not level /= (shift fema 32)*);
if should-reduce
then if should-inprocess
then RETURN FLAG-Inprocess-restart
else if should-GC
then RETURN FLAG-GC-restart
else RETURN FLAG-Reduce-restart
else if should-restart

```
    then RETURN FLAG-restart
    else RETURN FLAG-no-restart
   }
  }›
```

definition *cdcl-twl-full-restart-wl-D-GC-heur-prog* **where**

```
‹cdcl-twl-full-restart-wl-D-GC-heur-prog S0 = do {
  - ← RETURN (IsaSAT-Profile.start-GC);
  S ← do {
    if count-decided-st-heur S0 > 0
    then do {
      S ← find-decomp-wl-st-int 0 S0;
      empty-Q (empty-US-heur S)
    } else RETURN (empty-US-heur S0)
  };
  ASSERT(length (get-clauses-wl-heur S) = length (get-clauses-wl-heur S0));
  ASSERT(learned-clss-count S ≤ learned-clss-count S0);
  T ← remove-one-annot-true-clause-imp-wl-D-heur S;
  ASSERT(length (get-clauses-wl-heur T) = length (get-clauses-wl-heur S0));
  ASSERT(learned-clss-count T ≤ learned-clss-count S0);
  U ← mark-to-delete-clauses-GC-wl-D-heur T;
  ASSERT(length (get-clauses-wl-heur U) = length (get-clauses-wl-heur S0));
  ASSERT(learned-clss-count U ≤ learned-clss-count S0);
  V ← isasat-GC-clauses-wl-D False U;
  - ← RETURN (IsaSAT-Profile.stop-GC);
  RETURN (clss-size-resetUS0-st V)
 }›
```

definition *cdcl-twl-full-restart-wl-D-inprocess-heur-prog* **where**

```
‹cdcl-twl-full-restart-wl-D-inprocess-heur-prog S0 = do {
  - ← RETURN (IsaSAT-Profile.start-reduce);
  S ← do {
    if count-decided-st-heur S0 > 0
    then do {
      S ← find-decomp-wl-st-int 0 S0;
      empty-Q (empty-US-heur S)
    } else RETURN (empty-US-heur S0)
  };
  ASSERT(length (get-clauses-wl-heur S) = length (get-clauses-wl-heur S0));
  ASSERT(learned-clss-count S ≤ learned-clss-count S0);
  T ← remove-one-annot-true-clause-imp-wl-D-heur S;
  - ← RETURN (IsaSAT-Profile.stop-reduce);
  ASSERT(length (get-clauses-wl-heur T) = length (get-clauses-wl-heur S0));
     ASSERT(learned-clss-count T ≤ learned-clss-count S0);
  - ← RETURN (IsaSAT-Profile.start-binary-simp);
  T ← isa-mark-duplicated-binary-clauses-as-garbage-wl2 T;
  - ← RETURN (IsaSAT-Profile.stop-binary-simp);
  ASSERT(length (get-clauses-wl-heur T) = length (get-clauses-wl-heur S0));
     ASSERT(learned-clss-count T ≤ learned-clss-count S0);
  - ← RETURN (IsaSAT-Profile.start-subsumption);
  T ← isa-forward-subsume T;
```

```
  - ← RETURN (IsaSAT-Profile.stop-subsumption);
  ASSERT(length (get-clauses-wl-heur T) = length (get-clauses-wl-heur S0));
     ASSERT(learned-clss-count T ≤ learned-clss-count S0);
  - ← RETURN (IsaSAT-Profile.start-pure-literal);
  T ← isa-pure-literal-eliminate T;
  - ← RETURN (IsaSAT-Profile.stop-pure-literal);
  ASSERT(length (get-clauses-wl-heur T) = length (get-clauses-wl-heur S0));
  ASSERT(learned-clss-count T ≤ learned-clss-count S0);
  - ← RETURN (IsaSAT-Profile.start-reduce);
  T ← isa-simplify-clauses-with-units-st-wl2 T;
  - ← RETURN (IsaSAT-Profile.stop-reduce);
  ASSERT(length (get-clauses-wl-heur T) = length (get-clauses-wl-heur S0));
  ASSERT(learned-clss-count T ≤ learned-clss-count S0);
  if ¬get-conflict-wl-is-None-heur T then RETURN T
  else do {
    - ← RETURN (IsaSAT-Profile.start-GC);
    U ← mark-to-delete-clauses-GC-wl-D-heur T;
    ASSERT(length (get-clauses-wl-heur U) = length (get-clauses-wl-heur S0));
    ASSERT(learned-clss-count U ≤ learned-clss-count S0);
    V ← isasat-GC-clauses-wl-D True U;
    - ← RETURN (IsaSAT-Profile.stop-GC);
    RETURN (clss-size-resetUS0-st V)
  }
}›

definition restart-prog-wl-D-heur
  :: ‹isasat ⇒ nat ⇒ nat ⇒ nat ⇒ bool ⇒ (isasat × nat × nat × nat) nres›
where
  ‹restart-prog-wl-D-heur S last-GC last-Restart n brk = do {
   if brk then RETURN (S, last-GC, last-Restart, n)
   else do {
     b ← restart-required-heur S last-GC last-Restart n;
     if b = FLAG-restart
     then do {
       T ← cdcl-twl-restart-wl-heur S;
       ASSERT(learned-clss-count T ≤ learned-clss-count S);
       RETURN (T, last-GC, learned-clss-count T, n)
     }
     else if b ≠ FLAG-no-restart
     then if b ≠ FLAG-Inprocess-restart then do {
       if b = FLAG-Reduce-restart
       then do {
         T ← cdcl-twl-mark-clauses-to-delete S;
         ASSERT(learned-clss-count T ≤ learned-clss-count S);
         RETURN (T, learned-clss-count T, learned-clss-count T, n+1)
       }
       else do {
         T ← cdcl-twl-full-restart-wl-D-GC-heur-prog S;
         ASSERT(learned-clss-count T ≤ learned-clss-count S);
         RETURN (T, learned-clss-count T, learned-clss-count T, n+1)
       }
    } else do {
       T ← cdcl-twl-full-restart-wl-D-inprocess-heur-prog S;
       ASSERT(learned-clss-count T ≤ learned-clss-count S);
       RETURN (T, learned-clss-count T, learned-clss-count T, n+1)
```

```
    }
     else RETURN (S, last-GC, last-Restart, n)
   }
  }›

end
theory IsaSAT-Restart-Heuristics
imports
  IsaSAT-Restart-Heuristics-Defs
  IsaSAT-Restart-Reduce IsaSAT-Restart-Inprocessing
begin


lemma cdcl-twl-full-restart-wl-D-GC-heur-prog-alt-def:
  ‹cdcl-twl-full-restart-wl-D-GC-heur-prog S0 = do {
    S ← do {
    if count-decided-st-heur S0 > 0
    then do {
    S ← find-decomp-wl-st-int 0 S0;
    empty-Q (empty-US-heur S)
    } else RETURN (empty-US-heur S0)
    };
    ASSERT(length (get-clauses-wl-heur S) = length (get-clauses-wl-heur S0));
    ASSERT(learned-clss-count S ≤ learned-clss-count S0);
    T ← remove-one-annot-true-clause-imp-wl-D-heur S;
    ASSERT(length (get-clauses-wl-heur T) = length (get-clauses-wl-heur S0));
    ASSERT(learned-clss-count T ≤ learned-clss-count S0);
    U ← mark-to-delete-clauses-GC-wl-D-heur T;
    ASSERT(length (get-clauses-wl-heur U) = length (get-clauses-wl-heur S0));
    ASSERT(learned-clss-count U ≤ learned-clss-count S0);
    V ← isasat-GC-clauses-wl-D False U;
    RETURN (clss-size-resetUS0-st V)
  }›
  unfolding cdcl-twl-full-restart-wl-D-GC-heur-prog-def IsaSAT-Profile.start-def
    IsaSAT-Profile.stop-def by auto

lemma twl-st-heur-twl-st-heur-loopD:
  ‹(S, T) ∈ twl-st-heur ⟹ (S, T) ∈ twl-st-heur-loop› and
  twl-st-heur-loop-twl-st-heurD:
  ‹(S, T) ∈ twl-st-heur-loop ⟹ get-conflict-wl T = None ⟹ (S, T) ∈ twl-st-heur›
  by (auto simp: twl-st-heur-loop-def twl-st-heur-def)

lemma
    cdcl-twl-full-restart-wl-GC-prog-pre-heur:
      ‹cdcl-twl-full-restart-wl-GC-prog-pre T ⟹
        (S, T) ∈ twl-st-heur''' r ⟹ (S, T) ∈ twl-st-heur-restart-ana r› (is ‹- ⟹ ?Apre ⟹ ?A›) and
     cdcl-twl-full-restart-wl-D-GC-prog-post-heur:
       ‹cdcl-twl-full-restart-wl-GC-prog-post S0 T ⟹
       (S, T) ∈ twl-st-heur-restart ⟹ (clss-size-resetUS0-st S, T) ∈ twl-st-heur› (is ‹- ⟹ -?Bpre ⟹
?B›) and
     cdcl-twl-full-restart-wl-D-GC-prog-post-confl-heur:
       ‹cdcl-twl-full-restart-wl-GC-prog-post-confl S0 T ⟹
      (S, T) ∈ twl-st-heur-restart ⟹ get-conflict-wl T ≠ None ⟹
    (S, T) ∈ twl-st-heur-loop› (is ‹- ⟹ ?Cpre ⟹ ?Cconfl ⟹ ?C›)
proof −
  note cong = trail-pol-cong heuristic-rel-cong
```

option-lookup-clause-rel-cong D_0*-cong isa-vmtf-cong phase-saving-cong*
cach-refinement-empty-cong vdom-m-cong isasat-input-nempty-cong
isasat-input-bounded-cong empty-occs-list-cong

show ‹*cdcl-twl-full-restart-wl-GC-prog-pre T* ⟹ *?Apre* ⟹ *?A*›
supply [[*goals-limit=1*]]
unfolding *cdcl-twl-full-restart-wl-GC-prog-pre-def cdcl-twl-full-restart-l-GC-prog-pre-def*
apply *normalize-goal+*
subgoal for *U V*
using *literals-are-*$\mathcal{L}_{in}$*'-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*of T U V*]
cong[*of* ‹*all-atms-st T*› ‹*all-init-atms-st T*›]
vdom-m-cong[*of* ‹*all-atms-st T*› ‹*all-init-atms-st T*› ‹*get-watched-wl T*› ‹*get-clauses-wl T*›]
apply −
apply (*simp all del*: *isasat-input-nempty-def isasat-input-bounded-def*)
apply (*cases S*; *cases T*)
by (*auto simp add*: *twl-st-heur-def twl-st-heur-restart-ana-def all-atms-st-def*
clss-size-corr-restart-def clss-size-corr-def
twl-st-heur-restart-def all-init-atms-st-def simp del: *isasat-input-nempty-def*)
done
show ‹*cdcl-twl-full-restart-wl-GC-prog-post S0 T* ⟹ *?Bpre* ⟹ *?B*›
supply [[*goals-limit=1*]]
unfolding *cdcl-twl-full-restart-wl-GC-prog-post-def*
cdcl-twl-full-restart-wl-GC-prog-post-def
cdcl-twl-full-restart-l-GC-prog-pre-def
apply *normalize-goal+*
subgoal for *S0′ T′ S0″*
apply (*rule rtranclp-cdcl-twl-restart-l-inp-cdcl-twl-restart-inp*[*of S0′ T′ S0″*], *assumption+*)
apply (*frule rtranclp-cdcl-twl-inp-twl-struct-invs*, *assumption+*)
subgoal for *V*
using *literals-are-*$\mathcal{L}_{in}$*'-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*of T T′*]
cong[*of* ‹*all-init-atms-st T*› ‹*all-atms-st T*›]
vdom-m-cong[*of* ‹*all-init-atms-st T*› ‹*all-atms-st T*› ‹*get-watched-wl T*› ‹*get-clauses-wl T*›]
cdcl-twl-restart-l-invs[*of S0′ S0″ T′*]
apply −
apply (*cases S*; *cases T*)
by (*auto simp add*: *twl-st-heur-def twl-st-heur-restart-def all-atms-st-def all-init-atms-st-def*
clss-size-resetUS0-st-def
simp del: *isasat-input-nempty-def intro*: *clss-size-corr-restart-clss-size-corr*)
done
done
show ‹*cdcl-twl-full-restart-wl-GC-prog-post-confl S0 T* ⟹ *?Cpre* ⟹ *?Cconfl* ⟹ *?C*›
supply [[*goals-limit=1*]]
unfolding *cdcl-twl-full-restart-wl-GC-prog-post-confl-def*
cdcl-twl-full-restart-wl-GC-prog-post-def
cdcl-twl-full-restart-l-GC-prog-pre-def
apply *normalize-goal+*
subgoal for *S0′ T′ S0″*
apply (*rule rtranclp-cdcl-twl-restart-l-inp-cdcl-twl-restart-inp*[*of S0′ T′ S0″*], *assumption+*)
apply (*frule rtranclp-cdcl-twl-inp-twl-struct-invs*, *assumption+*)
subgoal for *V*
using *literals-are-*$\mathcal{L}_{in}$*'-literals-are-*$\mathcal{L}_{in}$*-iff*(*3*)[*of T T′*]
cong[*of* ‹*all-init-atms-st T*› ‹*all-atms-st T*›]
vdom-m-cong[*of* ‹*all-init-atms-st T*› ‹*all-atms-st T*› ‹*get-watched-wl T*› ‹*get-clauses-wl T*›]
cdcl-twl-restart-l-invs[*of S0′ S0″ T′*]
apply −
apply (*cases S*; *cases T*)

by (*clarsimp simp add*: *twl-st-heur-loop-def twl-st-heur-restart-def all-atms-st-def all-init-atms-st-def*
ac-simps
simp del: *isasat-input-nempty-def*)
done
done
qed

lemma *cdcl-twl-full-restart-wl-D-GC-heur-prog*:
‹(*cdcl-twl-full-restart-wl-D-GC-heur-prog*, *cdcl-twl-full-restart-wl-GC-prog*) ∈
twl-st-heur''''u r u →$_f$ ⟨*twl-st-heur''''uu r u*⟩*nres-rel*›
proof −
have *H*: ‹(*S*, *S'*) ∈ *twl-st-heur-restart-ana r* ⟹
(*S*, *S'*) ∈ *twl-st-heur-restart-ana' r* (*learned-clss-count S*)› **for** *S S'*
by *auto*
have *H2*: ‹(*x*, *y*) ∈ *IsaSAT-Setup.twl-st-heur''''u r u* ⟹
cdcl-twl-full-restart-wl-GC-prog-pre y ⟹
(*x*, *y*) ∈ *twl-st-heur-restart-ana' r* (*learned-clss-count x*)› **for** *x y*
using *cdcl-twl-full-restart-wl-GC-prog-pre-heur*[*of y x r*]
by *auto*
have *UUa*: ‹(*U*, *Ua*) ∈ *twl-st-heur-restart-ana' r u* ⟹
(*U*, *Ua*) ∈ *twl-st-heur-restart'''u r u*› **for** *U Ua r u*
by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*)
show *?thesis*
unfolding *cdcl-twl-full-restart-wl-D-GC-heur-prog-alt-def*
cdcl-twl-full-restart-wl-GC-prog-def
apply (*intro frefI nres-relI*)
apply (*refine-rcg cdcl-twl-local-restart-wl-spec0*
remove-one-annot-true-clause-imp-wl-D-heur-remove-one-annot-true-clause-imp-wl-D[**where** *r*=*r*,
THEN fref-to-Down]
mark-to-delete-clauses-wl-D-heur-mark-to-delete-clauses-GC-wl-D[**where** *r*=*r*, *THEN fref-to-Down*]
isasat-GC-clauses-wl-D[**where** *r*=*r*, *THEN fref-to-Down*])
apply (*rule H2*; *assumption*)
subgoal
unfolding *cdcl-twl-full-restart-wl-GC-prog-pre-def*
cdcl-twl-full-restart-l-GC-prog-pre-def
by *normalize-goal+ auto*
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
apply (*assumption*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
apply (*assumption*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
apply (*rule UUa*; *assumption*)
subgoal for *x y S S' T T' U U' V V'*
using *learned-clss-count-clss-size-resetUS0-st-le*[*of V*]
unfolding *mem-Collect-eq prod.case*
apply (*intro conjI cdcl-twl-full-restart-wl-D-GC-prog-post-heur*)
apply *assumption+*
by *auto*
done
qed

lemma *cdcl-twl-full-restart-wl-D-inprocess-heur-prog-alt-def*:
‹*cdcl-twl-full-restart-wl-D-inprocess-heur-prog S0* = *do* {

```
    S ← do {
      if count-decided-st-heur S0 > 0
      then do {
        S ← find-decomp-wl-st-int 0 S0;
        empty-Q (empty-US-heur S)
      } else RETURN (empty-US-heur S0)
    };
    ASSERT(length (get-clauses-wl-heur S) = length (get-clauses-wl-heur S0));
    ASSERT(learned-clss-count S ≤ learned-clss-count S0);
    T ← remove-one-annot-true-clause-imp-wl-D-heur S;
    ASSERT(length (get-clauses-wl-heur T) = length (get-clauses-wl-heur S0));
        ASSERT(learned-clss-count T ≤ learned-clss-count S0);
    T ← isa-mark-duplicated-binary-clauses-as-garbage-wl2 T;
    ASSERT(length (get-clauses-wl-heur T) = length (get-clauses-wl-heur S0));
        ASSERT(learned-clss-count T ≤ learned-clss-count S0);
    T ← isa-forward-subsume T;
    ASSERT(length (get-clauses-wl-heur T) = length (get-clauses-wl-heur S0));
        ASSERT(learned-clss-count T ≤ learned-clss-count S0);
    T ← isa-pure-literal-eliminate T;
    ASSERT(length (get-clauses-wl-heur T) = length (get-clauses-wl-heur S0));
    ASSERT(learned-clss-count T ≤ learned-clss-count S0);
    T ← isa-simplify-clauses-with-units-st-wl2 T;
    ASSERT(length (get-clauses-wl-heur T) = length (get-clauses-wl-heur S0));
    ASSERT(learned-clss-count T ≤ learned-clss-count S0);
    if ¬get-conflict-wl-is-None-heur T then RETURN T
    else do {
      U ← mark-to-delete-clauses-GC-wl-D-heur T;
      ASSERT(length (get-clauses-wl-heur U) = length (get-clauses-wl-heur S0));
      ASSERT(learned-clss-count U ≤ learned-clss-count S0);
      V ← isasat-GC-clauses-wl-D True U;
      RETURN (clss-size-resetUS0-st V)
    }
  }›
  unfolding cdcl-twl-full-restart-wl-D-inprocess-heur-prog-def IsaSAT-Profile.start-def
    IsaSAT-Profile.stop-def by (auto intro!: bind-cong[OF refl])
```

We need the plus one if we derive the empty conflict...

TODO: we don't care about that case and can live with an overflow!

```
abbreviation twl-st-heur''''u'
   :: ‹nat ⇒ nat ⇒ (isasat × nat twl-st-wl) set›
where
‹twl-st-heur''''u' r u ≡ {(S, T). (S, T) ∈ twl-st-heur ∧
          length (get-clauses-wl-heur S) = r ∧
          (get-conflict-wl T = None ⟶ learned-clss-count S ≤ u) ∧
          (get-conflict-wl T ≠ None ⟶ learned-clss-count S ≤ u+1)}›


lemma isa-simplify-clauses-with-unit-st2-isa-simplify-clauses-with-unit-wl:
  assumes ‹(S,S') ∈ twl-st-heur-restart-ana' r u›
  shows
   ‹isa-simplify-clauses-with-units-st-wl2 S ≤ ⇓ (twl-st-heur-restart-ana' r u) (simplify-clauses-with-units-st-wl
S')›
  apply (rule order-trans)
    defer
  apply (rule ref-two-step')
```

apply (*rule simplify-clauses-with-units-st-wl2-simplify-clauses-with-units-st-wl*[*unfolded Down-id-eq*, *of* - *S'*])
subgoal by *auto*
subgoal
apply (*rule isa-simplify-clauses-with-units-st2-simplify-clauses-with-units-st2*[*THEN order-trans*, *of* - *S'*])
apply (*rule assms*)
subgoal using *assms* **by** *auto*
done
done

abbreviation *twl-st-heur-loop''''uu* **where**
‹*twl-st-heur-loop''''uu r u* ≡ {(*S*, *T*). (*S*, *T*) ∈ *twl-st-heur-loop* ∧ *length* (*get-clauses-wl-heur S*) ≤ *r* ∧
learned-clss-count S ≤ *u*}›

lemma *cdcl-twl-full-restart-wl-D-inprocess-heur-prog*:
‹(*cdcl-twl-full-restart-wl-D-inprocess-heur-prog*, *cdcl-twl-full-restart-inprocess-wl-prog*) ∈
twl-st-heur''''u r u →*f* ⟨*twl-st-heur-loop''''uu r u*⟩*nres-rel*›
proof −
have *H*: ‹(*S*, *S'*) ∈ *twl-st-heur-restart-ana r* ⟹
(*S*, *S'*) ∈ *twl-st-heur-restart-ana' r* (*learned-clss-count S*)› **for** *S S'*
by *auto*
have *H2*: ‹(*x*, *y*) ∈ *IsaSAT-Setup.twl-st-heur''''u r u* ⟹
cdcl-twl-full-restart-wl-GC-prog-pre y ⟹
(*x*, *y*) ∈ *twl-st-heur-restart-ana' r* (*learned-clss-count x*)› **for** *x y*
using *cdcl-twl-full-restart-wl-GC-prog-pre-heur*[*of y x r*]
by *auto*
have *UUa*: ‹(*U*, *Ua*) ∈ *twl-st-heur-restart-ana' r u* ⟹
(*U*, *Ua*) ∈ *twl-st-heur-restart'''u r u*› **for** *U Ua r u*
by (*auto simp*: *twl-st-heur-restart-ana-def twl-st-heur-restart-def*)
show *?thesis*
unfolding *cdcl-twl-full-restart-wl-D-inprocess-heur-prog-alt-def*
cdcl-twl-full-restart-inprocess-wl-prog-def
apply (*intro frefI nres-relI*)
apply (*refine-rcg cdcl-twl-local-restart-wl-spec0*
remove-one-annot-true-clause-imp-wl-D-heur-remove-one-annot-true-clause-imp-wl-D[**where** *r*=*r*, *THEN fref-to-Down*]
mark-to-delete-clauses-wl-D-heur-mark-to-delete-clauses-GC-wl-D[**where** *r*=*r*, *THEN fref-to-Down*]
isasat-GC-clauses-wl-D[**where** *r*=*r*, *THEN fref-to-Down*]
isa-simplify-clauses-with-unit-st2-isa-simplify-clauses-with-unit-wl[**where** *r*=*r*]
isa-mark-duplicated-binary-clauses-as-garbage-wl-mark-duplicated-binary-clauses-as-garbage-wl2[**where** *r*=*r*]
isa-pure-literal-eliminate[**where** *r*=*r*]
isa-forward-subsume-forward-subsume-wl[**where** *r*=*r*])
apply (*rule H2*; *assumption*)
subgoal
unfolding *cdcl-twl-full-restart-wl-GC-prog-pre-def*
cdcl-twl-full-restart-l-GC-prog-pre-def
by *normalize-goal*+ *auto*
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
apply (*assumption*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
apply (*solves auto*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)

subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
apply (*assumption*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
apply (*assumption*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
apply (*assumption*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
subgoal
by (*subst get-conflict-wl-is-None-heur-get-conflict-wl-is-None-ana*[*THEN fref-to-Down-unRET-Id*])
(*auto simp*: *get-conflict-wl-is-None-def*)
subgoal for *x y*
unfolding *mem-Collect-eq prod.case*
apply (*subst cdcl-twl-full-restart-wl-D-GC-prog-post-confl-heur*)
apply *assumption*
by (*auto simp*: *twl-st-heur-restart-ana-def*)
apply (*assumption*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
subgoal by (*auto simp*: *twl-st-heur-restart-ana-def*)
apply (*rule UUa*; *assumption*)
subgoal for *x y S S′ T T′ U U′ V V′ W W′ X X′ Y Y′ Z Z′*
using *learned-clss-count-clss-size-resetUS0-st-le*[*of Z*]
unfolding *mem-Collect-eq prod.case*
apply (*intro conjI*)
by (*auto intro*: *twl-st-heur-twl-st-heur-loopD intro*!: *cdcl-twl-full-restart-wl-D-GC-prog-post-heur*)
done
qed

lemma *restart-required-heur-restart-required-wl0*:
‹(*uncurry3 restart-required-heur*, *uncurry3 restart-required-wl*) ∈
[λ(((*S*, -), -), -). (*get-init-clauses0-wl S* = {#} ∧ *get-learned-clauses0-wl S* = {#})]$_f$
twl-st-heur‴ r ×$_f$ *nat-rel* ×$_f$ *nat-rel* ×$_f$ *nat-rel* → ⟨*restart-flag-rel*⟩*nres-rel*›
unfolding *restart-required-heur-def restart-required-wl-def uncurry-def Let-def*
restart-flag-rel-def FLAG-GC-restart-def FLAG-restart-def FLAG-no-restart-def
GC-required-heur-def FLAG-Reduce-restart-def learned-clss-count-def
FLAG-Inprocess-restart-def
apply (*intro frefI nres-relI*)
apply *refine-rcg*
subgoal
by
(*auto simp add*: *twl-st-heur-def get-learned-clss-wl-def clss-size-lcountU0-def*
clss-size-def clss-size-lcount-def clss-size-lcountUE-def clss-size-corr-def
clss-size-lcountUS-def clss-size-lcountUEk-def get-unit-learned-clss-wl-alt-def)
subgoal
by
(*auto simp add*: *twl-st-heur-def get-learned-clss-wl-def clss-size-lcountU0-def*
clss-size-def clss-size-lcount-def clss-size-lcountUE-def clss-size-corr-def
clss-size-lcountUS-def clss-size-lcountUEk-def get-unit-learned-clss-wl-alt-def)
subgoal
by (*clarsimp split*: *if-splits simp add*: *twl-st-heur-def RETURN-RES-refine-iff*)
(*clarsimp simp add*: *twl-st-heur-def get-learned-clss-wl-def clss-size-corr-def*
clss-size-def clss-size-lcount-def clss-size-lcountUE-def RETURN-RES-refine-iff
clss-size-lcountUS-def clss-size-lcountU0-def clss-size-lcountUEk-def ac-simps
get-unit-learned-clss-wl-alt-def)

done

lemma *restart-prog-wl-D-heur-alt-def*:
‹*restart-prog-wl-D-heur S last-GC last-Restart n brk* =
(*if brk then RETURN* (*S*, *last-GC*, *last-Restart*, *n*)
else do {
b ← *restart-required-heur S last-GC last-Restart n*;
if b = *FLAG-restart*
then do {
T ← *cdcl-twl-restart-wl-heur S*;
ASSERT(*learned-clss-count T* ≤ *learned-clss-count S*);
RETURN (*T*, *last-GC*, *learned-clss-count T*, *n*)
}
else if b ≠ *FLAG-no-restart*
then if b ≠ *FLAG-Inprocess-restart then do* {
let b = *b*;
T ← (*if b* = *FLAG-Reduce-restart*
then cdcl-twl-mark-clauses-to-delete S
else cdcl-twl-full-restart-wl-D-GC-heur-prog S);
ASSERT(*learned-clss-count T* ≤ *learned-clss-count S*);
RETURN (*T*, *learned-clss-count T*, *learned-clss-count T*, *n*+*1*)
}
else do {
T ← *cdcl-twl-full-restart-wl-D-inprocess-heur-prog S*;
ASSERT(*learned-clss-count T* ≤ *learned-clss-count S*);
RETURN (*T*, *learned-clss-count T*, *learned-clss-count T*, *n*+*1*)
}
else RETURN (*S*, *last-GC*, *last-Restart*, *n*)
})›
unfolding *restart-prog-wl-D-heur-def Let-def*
by (*auto intro*: *bind-cong*[*OF refl*])

lemma *cdcl-twl-mark-clauses-to-delete-cdcl-twl-full-restart-wl-prog-D2*:
‹(*cdcl-twl-mark-clauses-to-delete*, *cdcl-twl-full-restart-wl-prog*) ∈
twl-st-heur''''u r u →$_f$ ⟨*twl-st-heur''''uu r u*⟩*nres-rel*›
apply (*intro frefI nres-relI*)
apply (*rule order-trans*[*OF cdcl-twl-mark-clauses-to-delete-cdcl-twl-full-restart-wl-prog-D*[*THEN fref-to-Down*]])
apply *fast*
apply *assumption*
apply (*rule conc-fun-R-mono*)
by *auto*

lemma *restart-prog-wl-alt-def2*:
‹*restart-prog-wl S last-GC last-Restart n brk* = *do*{
ASSERT(*restart-abs-wl-pre S last-GC last-Restart brk*);
ASSERT (*last-GC*
≤ *size* (*get-learned-clss-wl S*) + *size* (*get-unit-learned-clss-wl S*) +
size (*get-subsumed-learned-clauses-wl S*) +
size (*get-learned-clauses0-wl S*));
ASSERT (*last-Restart*
≤ *size* (*get-learned-clss-wl S*) + *size* (*get-unit-learned-clss-wl S*) +
size (*get-subsumed-learned-clauses-wl S*) +
size (*get-learned-clauses0-wl S*));
if brk then RETURN (*S*, *last-GC*, *last-Restart*, *n*)
else do {

```
    b ← restart-required-wl S last-GC last-Restart n;
    if b = RESTART ∧ ¬brk then do {
      T ← cdcl-twl-restart-wl-prog S;
      RETURN (T, last-GC, size (get-all-learned-clss-wl T), n)
    }
    else if (b = GC ∨ b = INPROCESS) ∧ ¬brk then
      if b ≠ INPROCESS then do {
        b ← SPEC(λ-. True);
        T ← (if b then cdcl-twl-full-restart-wl-prog S else cdcl-twl-full-restart-wl-GC-prog S);
        RETURN (T, size (get-all-learned-clss-wl T), size (get-all-learned-clss-wl T), n + 1)
      } else do {
        T ← cdcl-twl-full-restart-inprocess-wl-prog S;
        RETURN (T, size (get-all-learned-clss-wl T), size (get-all-learned-clss-wl T), n + 1)
      }
    else
      RETURN (S, last-GC, last-Restart, n)
  }}› (is ‹?A = ?B›)
proof −
  have ‹?A ≤ ⇓ Id ?B›
    unfolding restart-prog-wl-def restart-required-wl-def
    by refine-vcg
      (auto simp: restart-required-wl-def RES-RETURN-RES intro!: bind-cong[OF refl])
  moreover have ‹?B ≤ ⇓ Id ?A›
    unfolding restart-prog-wl-def restart-required-wl-def nres-monad3
    by refine-vcg
      (auto simp: restart-required-wl-def RES-RETURN-RES intro!: bind-cong[OF refl])
  ultimately show ?thesis by simp
qed

lemma restart-abs-wl-pre-emptyN0S:
  assumes ‹restart-abs-wl-pre S lastGC lastRestart C› and [simp]: ‹¬C›
  shows ‹get-init-clauses0-wl S = {#} ∧ get-learned-clauses0-wl S = {#}›
proof −
  obtain x xa where
    Sx: ‹(S, x) ∈ state-wl-l None› and
    ‹correct-watching S› and
    ‹blits-in-ℒ_in S› and
    xxa: ‹(x, xa) ∈ twl-st-l None› and
    struct: ‹twl-struct-invs xa› and
    ‹twl-list-invs x› and
    ‹clauses-to-update-l x = {#}› and
    ‹twl-stgy-invs xa› and
    ‹¬ C ⟶ get-conflict xa = None› and
    ‹lastRestart ≤ size (get-all-learned-clss xa)› and
    ‹lastGC ≤ size (get-all-learned-clss xa)›
    using assms unfolding restart-abs-wl-pre-def restart-abs-l-pre-def restart-prog-pre-def apply −
    apply normalize-goal+
    by fast

  then have ‹get-conflict xa = None›
    by simp
  moreover have ‹clauses0-inv (pstate_W-of xa)›
    using struct unfolding twl-struct-invs-def pcdcl-all-struct-invs-def by fast
  ultimately have ‹get-init-clauses0 xa = {#} ∧ get-learned-clauses0 xa = {#}›
    unfolding clauses0-inv-def
    by (cases xa; auto simp: clauses0-inv-def)
```

```
  then show ?thesis
    using Sx xxa by auto
qed

abbreviation restart-prog-wl-heur-rel :: ‹-› where
  ‹restart-prog-wl-heur-rel r u≡ {((T, a, b, c), (U, d, e, f)).
  (T,U) ∈ twl-st-heur-loop''''uu r u ∧
  (get-conflict-wl U = None ⟶((a,b,c), (d,e,f)) ∈ nat-rel ×r nat-rel ×r nat-rel)}›

abbreviation restart-prog-wl-heur-rel2 :: ‹-› where
  ‹restart-prog-wl-heur-rel2≡ {((T, a, b, c), (U, d, e, f)).
  (T,U) ∈ twl-st-heur-loop ∧
  (get-conflict-wl U = None ⟶((a,b,c), (d,e,f)) ∈ nat-rel ×r nat-rel ×r nat-rel)}›

lemma restart-prog-wl-D-heur-restart-prog-wl-D:
  ‹(uncurry4 restart-prog-wl-D-heur, uncurry4 restart-prog-wl) ∈
  twl-st-heur''''u r u ×f nat-rel ×f nat-rel ×f nat-rel ×f bool-rel →f
  ⟨restart-prog-wl-heur-rel r u⟩nres-rel›
proof −
  have [refine0]: ‹RETURN b ≤ ⇓{(c, c'). c' ⟷ (c = FLAG-Reduce-restart)} (SPEC (λ-::bool. True))›
for b
    by (auto simp: GC-required-heur-def RETURN-RES-refine-iff)
  have H: ‹(x1g, x1c)
       ∈ twl-st-heur''''u r (learned-clss-count x1g)›
    if
      ‹(x, y)
       ∈ twl-st-heur''''u r u ×f nat-rel ×f nat-rel ×f nat-rel ×f bool-rel› and
      ‹x1b = (x1c, x2)› and
      ‹x1a = (x1b, x2a)› and
      ‹x1 = (x1a, x2b)› and
      ‹y = (x1, x2c)› and
      ‹x1f = (x1g, x2d)› and
      ‹x1e = (x1f, x2e)› and
      ‹x1d = (x1e, x2f)› and
      ‹x = (x1d, x2g)›
    for x y x1 x1a x1b x1c x2 x2a x2b x2c x1d x1e x1f x1g x2d x2e x2f x2g b ba bb
    using that by auto

  show ?thesis
    supply RETURN-as-SPEC-refine[refine2 del] learned-clss-count-twl-st-heur[simp]
    unfolding restart-prog-wl-D-heur-alt-def restart-prog-wl-alt-def2 uncurry-def
    apply (intro frefI nres-relI)
    subgoal for x y
    apply (refine-rcg
      restart-required-heur-restart-required-wl0[where r=r, THEN fref-to-Down-curry3]
        cdcl-twl-restart-wl-heur-cdcl-twl-restart-wl-D-prog[where r=r, THEN fref-to-Down]
        cdcl-twl-full-restart-wl-D-GC-heur-prog[where r=r, THEN fref-to-Down, THEN order-trans]
      cdcl-twl-mark-clauses-to-delete-cdcl-twl-full-restart-wl-prog-D2[where r=r and
      u = ‹learned-clss-count (fst (fst (fst (fst x))))›, THEN fref-to-Down]
      cdcl-twl-full-restart-wl-D-inprocess-heur-prog[where r=r and
      u = ‹learned-clss-count (fst (fst (fst (fst x))))›, THEN fref-to-Down])
    subgoal by auto
    subgoal by (auto intro!: twl-st-heur-twl-st-heur-loopD)
    subgoal by (auto dest: restart-abs-wl-pre-emptyN0S)
    subgoal by (auto dest: restart-abs-wl-pre-emptyN0S)
    subgoal by auto
```

```
    subgoal by (auto simp: restart-flag-rel-def FLAG-GC-restart-def FLAG-restart-def
      FLAG-no-restart-def FLAG-Reduce-restart-def FLAG-Inprocess-restart-def)
    apply (rule twl-st-heur'''-twl-st-heur''''uD)
    subgoal by auto
    subgoal by auto
    subgoal by (auto intro!: twl-st-heur-twl-st-heur-loopD)
    subgoal by (auto simp: restart-flag-rel-def FLAG-GC-restart-def FLAG-restart-def
      FLAG-no-restart-def FLAG-Reduce-restart-def FLAG-Inprocess-restart-def)
    subgoal by (auto simp: restart-flag-rel-def FLAG-GC-restart-def FLAG-restart-def
      FLAG-no-restart-def FLAG-Reduce-restart-def FLAG-Inprocess-restart-def)
    subgoal by (auto simp: restart-flag-rel-def FLAG-GC-restart-def FLAG-restart-def
      FLAG-no-restart-def FLAG-Reduce-restart-def FLAG-Inprocess-restart-def)
    subgoal by auto
    apply (rule H; assumption)
    subgoal for x1 x1a x1b x1c x2 x2a x2b x2c x1d x1e x1f x1g x2d x2e x2f x2g b ba bb
      by (rule conc-fun-R-mono) auto
    subgoal
      by auto
    subgoal
      by (auto dest: restart-abs-wl-pre-emptyN0S intro!: twl-st-heur-twl-st-heur-loopD)
    subgoal
      by auto
    subgoal
      by auto
    subgoal
      by (auto dest: restart-abs-wl-pre-emptyN0S dest!: twl-st-heur-loop-twl-st-heurD)
    subgoal
      by (auto intro!: twl-st-heur-twl-st-heur-loopD)
    done
  done
 qed

lemma restart-prog-wl-D-heur-restart-prog-wl-D2:
  ‹(uncurry4 restart-prog-wl-D-heur, uncurry4 restart-prog-wl) ∈
  twl-st-heur ×f nat-rel ×f nat-rel ×f nat-rel ×f bool-rel →f ⟨restart-prog-wl-heur-rel2⟩nres-rel›
  apply (intro frefI nres-relI)
  apply (rule-tac r3 = ‹length(get-clauses-wl-heur (fst (fst (fst (fst x)))))› and
      u4 = ‹learned-clss-count (fst (fst (fst (fst x))))› in
    order-trans[OF restart-prog-wl-D-heur-restart-prog-wl-D[THEN fref-to-Down]])
  apply fast
  apply (auto intro!: conc-fun-R-mono)
  done

end
theory IsaSAT-Restart-Heuristics-LLVM
  imports IsaSAT-Restart-Heuristics-Defs IsaSAT-Setup-LLVM
    IsaSAT-VMTF-State-LLVM IsaSAT-Rephase-State-LLVM
    IsaSAT-Arena-Sorting-LLVM
    IsaSAT-Restart-Reduce-LLVM
    IsaSAT-Inprocessing-LLVM
    IsaSAT-Proofs-LLVM
begin

hide-fact (open) Sepref-Rules.frefI
```

lemma *trail-set-zeroed-until-state-alt-def*:
  ‹*RETURN oo trail-set-zeroed-until-state* = (λ*k S*. *do* {
    let (*M*, *S*) = *extract-trail-wl-heur S*;
    let M = *trail-set-zeroed-until k M*;
    RETURN (*update-trail-wl-heur M S*)
  })›
  unfolding *trail-set-zeroed-until-state-def*
  by (*auto simp*: *state-extractors*
    intro!: *ext split*: *isasat-int-splits*)

sepref-def *trail-set-zeroed-until-state*
  is ‹*uncurry* (*RETURN oo trail-set-zeroed-until-state*)›
  :: ‹*sint64-nat-assn*^{k} *$*_a$ *isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
  unfolding *trail-set-zeroed-until-state-alt-def*
  by *sepref*

lemma *trail-zeroed-until-state-alt-def*:
  ‹*RETURN o trail-zeroed-until-state* = *read-trail-wl-heur* (*RETURN* ∘ *trail-zeroed-until*)›
  by (*auto intro*!: *ext simp*: *trail-zeroed-until-state-def trail-zeroed-until-def*
    read-all-st-def split: *isasat-int-splits*)

definition *trail-zeroed-until-state-impl* **where**
  ‹*trail-zeroed-until-state-impl* = *read-trail-wl-heur-code count-decided-pol-impl*›

sepref-register *extract-trail-wl-heur count-decided-pol trail-zeroed-until-state trail-set-zeroed-until-state*

definition *trail-zeroed-until-state-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
  ‹*trail-zeroed-until-state-fast-code* = *read-trail-wl-heur-code trail-zeroed-until-impl*›

global-interpretation *trail-zeroed-until*: *read-trail-param-adder0* **where**
  f = ‹*trail-zeroed-until-impl*› **and**
  f' = ‹*RETURN o trail-zeroed-until*› **and**
  x-assn = *sint64-nat-assn* **and**
  P = ‹(λ*S*. *True*)›
  rewrites ‹*read-trail-wl-heur* (*RETURN o trail-zeroed-until*) = *RETURN o trail-zeroed-until-state*›
and
  ‹*read-trail-wl-heur-code trail-zeroed-until-impl* = *trail-zeroed-until-state-fast-code*›
  apply *unfold-locales*
  apply (*rule trail-zeroed-until-impl.refine*)
  subgoal
    by (*auto simp*: *read-all-st-def trail-zeroed-until-state-def intro*!: *ext*
      split: *isasat-int-splits*)
  subgoal
    by (*auto simp*: *trail-zeroed-until-state-fast-code-def*)
  done

lemmas [*sepref-fr-rules*] = *trail-zeroed-until.refine*[*unfolded lambda-comp-true*]
lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
  trail-zeroed-until-state-fast-code-def[*unfolded read-all-st-code-def*]

sepref-def *FLAG-restart-impl*
  is ‹*uncurry0* (*RETURN FLAG-restart*)›
  :: ‹*unit-assn*^{k} →$_a$ *word-assn*›
  unfolding *FLAG-restart-def*
  by *sepref*

sepref-def *FLAG-no-restart-impl*
  is ‹*uncurry0* (*RETURN FLAG-no-restart*)›
  :: ‹*unit-assn*^{k} →$_a$ *word-assn*›
  unfolding *FLAG-no-restart-def*
  by *sepref*

sepref-def *FLAG-GC-restart-impl*
  is ‹*uncurry0* (*RETURN FLAG-GC-restart*)›
  :: ‹*unit-assn*^{k} →$_a$ *word-assn*›
  unfolding *FLAG-GC-restart-def*
  by *sepref*

sepref-def *FLAG-Reduce-restart-impl*
  is ‹*uncurry0* (*RETURN FLAG-Reduce-restart*)›
  :: ‹*unit-assn*^{k} →$_a$ *word-assn*›
  unfolding *FLAG-Reduce-restart-def*
  by *sepref*

sepref-def *FLAG-Inprocess-restart-impl*
  is ‹*uncurry0* (*RETURN FLAG-Inprocess-restart*)›
  :: ‹*unit-assn*^{k} →$_a$ *word-assn*›
  unfolding *FLAG-Inprocess-restart-def*
  by *sepref*

definition *end-of-restart-phase-st-impl* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
  ‹*end-of-restart-phase-st-impl* = *read-heur-wl-heur-code end-of-restart-phase-impl*›

global-interpretation *end-of-restart-phase*: *read-heur-param-adder0* **where**
  f' = ‹*RETURN o end-of-restart-phase*› **and**
  f = *end-of-restart-phase-impl* **and**
  x-assn = *word-assn* **and**
  P = ‹λ-. *True*›
  rewrites ‹*read-heur-wl-heur* (*RETURN o end-of-restart-phase*) = *RETURN o end-of-restart-phase-st*›
and
    ‹*read-heur-wl-heur-code end-of-restart-phase-impl* = *end-of-restart-phase-st-impl*›
  apply *unfold-locales*
  apply (*rule end-of-restart-phase-impl-refine*)
  subgoal by (*auto simp*: *read-all-st-def end-of-restart-phase-st-def intro*!: *ext*
    split: *isasat-int-splits*)
  subgoal by (*auto simp*: *end-of-restart-phase-st-impl-def*)
  done

definition *end-of-rephasing-phase-st-impl* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
  ‹*end-of-rephasing-phase-st-impl* = *read-heur-wl-heur-code end-of-rephasing-phase-heur-stats-impl*›

global-interpretation *end-of-rephasing-phase*: *read-heur-param-adder0* **where**
  f' = ‹*RETURN o end-of-rephasing-phase-heur*› **and**
  f = *end-of-rephasing-phase-heur-stats-impl* **and**
  x-assn = *word-assn* **and**

P = ‹λ-. *True*›
rewrites ‹*read-heur-wl-heur* (*RETURN o end-of-rephasing-phase-heur*) = *RETURN o end-of-rephasing-phase-st*›
and
‹*read-heur-wl-heur-code end-of-rephasing-phase-heur-stats-impl* = *end-of-rephasing-phase-st-impl*›
apply *unfold-locales*
apply (*rule heur-refine*)
subgoal by (*auto simp*: *read-all-st-def end-of-rephasing-phase-st-def intro*!: *ext split*: *isasat-int-splits*)
subgoal by (*auto simp*: *end-of-rephasing-phase-st-impl-def*)
done

lemmas [*sepref-fr-rules*] = *end-of-restart-phase.refine end-of-rephasing-phase.refine*
lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] =
end-of-restart-phase-st-impl-def[*unfolded read-all-st-code-def*]
end-of-rephasing-phase-st-impl-def[*unfolded read-all-st-code-def*]

sepref-register *incr-restart-phase incr-restart-phase-end*
update-restart-phases

lemma *update-restart-phases-alt-def*:
‹*update-restart-phases* = (λ*S*. *do* {
let lcount = *get-global-conflict-count S*;
let (*heur*, *S*) = *extract-heur-wl-heur S*;
let (*vm*, *S*) = *extract-vmtf-wl-heur S*;
let vm = *switch-bump-heur vm*;
heur ← *RETURN* (*incr-restart-phase heur*);
heur ← *RETURN* (*incr-restart-phase-end lcount heur*);
heur ← *RETURN* (*if current-restart-phase heur* = *STABLE-MODE then heuristic-reluctant-enable heur else heuristic-reluctant-disable heur*);
heur ← *RETURN* (*swap-emas heur*);
RETURN (*update-heur-wl-heur heur* (*update-vmtf-wl-heur vm S*))
})›
by (*auto simp*: *update-restart-phases-def state-extractors split*: *isasat-int-splits intro*!: *ext*)

sepref-def *update-restart-phases-impl*
is ‹*update-restart-phases*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
unfolding *update-restart-phases-alt-def*
by *sepref*

sepref-register *upper-restart-bound-reached*

sepref-def *upper-restart-bound-reached-fast-impl*
is ‹(*RETURN o upper-restart-bound-reached*)›
:: ‹*isasat-bounded-assn*^{k} →$_a$ *bool1-assn*›
unfolding *upper-restart-bound-reached-def PR-CONST-def*
fold-tuple-optimizations get-restart-count-st-def[*symmetric*]
get-global-conflict-count-def[*symmetric*]
supply [[*goals-limit* = *1*]]
by *sepref*

sepref-register *max-restart-decision-lvl*

sepref-def *minimum-number-between-restarts-impl*

is ‹*uncurry0* (*RETURN minimum-number-between-restarts*)›
:: ‹*unit-assn*^{k} →$_a$ *word-assn*›
unfolding *minimum-number-between-restarts-def*
by *sepref*

sepref-def *uint32-nat-assn-impl*
is ‹*uncurry0* (*RETURN max-restart-decision-lvl*)›
:: ‹*unit-assn*^{k} →$_a$ *uint32-nat-assn*›
unfolding *max-restart-decision-lvl-def*
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

sepref-def *GC-required-heur-fast-code*
is ‹*uncurry GC-required-heur*›
:: ‹*isasat-bounded-assn*^{k} *$*_a$ *uint64-nat-assn*^{k} →$_a$ *bool1-assn*›
supply [[*goals-limit=1*]] *of-nat-snat*[*sepref-import-param*]
unfolding *GC-required-heur-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *GC-units-required-heur-fast-code*
is ‹*RETURN o GC-units-required*›
:: ‹*isasat-bounded-assn*^{k} →$_a$ *bool1-assn*›
supply [[*goals-limit=1*]] *of-nat-snat*[*sepref-import-param*]
unfolding *GC-units-required-def*
by *sepref*

sepref-register *should-inprocess-or-unit-reduce-st*

sepref-def *should-inprocess-or-unit-reduce-st*
is ‹*uncurry* (*RETURN oo should-inprocess-or-unit-reduce-st*)›
:: ‹*isasat-bounded-assn*^{k} *$*_a$ *bool1-assn*^{k} →$_a$ *bool1-assn*›
unfolding *should-inprocess-or-unit-reduce-st-def should-inprocess-st-def*
by *sepref*

sepref-register *ema-get-value get-fast-ema-heur get-slow-ema-heur*

sepref-def *restart-required-heur-fast-code*
is ‹*uncurry3 restart-required-heur*›
:: ‹[λ(((*S*, -), -), -). *learned-clss-count S* ≤ *unat64-max*]$_a$ *isasat-bounded-assn*^{k} *$*_a$
uint64-nat-assn^k *$*_a$ *uint64-nat-assn*^{k} *$*_a$ *uint64-nat-assn*^{k} → *word-assn*›
supply [[*goals-limit=1*]] *isasat-fast-def*[*simp*] *clss-size-allcount-alt-def*[*simp*]
learned-clss-count-def[*simp*]
unfolding *restart-required-heur-def get-slow-ema-heur-st-def*[*symmetric*]
get-fast-ema-heur-st-def[*symmetric*]
apply (*rewrite* **in** ‹¤ ≤ -› *unat-const-fold*[**where** *'a=32*])

apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *replace-reason-in-trail-code*
is ‹*uncurry2 replace-reason-in-trail*›
:: ‹*unat-lit-assn*^{k} *$*_a$ (*sint64-nat-assn*)^k *$*_a$ *trail-pol-fast-assn*^{d} →$_a$ *trail-pol-fast-assn*›
supply [[*goals-limit=1*]]
unfolding *trail-pol-fast-assn-def replace-reason-in-trail-def trail-update-reason-at-def*

apply (*annot-snat-const* ‹*TYPE*(*64*)›)
apply (*rewrite at* ‹*list-update - - -*› *annot-index-of-atm*)
by *sepref*

lemma *isasat-replace-annot-in-trail-alt-def*:
‹*isasat-replace-annot-in-trail L C* = (λ*S*. *do* {
let (*lcount*, *S*) = *extract-lcount-wl-heur S*;
let (*M*, *S*) = *extract-trail-wl-heur S*;
let lcount = *clss-size-resetUS0 lcount*;
M ← *replace-reason-in-trail L C M*;
RETURN (*update-trail-wl-heur M* (*update-lcount-wl-heur lcount S*))
})›
by (*auto simp*: *isasat-replace-annot-in-trail-def state-extractors*
intro!: *ext split*: *isasat-int-splits*)
sepref-register *isasat-replace-annot-in-trail*
sepref-def *isasat-replace-annot-in-trail-code*
is ‹*uncurry2 isasat-replace-annot-in-trail*›
:: ‹*unat-lit-assn*^{k} $*_a$ (*sint64-nat-assn*)^k $*_a$ *isasat-bounded-assn*^{d} $\to_a$ *isasat-bounded-assn*›
supply [[*goals-limit*=*1*]]
unfolding *isasat-replace-annot-in-trail-alt-def*
by *sepref*

sepref-register *remove-one-annot-true-clause-one-imp-wl-D-heur*

lemma *remove-one-annot-true-clause-one-imp-wl-D-heurI*:
‹*isasat-fast b* ⟹
learned-clss-count xb ≤ *learned-clss-count b* ⟹
learned-clss-count xb ≤ *unat64-max*›
by (*auto simp*: *isasat-fast-def*)

sepref-def *remove-one-annot-true-clause-one-imp-wl-D-heur-code*
is ‹*uncurry remove-one-annot-true-clause-one-imp-wl-D-heur*›
:: ‹[λ(*C*, *S*). *learned-clss-count S* ≤ *unat64-max*]$_a$
sint64-nat-assn^k $*_a$ *isasat-bounded-assn*^{d} → *sint64-nat-assn* $\times_a$ *isasat-bounded-assn*›
supply [[*goals-limit*=*1*]] *remove-one-annot-true-clause-one-imp-wl-D-heurI*[*intro*]
unfolding *remove-one-annot-true-clause-one-imp-wl-D-heur-def*
isasat-trail-nth-st-def[*symmetric*] *get-the-propagation-reason-pol-st-def*[*symmetric*]
fold-tuple-optimizations
apply (*rewrite* **in** ‹*- = ¤*› *snat-const-fold*(*1*)[**where** *'a*=*64*])
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-register *remove-one-annot-true-clause-imp-wl-D-heur*

lemma *remove-one-annot-true-clause-imp-wl-D-heurI*:
‹*learned-clss-count x* ≤ *unat64-max* ⟹
remove-one-annot-true-clause-imp-wl-D-heur-inv x (*a1'*, *a2'*) ⟹
learned-clss-count a2' ≤ *unat64-max*›
by (*auto simp*: *isasat-fast-def remove-one-annot-true-clause-imp-wl-D-heur-inv-def*)

sepref-def *remove-one-annot-true-clause-imp-wl-D-heur-code*
is ‹*remove-one-annot-true-clause-imp-wl-D-heur*›
:: ‹[λ*S*. *length* (*get-clauses-wl-heur S*) ≤ *snat64-max* ∧
learned-clss-count S ≤ *unat64-max*]$_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›

supply [[*goals-limit=1*]] *remove-one-annot-true-clause-imp-wl-D-heurI*[*intro*]
unfolding *remove-one-annot-true-clause-imp-wl-D-heur-def*
isasat-length-trail-st-def[*symmetric*] *get-pos-of-level-in-trail-imp-st-def*[*symmetric*]
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

sepref-register *number-clss-to-keep*

lemma [*sepref-fr-rules*]:
‹(*Mreturn o id*, *RETURN o unat*) ∈ *word64-assn*^{k} →$_a$ *uint64-nat-assn*›
proof −
have [*simp*]: ‹(λ*s*. ∃ *xa*. (↑(*xa* = *unat x*) ∧∗ ↑(*xa* = *unat x*)) *s*) = ↑*True*›
by (*intro ext*)
(*auto intro*!: *exI*[*of* - ‹*unat x*›] *simp*: *pure-true-conv pure-part-pure-eq pred-lift-def*
simp flip: *import-param-3*)
show *?thesis*
apply *sepref-to-hoare*
apply (*vcg*)
apply (*auto simp*: *unat-rel-def unat.rel-def br-def pred-lift-def ENTAILS-def pure-true-conv simp flip*: *import-param-3 pure-part-def*)
done
qed

sepref-def *number-clss-to-keep-fast-code*
is ‹*number-clss-to-keep-impl*›
:: ‹*isasat-bounded-assn*^{k} →$_a$ *sint64-nat-assn*›
supply [[*goals-limit* = *1*]]
unfolding *number-clss-to-keep-impl-def length-tvdom-def*[*symmetric*] *length-tvdom-aivdom-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *number-clss-to-keep-impl-number-clss-to-keep*:
‹(*number-clss-to-keep-impl*, *number-clss-to-keep*) ∈ *Sepref-Rules.freft Id* (λ-. ⟨*nat-rel*⟩*nres-rel*)›
by (*auto simp*: *number-clss-to-keep-impl-def number-clss-to-keep-def Let-def intro*!: *Sepref-Rules.frefI nres-relI*)

lemma *number-clss-to-keep-fast-code-refine*[*sepref-fr-rules*]:
‹(*number-clss-to-keep-fast-code*, *number-clss-to-keep*) ∈ (*isasat-bounded-assn*)^k →$_a$ *snat-assn*›
using *hfcomp*[*OF number-clss-to-keep-fast-code.refine*
number-clss-to-keep-impl-number-clss-to-keep, *simplified*]
by *auto*

experiment
begin
export-llvm *restart-required-heur-fast-code access-avdom-at-fast-code*
trail-zeroed-until-state-fast-code
end

end
theory *IsaSAT-Restart-Simp-Defs*
imports *IsaSAT-Restart-Heuristics-Defs IsaSAT-Other-Defs IsaSAT-Propagate-Conflict-Defs IsaSAT-Restart-Inprocess*
Watched-Literals.Watched-Literals-Watch-List-Reduce
begin

Chapter 23

Full CDCL with Restarts

definition *cdcl-twl-stgy-restart-abs-wl-heur-inv* **where**
‹*cdcl-twl-stgy-restart-abs-wl-heur-inv* S_0 = (λ(*brk*, *T*, *last-GC*, *last-Rephase*).
(∃ S_0' *T'*. (S_0, S_0') ∈ *twl-st-heur* ∧ (*T*, *T'*) ∈ *twl-st-heur-loop* ∧
(¬*brk* ⟶ *cdcl-twl-stgy-restart-abs-wl-inv* S_0' (*brk*, *T'*, *last-GC*, *last-Rephase*))))›

definition *cdcl-twl-stgy-restart-abs-wl-heur-inv2* **where**
‹*cdcl-twl-stgy-restart-abs-wl-heur-inv2* S_0 = (λ(*brk*, *T*, *last-GC*, *last-Rephase*).
(∃ S_0' *T'*. (S_0, S_0') ∈ *twl-st-heur-loop* ∧ (*T*, *T'*) ∈ *twl-st-heur-loop* ∧
(¬*brk*⟶*cdcl-twl-stgy-restart-abs-wl-inv* S_0' (*brk*, *T'*, *last-GC*, *last-Rephase*))))›

It would be better to add a backtrack to level 0 before instead of delaying the restart.

definition *update-all-phases* :: ‹*isasat* ⇒ (*isasat*) *nres*› **where**
‹*update-all-phases* = (λ*S*. *do* {
if (*count-decided-st-heur S* = *0*) *then do* {
let lcount = *get-global-conflict-count S*;
end-of-restart-phase ← *RETURN* (*end-of-restart-phase-st S*);
S ← (*if end-of-restart-phase* < *lcount then update-restart-phases S else RETURN S*);
S ← (*if end-of-rephasing-phase-st S* < *lcount then rephase-heur-st S else RETURN S*);
RETURN S
}
else RETURN S
})›

definition *isasat-fast-slow* :: ‹*isasat* ⇒ *isasat nres*› **where**
[*simp*]: ‹*isasat-fast-slow S* = *RETURN S*›

definition *cdcl-twl-stgy-restart-prog-early-wl-heur*
:: ‹*isasat* ⇒ *isasat nres*›
where
‹*cdcl-twl-stgy-restart-prog-early-wl-heur* S_0 = *do* {
ebrk ← *RETURN* (¬*isasat-fast* S_0);
(*ebrk*, *brk*, *T*, *n*) ←
$WHILE_T$^λ(*ebrk*, *brk*, *T*, *last-GC*, *last-Restart*, *n*). *cdcl-twl-stgy-restart-abs-wl-heur-inv* S_0 (*brk*, *T*, *last-GC*, *last-Resta*
(λ(*ebrk*, *brk*, -). ¬*brk* ∧ ¬*ebrk*)
(λ(*ebrk*, *brk*, *S*, *last-GC*, *last-Restart*, *n*).
do {
ASSERT(¬*brk* ∧ ¬*ebrk*);
ASSERT(*length* (*get-clauses-wl-heur S*) ≤ *unat64-max*);
T ← *unit-propagation-outer-loop-wl-D-heur S*;
ASSERT(*length* (*get-clauses-wl-heur T*) ≤ *unat64-max*);
ASSERT(*length* (*get-clauses-wl-heur T*) = *length* (*get-clauses-wl-heur S*));

```
      (brk, T) ← cdcl-twl-o-prog-wl-D-heur T;
      ASSERT(length (get-clauses-wl-heur T) ≤ unat64-max);
      (T, n) ← restart-prog-wl-D-heur T last-GC last-Restart n brk;
ebrk ← RETURN (¬isasat-fast T);
      RETURN (ebrk, brk ∨ ¬get-conflict-wl-is-None-heur T, T, n)
    })
    (ebrk, False, S₀::isasat, learned-clss-count S₀, learned-clss-count S₀, 0);
  ASSERT(length (get-clauses-wl-heur T) ≤ unat64-max ∧
    get-old-arena T = []);
  if ¬brk then do {
    T ← isasat-fast-slow T;
    (brk, T, -) ← WHILE_T^{cdcl-twl-stgy-restart-abs-wl-heur-inv2 T}
      (λ(brk, -). ¬brk)
      (λ(brk, S, last-GC, last-Restart, n).
      do {
        T ← unit-propagation-outer-loop-wl-D-heur S;
        (brk, T) ← cdcl-twl-o-prog-wl-D-heur T;
        (T, last-GC, last-Restart, n) ← restart-prog-wl-D-heur T last-GC last-Restart n brk;
        RETURN (brk ∨ ¬get-conflict-wl-is-None-heur T, T, last-GC, last-Restart, n)
      })
      (False, T, n);
    RETURN T
  }
  else isasat-fast-slow T
}›

definition cdcl-twl-stgy-restart-prog-bounded-wl-heur
  :: ‹isasat ⇒ (bool × isasat) nres›
where
‹cdcl-twl-stgy-restart-prog-bounded-wl-heur S₀ = do {
  ebrk ← RETURN (¬isasat-fast S₀);
  (ebrk, brk, T, n) ←
  WHILE_T^{λ(ebrk, brk, T, last-GC, last-Restart, n). cdcl-twl-stgy-restart-abs-wl-heur-inv S₀ (brk, T, last-GC, last-Restart, n)}
    (λ(ebrk, brk, -). ¬brk ∧ ¬ebrk)
    (λ(ebrk, brk, S, last-GC, last-Restart, n).
    do {
      ASSERT(¬brk ∧ ¬ebrk);
      ASSERT(isasat-fast S);
      T ← unit-propagation-outer-loop-wl-D-heur S;
      ASSERT(isasat-fast T);
      (brk, T) ← cdcl-twl-o-prog-wl-D-heur T;
      ASSERT(isasat-fast-relaxed2 T n);
      (T, last-GC, last-Restart, n) ← restart-prog-wl-D-heur T last-GC last-Restart n brk;
      T ← update-all-phases T;
      ASSERT(isasat-fast-relaxed T);
      ebrk ← RETURN (¬(isasat-fast T ∧ n < unat64-max));
      RETURN (ebrk, brk ∨ ¬get-conflict-wl-is-None-heur T, T, last-GC, last-Restart, n)
    })
    (ebrk, False, S₀::isasat, learned-clss-count S₀, learned-clss-count S₀, 0);
  RETURN (ebrk, T)
}›


definition cdcl-twl-stgy-restart-prog-wl-heur
  :: ‹isasat ⇒ isasat nres›
where
```

```
  ‹cdcl-twl-stgy-restart-prog-wl-heur S₀ = do {
    (brk, T, -) ← WHILE_T^cdcl-twl-stgy-restart-abs-wl-heur-inv S₀
      (λ(brk, -). ¬brk)
      (λ(brk, S, last-GC, last-Rephase, n).
      do {
        T ← unit-propagation-outer-loop-wl-D-heur S;
        (brk, T) ← cdcl-twl-o-prog-wl-D-heur T;
        (T, last-GC, last-Rephase, n) ← restart-prog-wl-D-heur T last-GC last-Rephase n brk;
        RETURN (brk ∨ ¬get-conflict-wl-is-None-heur T, T, last-GC, last-Rephase, n)
      })
      (False, S₀::isasat, learned-clss-count S₀, learned-clss-count S₀, 0);
    RETURN T
  }›

end
theory IsaSAT-Restart-LLVM
 imports IsaSAT-Restart-Simp-Defs IsaSAT-Propagate-Conflict-LLVM IsaSAT-Restart-Heuristics-LLVM

begin

sepref-register update-all-phases

sepref-def update-all-phases-impl
  is ‹update-all-phases›
  :: ‹isasat-bounded-assn^d →_a isasat-bounded-assn›
  unfolding update-all-phases-def
  apply (annot-unat-const ‹TYPE(32)›)
  by sepref

sepref-register mark-to-delete-clauses-wl-D-heur


sepref-register delete-index-and-swap mop-mark-garbage-heur mop-mark-garbage-heur3
  mop-access-lit-in-clauses-heur


lemma [def-pat-rules]: ‹get-the-propagation-reason-heur ≡ get-the-propagation-reason-pol-st›
proof −
  have ‹get-the-propagation-reason-heur = get-the-propagation-reason-pol-st›
    by (auto intro!: ext simp: get-the-propagation-reason-pol-st-def
      get-the-propagation-reason-heur-def)
  then show  ‹get-the-propagation-reason-heur ≡ get-the-propagation-reason-pol-st›
   by auto
qed

sepref-def mark-to-delete-clauses-wl-D-heur-fast-impl
  is ‹mark-to-delete-clauses-wl-D-heur›
  :: ‹[λS. length (get-clauses-wl-heur S) ≤ snat64-max]_a isasat-bounded-assn^d → isasat-bounded-assn›
  unfolding mark-to-delete-clauses-wl-D-heur-def
    access-tvdom-at-def[symmetric] length-tvdom-def[symmetric]
    get-the-propagation-reason-heur-def[symmetric]
    clause-is-learned-heur-def[symmetric]
    clause-lbd-heur-def[symmetric]
    access-length-heur-def[symmetric]
    mark-to-delete-clauses-wl-D-heur-is-Some-iff
    marked-as-used-st-def[symmetric] if-conn(4)
```

fold-tuple-optimizations
mop-arena-lbd-st-def[*symmetric*]
mop-marked-as-used-st-def[*symmetric*]
mop-arena-status-st-def[*symmetric*]
mop-arena-length-st-def[*symmetric*]
supply [[*goals-limit* = *1*]] *of-nat-snat*[*sepref-import-param*]
length-tvdom-def[*symmetric*, *simp*] *access-tvdom-at-def*[*simp*]
apply (*rewrite* **in** ‹*let* - = ⌑ *in* -› *short-circuit-conv*)+
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-def *mark-to-delete-clauses-GC-wl-D-heur-heur-fast-impl*
is ‹*mark-to-delete-clauses-GC-wl-D-heur*›
:: ‹⟦λ*S*. *length* (*get-clauses-wl-heur S*) ≤ *snat64-max*⟧$_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
unfolding *mark-to-delete-clauses-GC-wl-D-heur-def*
access-tvdom-at-def[*symmetric*] *length-tvdom-def*[*symmetric*]
get-the-propagation-reason-heur-def[*symmetric*]
clause-is-learned-heur-def[*symmetric*]
clause-lbd-heur-def[*symmetric*]
access-length-heur-def[*symmetric*]
mark-to-delete-clauses-wl-D-heur-is-Some-iff
marked-as-used-st-def[*symmetric*] *if-conn*(*4*)
fold-tuple-optimizations
mop-arena-lbd-st-def[*symmetric*]
mop-marked-as-used-st-def[*symmetric*]
mop-arena-status-st-def[*symmetric*]
mop-arena-length-st-def[*symmetric*]
supply [[*goals-limit* = *1*]] *of-nat-snat*[*sepref-import-param*]
length-avdom-def[*symmetric*, *simp*] *access-avdom-at-def*[*simp*]
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

definition *isasat-fast-bound* **where**
‹*isasat-fast-bound* = *snat64-max* − (*unat32-max div 2* + *MAX-HEADER-SIZE* + *1*)›

lemma *isasat-fast-bound-alt-def*:
‹*isasat-fast-bound* = *9223372034707292156*›
‹*unat64-max* = *18446744073709551615*›
by (*auto simp*: *br-def isasat-fast-bound-def*
snat64-max-def unat32-max-def unat64-max-def)

lemma *isasat-fast-alt-def*: ‹*isasat-fast S* = (*length-clauses-heur S* ≤ *9223372034707292156* ∧
clss-size-lcount (*get-learned-count S*) +
clss-size-lcountUE (*get-learned-count S*) + *clss-size-lcountUS* (*get-learned-count S*) +
clss-size-lcountU0 (*get-learned-count S*)+
clss-size-lcountUEk (*get-learned-count S*) < *18446744073709551615*)›
by (*cases S*; *auto simp*: *isasat-fast-def snat64-max-def unat32-max-def length-clauses-heur-def*
unat64-max-def learned-clss-count-def)

sepref-register *isasat-fast*

lemma *all-count-learned*[*simp*]: ‹*clss-size-allcount* (*get-learned-count S*) = *learned-clss-count S*›
by (*auto simp*: *twl-st-heur′-def clss-size-allcount-def learned-clss-count-def clss-size-lcountU0-def*
clss-size-lcount-def clss-size-lcountUE-def clss-size-lcountUS-def clss-size-lcountUEk-def

split: *prod.splits*)

sepref-def *isasat-fast-code*
is ‹*RETURN o isasat-fast*›
:: ‹[λ*S*. *isasat-fast-relaxed S*]$_a$ *isasat-bounded-assn*^{k} → *bool1-assn*›
unfolding *isasat-fast-alt-def isasat-fast-relaxed-def*
apply (*rewrite at* ‹- < ¤› *unat-const-fold*[**where** *'a=64*])+
unfolding *all-count-learned*[*symmetric*] *clss-size-allcount-alt-def*
supply [[*goals-limit* = *1*]]
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

sepref-register *should-inprocess-st*

end
theory *IsaSAT-Restart-Simp*
imports *IsaSAT-Restart-Heuristics IsaSAT-Other IsaSAT-Propagate-Conflict IsaSAT-Restart-Inprocessing*
IsaSAT-Restart-Simp-Defs
begin

fun *Pair4* :: ‹*'a* ⇒ *'b* ⇒ *'c* ⇒ *'d* ⇒ *'a* × *'b* × *'c* × *'d*› **where**
‹*Pair4 a b c d* = (*a*, *b*, *c*, *d*)›

lemma *rephase-heur-st-spec*:
‹(*S*, *S'*) ∈ *twl-st-heur-loop* ⟹ *rephase-heur-st S* ≤ *SPEC*(λ*S*. (*S*, *S'*) ∈ *twl-st-heur-loop*)›
unfolding *rephase-heur-st-def*
apply (*cases S'*)
apply (*refine-vcg rephase-heur-spec*[*THEN order-trans*, *of* ‹*all-atms-st S'*›])
apply (*simp-all add*: *twl-st-heur-loop-def all-atms-st-def*)
done

lemma *update-all-phases-Pair*:
‹(*S*, *S'*) ∈ *twl-st-heur-loop''''uu r u*⟹
update-all-phases S ≤ ⇓ ({(*T*, *T'*). (*T*,*T'*)∈*twl-st-heur-loop''''uu r u* ∧ *T'* = *S'*}) (*RETURN* (*id S'*))›
proof −
have [*refine0*]: ‹(*S*, *S'*) ∈ *twl-st-heur-loop''''uu r u* ⟹ *count-decided* (*get-trail-wl S'*) = *0* ⟹
update-restart-phases S ≤ *SPEC*(λ*S*. (*S*, *S'*) ∈ *twl-st-heur-loop''''uu r u*)›
for *S* :: *isasat* **and** *S'* :: ‹*nat twl-st-wl*›
unfolding *update-all-phases-def update-restart-phases-def Let-def*
by (*auto simp*: *twl-st-heur'-def twl-st-heur-loop-def learned-clss-count-def*
intro!: *rephase-heur-st-spec*[*THEN order-trans*] *switch-bump-heur*
simp del: *incr-restart-phase-end-stats.simps incr-restart-phase-stats.simps*)
have [*refine0*]: ‹(*S*, *S'*) ∈ *twl-st-heur-loop''''uu r u* ⟹ *rephase-heur-st S* ≤ *SPEC*(λ*S*. (*S*, *S'*) ∈ *twl-st-heur-loop''''uu r u*)›
for *S* :: *isasat* **and** *S'* :: ‹*nat twl-st-wl*›
unfolding *update-all-phases-def rephase-heur-st-def*
apply (*cases S'*)
apply (*refine-vcg rephase-heur-spec*[*THEN order-trans*, *of* ‹*all-atms-st S'*›])
apply (*clarsimp simp*: *twl-st-heur'-def twl-st-heur-loop-def learned-clss-count-def*)
apply (*simp add*: *learned-clss-count-def*)
apply (*clarsimp simp add*: *twl-st-heur-loop-def learned-clss-count-def*)
done

show ‹(*S*, *S'*) ∈ *twl-st-heur-loop''''uu r u*⟹
update-all-phases S ≤ ⇓ ({(*T*, *T'*). (*T*,*T'*)∈*twl-st-heur-loop''''uu r u* ∧ *T'* = *S'*}) (*RETURN* (*id*

```
S'))› for S S'
    unfolding update-all-phases-def
    apply (subst (1) bind-to-let-conv)
    apply (subst (1) Let-def)
    apply (subst (1) Let-def)
    apply refine-vcg
    apply assumption
    subgoal
      using count-decided-trail-ref[THEN fref-to-Down-unRET-Id, of ‹get-trail-wl-heur S›
        ‹get-trail-wl S'› ‹all-atms-st S'›]
      by (simp add: count-decided-st-def twl-st-heur-loop-def count-decided-st-heur-def Let-def)
    apply assumption
    subgoal by simp
    subgoal by simp
    apply assumption
    subgoal by simp
    subgoal by simp
    subgoal by simp
    done
qed

lemma cdcl-twl-stgy-restart-abs-wl-inv-NoneD:
  ‹cdcl-twl-stgy-restart-abs-wl-inv y (False, x1a, x1b, x1c, x2c) ⟹
  get-conflict-wl x1a = None›
  unfolding cdcl-twl-stgy-restart-abs-wl-inv-def cdcl-twl-stgy-restart-abs-l-inv-def
    prod.simps cdcl-twl-stgy-restart-prog-inv-def cdcl-twl-stgy-restart-prog-int-inv-def
  unfolding not-False-eq-True simp-thms
  apply normalize-goal+
  by simp


lemma get-conflict-wl-is-None-heur-get-conflict-wl-is-None:
  ‹(RETURN o get-conflict-wl-is-None-heur,  RETURN o get-conflict-wl-is-None) ∈
    twl-st-heur-loop →f ⟨Id⟩nres-rel›
  unfolding get-conflict-wl-is-None-heur-def get-conflict-wl-is-None-def comp-def
  apply (intro frefI nres-relI) apply refine-rcg
  by (auto simp: twl-st-heur-loop-def get-conflict-wl-is-None-heur-def get-conflict-wl-is-None-def
      option-lookup-clause-rel-def
    split: option.splits)

lemma cdcl-twl-stgy-restart-prog-wl-heur-cdcl-twl-stgy-restart-prog-wl-D:
  ‹(cdcl-twl-stgy-restart-prog-wl-heur, cdcl-twl-stgy-restart-prog-wl) ∈
    twl-st-heur →f ⟨twl-st-heur-loop⟩nres-rel›
proof −
  have [refine0]: ‹(x1e, x1a) ∈ twl-st-heur ⟹ (x1e, x1a)
      ∈ {(S, T).
        (S, T) ∈ twl-st-heur ∧
        get-learned-count S = get-learned-count x1e}› for x1e x1a
    by auto
  show ?thesis
    unfolding cdcl-twl-stgy-restart-prog-wl-heur-def cdcl-twl-stgy-restart-prog-wl-def
    apply (intro frefI nres-relI)
    apply (refine-rcg
        restart-prog-wl-D-heur-restart-prog-wl-D2[THEN fref-to-Down-curry4]
        cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D2[THEN fref-to-Down]
        unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D[THEN fref-to-Down]
```

```
    WHILEIT-refine[where R = ‹bool-rel ×r restart-prog-wl-heur-rel2›])
  subgoal by (auto simp: learned-clss-count-twl-st-heur intro!: twl-st-heur-twl-st-heur-loopD)
  subgoal for x y xa x'
    using cdcl-twl-stgy-restart-abs-wl-inv-NoneD[of y ‹fst (snd x')›
      ‹fst (snd (snd x'))› ‹fst (snd (snd (snd x')))› ‹snd (snd (snd (snd x')))›] apply -
    unfolding cdcl-twl-stgy-restart-abs-wl-heur-inv-def prod-rel-fst-snd-iff case-prod-beta
    apply (rule-tac x = ‹y› in exI)
    apply (rule-tac x = ‹fst (snd x')› in exI)
    apply (cases x', cases xa)
    by auto
  subgoal by auto
  subgoal
    by (rule twl-st-heur-loop-twl-st-heurD)
     (auto dest: cdcl-twl-stgy-restart-abs-wl-inv-NoneD)
  subgoal by auto
  subgoal by (auto dest!: cdcl-twl-stgy-restart-abs-wl-inv-NoneD)
  subgoal unfolding get-conflict-wl-is-None
   by (auto simp: get-conflict-wl-is-None-heur-get-conflict-wl-is-None[THEN fref-to-Down-unRET-Id])
  subgoal by auto
  done
qed

definition fast-number-of-iterations :: ‹- ⇒ bool› where
‹fast-number-of-iterations n ⟷ n < unat64-max >> 1›


definition convert-to-full-state-wl-heur :: ‹isasat ⇒ isasat nres› where
  [simp]: ‹convert-to-full-state-wl-heur S = RETURN S›
definition convert-to-full-state-wl :: ‹'v twl-st-wl ⇒ 'v twl-st-wl nres› where
  [simp]: ‹convert-to-full-state-wl S = RETURN S›

lemma convert-to-full-state-wl-heur:
  ‹(S, T) ∈ twl-st-heur-loop ⟹ get-conflict-wl T = None ⟹
  convert-to-full-state-wl-heur S ≤ ⇓(twl-st-heur''
        (dom-m (get-clauses-wl T))
        (length (get-clauses-wl-heur S))
         (get-learned-count S)) (convert-to-full-state-wl T)›
  by (auto intro!: nres-relI frefI twl-st-heur-loop-twl-st-heurD simp: twl-st-heur'-def)


lemma cdcl-twl-stgy-restart-prog-early-wl-heur-alt-def:
  ‹cdcl-twl-stgy-restart-prog-early-wl-heur S_0 = do {
    ebrk ← RETURN (¬isasat-fast S_0);
    (ebrk, brk, T, n) ←
     WHILE_T^λ(ebrk, brk, T, last-GC, last-Restart, n).       cdcl-twl-stgy-restart-abs-wl-heur-inv S_0 (brk, T, last-GC, last-Resta
     (λ(ebrk, brk, -). ¬brk ∧ ¬ebrk)
     (λ(ebrk, brk, S, last-GC, last-Restart, n).
     do {
       ASSERT(¬brk ∧ ¬ebrk);
       ASSERT(length (get-clauses-wl-heur S) ≤ unat64-max);
       S ← convert-to-full-state-wl-heur S;
       T ← unit-propagation-outer-loop-wl-D-heur S;
       ASSERT(length (get-clauses-wl-heur T) ≤ unat64-max);
       ASSERT(length (get-clauses-wl-heur T) = length (get-clauses-wl-heur S));
       (brk, T) ← cdcl-twl-o-prog-wl-D-heur T;
       ASSERT(length (get-clauses-wl-heur T) ≤ unat64-max);
```

```
      (T, n) ← restart-prog-wl-D-heur T last-GC last-Restart n brk;
 ebrk ← RETURN (¬isasat-fast T);
      RETURN (ebrk, brk ∨ ¬get-conflict-wl-is-None-heur T, T, n)
    })
    (ebrk, False, S₀::isasat, learned-clss-count S₀, learned-clss-count S₀,  0);
  ASSERT(length (get-clauses-wl-heur T) ≤ unat64-max ∧
      get-old-arena T = []);
  if ¬brk then do {
    T ← isasat-fast-slow T;
    (brk, T, -) ← WHILE_T^cdcl-twl-stgy-restart-abs-wl-heur-inv2 T
      (λ(brk, -). ¬brk)
      (λ(brk, S, last-GC, last-Restart, n).
      do {
            S ← convert-to-full-state-wl-heur S;
        T ← unit-propagation-outer-loop-wl-D-heur S;
        (brk, T) ← cdcl-twl-o-prog-wl-D-heur T;
        (T, last-GC, last-Restart, n) ← restart-prog-wl-D-heur T last-GC last-Restart n brk;
        RETURN (brk ∨ ¬get-conflict-wl-is-None-heur T, T, last-GC, last-Restart, n)
      })
      (False, T, n);
    RETURN T
  }
  else isasat-fast-slow T
              }›
  unfolding convert-to-full-state-wl-heur-def Let-def cdcl-twl-stgy-restart-prog-early-wl-heur-def
    bind-to-let-conv
  by auto

lemma cdcl-twl-stgy-restart-prog-early-wl-heur-cdcl-twl-stgy-restart-prog-early-wl-D:
  assumes r: ‹r ≤ unat64-max›
  shows ‹(cdcl-twl-stgy-restart-prog-early-wl-heur, cdcl-twl-stgy-restart-prog-early-wl) ∈
   twl-st-heur''' r →_f ⟨twl-st-heur-loop⟩nres-rel›
proof −
  have cdcl-twl-stgy-restart-prog-early-wl-alt-def:
  ‹cdcl-twl-stgy-restart-prog-early-wl S₀ = do {
      ebrk ← RES UNIV;
      (ebrk, brk, T, last-GC, last-Res, n) ← WHILE_T^cdcl-twl-stgy-restart-abs-wl-inv S₀ o snd
        (λ(ebrk, brk, -). ¬brk ∧ ¬ebrk)
        (λ(-, brk, S, last-GC, last-Res, n).
        do {
              S ← convert-to-full-state-wl S;
          T ← unit-propagation-outer-loop-wl S;
          (brk, T) ← cdcl-twl-o-prog-wl T;
          (T, last-GC, last-Res, n) ← restart-prog-wl T last-GC last-Res n brk;
          ebrk ← RES UNIV;
          RETURN (ebrk, brk ∨ get-conflict-wl T ≠ None, T, last-GC, last-Res, n)
        })
         (ebrk, False, S₀::nat twl-st-wl, size (get-all-learned-clss-wl S₀),
           size (get-all-learned-clss-wl S₀),0);
      if ¬brk then do {
        T ← RETURN T;
 (brk, T, -) ← WHILE_T^cdcl-twl-stgy-restart-abs-wl-inv T
   (λ(brk, -). ¬brk)
   (λ(brk, S, last-GC, last-Res, n).
     do {
          S ← convert-to-full-state-wl S;
```

```
    T ← unit-propagation-outer-loop-wl S;
    (brk, T) ← cdcl-twl-o-prog-wl T;
    (T, last-GC, last-Res, n) ← restart-prog-wl T last-GC last-Res n brk;
    RETURN (brk ∨ get-conflict-wl T ≠ None, T, last-GC, last-Res, n)
  })
  (False, T::nat twl-st-wl, last-GC, last-Res, n);
RETURN T
     }
     else RETURN T
   }› for S_0
     unfolding cdcl-twl-stgy-restart-prog-early-wl-def nres-monad1 bind-to-let-conv
     by (auto intro!: bind-cong)
 have [refine0]: ‹RETURN (¬isasat-fast x) ≤ ⇓
    {(b, b'). b = b' ∧ (b = (¬isasat-fast x))} (RES UNIV)›
   for x
   by (auto intro: RETURN-RES-refine)
 have [refine0]: ‹isasat-fast-slow x1e
    ≤ ⇓ {(S, S'). S = x1e ∧ S' = x1b}
   (RETURN x1b)›
   for x1e x1b
 proof −
   show ?thesis
     unfolding isasat-fast-slow-def by auto
 qed

 let ?R = ‹{((ebrk, brk, T, last-GC, last-Rephase, n),
       (ebrk', brk', T', last-GC', last-Rephase', n')).
   (ebrk = ebrk') ∧ (brk = brk') ∧ (T, T') ∈ twl-st-heur-loop ∧
   (¬brk ⟶ (n = n' ∧ last-GC' = last-GC ∧ last-Rephase' = last-Rephase)) ∧
     (¬ebrk ⟶ isasat-fast T) ∧ length (get-clauses-wl-heur T) ≤ unat64-max}›
 let ?S = ‹{((brk, T, last-GC, last-Rephase, n),
       (brk', T', last-GC', last-Rephase', n')).
    (brk = brk') ∧ (T, T') ∈ twl-st-heur-loop ∧
   (¬brk ⟶ (n = n' ∧ last-GC' = last-GC ∧ last-Rephase' = last-Rephase))}›
 have twl-st-heur'': ‹(x1e, x1b) ∈ twl-st-heur-loop ⟹ get-conflict-wl x1b = None ⟹
   (convert-to-full-state-wl-heur x1e) ≤⇓
   (twl-st-heur''
      (dom-m (get-clauses-wl x1b))
      (length (get-clauses-wl-heur x1e))
      (get-learned-count x1e)) (convert-to-full-state-wl x1b)›
   for x1e x1b
   by (auto simp: twl-st-heur'-def intro!: nres-relI twl-st-heur-loop-twl-st-heurD)
 have twl-st-heur''': ‹(x1e, x1b) ∈ twl-st-heur'' 𝒟 r lcount ⟹
   (x1e, x1b)
   ∈ {(S, Taa).
       (S, Taa) ∈ twl-st-heur ∧
       length (get-clauses-wl-heur S) = r ∧
       learned-clss-count S = clss-size-allcount lcount}›
   for x1e x1b r 𝒟 lcount
   by (auto simp: twl-st-heur'-def clss-size-allcount-def learned-clss-count-def clss-size-lcountU0-def
     clss-size-lcount-def clss-size-lcountUE-def clss-size-lcountUS-def clss-size-lcountUEk-def
     split: prod.splits)
 have H: ‹(xb, x'a)
   ∈ bool-rel ×_f
     twl-st-heur'''' (length (get-clauses-wl-heur x1e) + MAX-HEADER-SIZE+1 + unat32-max div 2)
⟹
```

$x'a = (x1f,\ x2f) \Longrightarrow$
$xb = (x1g,\ x2g) \Longrightarrow$
$(x1g,\ x1f) \in bool\text{-}rel \Longrightarrow$
$(x2e,\ x2b) \in nat\text{-}rel \Longrightarrow$
$(((x2g,\ x2e),\ x1g),\ (x2f,\ x2b),\ x1f)$
$\in twl\text{-}st\text{-}heur''''\ (length\ (get\text{-}clauses\text{-}wl\text{-}heur\ x2g)) \times_f$
$nat\text{-}rel \times_f$
$bool\text{-}rel\rangle$ **for** $x\ y\ ebrk\ ebrka\ xa\ x'\ x1\ x2\ x1a\ x2a\ x1b\ x2b\ x1c\ x2c\ x1d\ x2d\ x1e\ x2e\ T\ Ta\ xb$
$x'a\ x1f\ x2f\ x1g\ x2g$
by *auto*

have H''':
$\langle\bigwedge x\ y\ ebrk\ ebrka\ xa\ x'\ x1\ x2\ x1a\ x2a\ x1b\ x2b\ x1c\ x2c\ x1d\ x2d\ x1e\ x2e\ x1f\ x2f\ x1g\ x2g\ x1h\ x2h\ x1i\ x2i$
$S\ Sa\ T\ Ta$
$xb\ x'a\ x1j\ x2j\ x1k\ x2k.$
$(xa,\ x') \in ?R \Longrightarrow$
$x2c = (x1d,\ x2d) \Longrightarrow$
$x2b = (x1c,\ x2c) \Longrightarrow$
$x2a = (x1b,\ x2b) \Longrightarrow$
$x2 = (x1a,\ x2a) \Longrightarrow$
$x' = (x1,\ x2) \Longrightarrow$
$x2h = (x1i,\ x2i) \Longrightarrow$
$x2g = (x1h,\ x2h) \Longrightarrow$
$x2f = (x1g,\ x2g) \Longrightarrow$
$x2e = (x1f,\ x2f) \Longrightarrow$
$xa = (x1e,\ x2e) \Longrightarrow$
$\neg\ x1f \wedge \neg\ x1e \Longrightarrow$
$length\ (get\text{-}clauses\text{-}wl\text{-}heur\ x1g) \leq unat64\text{-}max \Longrightarrow$
$(xb,\ x'a)$
$\in bool\text{-}rel \times_f$
$twl\text{-}st\text{-}heur''''uu\ (length\ (get\text{-}clauses\text{-}wl\text{-}heur\ x1g) + 3 + 1 + unat32\text{-}max\ div\ 2)$
$(Suc\ (clss\text{-}size\text{-}allcount\ (get\text{-}learned\text{-}count\ x1g))) \Longrightarrow$
$x'a = (x1j,\ x2j) \Longrightarrow$
$xb = (x1k,\ x2k) \Longrightarrow$
$(((((x2k,\ x1h),\ x1i),\ x2i),\ x1k),\ (((x2j,\ x1c),\ x1d),\ x2d),\ x1j)$
$\in twl\text{-}st\text{-}heur''''u\ (length\ (get\text{-}clauses\text{-}wl\text{-}heur\ x2k))$
$(Suc\ (clss\text{-}size\text{-}allcount\ (get\text{-}learned\text{-}count\ x1g))) \times_f nat\text{-}rel \times_f nat\text{-}rel \times_f nat\text{-}rel \times_f bool\text{-}rel\rangle$
by *simp*

have $H4$: $\langle\bigwedge x\ y\ ebrk\ ebrka\ xa\ x'\ x1\ x2\ x1a\ x2a\ x1b\ x2b\ x1c\ x2c\ x1d\ x2d\ x1e\ x2e\ x1f\ x2f\ x1g\ x2g\ T\ Ta$
$xb\ x'a\ x1h\ x2h$
$x1i\ x2i\ x1j\ x2j\ x1k\ x2k\ x1l\ x2l\ x1m\ x2m\ x1n\ x2n\ x1o\ x2o\ S\ Sa\ Tb\ Tc\ xc\ x'b\ x1p\ x2p\ x1q\ x2q.$
$(xb,\ x'a) \in ?S \Longrightarrow$
$case\ xb\ of\ (brk,\ uu\text{-}) \Rightarrow \neg\ brk \Longrightarrow$
$case\ x'a\ of\ (brk,\ uu\text{-}) \Rightarrow \neg\ brk \Longrightarrow$
$cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}abs\text{-}wl\text{-}heur\text{-}inv2\ T\ xb \Longrightarrow$
$cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}abs\text{-}wl\text{-}inv\ Ta\ x'a \Longrightarrow$
$x2j = (x1k,\ x2k) \Longrightarrow$
$x2i = (x1j,\ x2j) \Longrightarrow$
$x2h = (x1i,\ x2i) \Longrightarrow$
$x'a = (x1h,\ x2h) \Longrightarrow$
$x2n = (x1o,\ x2o) \Longrightarrow$
$x2m = (x1n,\ x2n) \Longrightarrow$
$x2l = (x1m,\ x2m) \Longrightarrow$
$xb = (x1l,\ x2l) \Longrightarrow$
$(S,\ Sa)$

∈ *twl-st-heur''* (*dom-m* (*get-clauses-wl x1i*)) (*length* (*get-clauses-wl-heur x1m*))
(*get-learned-count x1m*) ⟹
(*Tb*, *Tc*)
∈ *twl-st-heur''* (*dom-m* (*get-clauses-wl x1i*)) (*length* (*get-clauses-wl-heur x1m*))
(*get-learned-count x1m*) ⟹
(*xc*, *x'b*)
∈ *bool-rel* ×$_f$
twl-st-heur''''uu (*length* (*get-clauses-wl-heur x1m*) + *3* + *1* + *unat32-max div 2*)
(*Suc* (*clss-size-allcount* (*get-learned-count x1m*))) ⟹
x'b = (*x1p*, *x2p*) ⟹
xc = (*x1q*, *x2q*) ⟹
(((((*x2q*, *x1n*), *x1o*), *x2o*), *x1q*), (((*x2p*, *x1j*), *x1k*), *x2k*), *x1p*)
∈ *twl-st-heur''''u* (*length* (*get-clauses-wl-heur x2q*)) (*learned-clss-count x2q*) ×$_f$
nat-rel ×$_f$
nat-rel ×$_f$
nat-rel ×$_f$
bool-rel›
by *auto*

show *?thesis*
supply[[*goals-limit=1*]] *isasat-fast-length-leD*[*dest*] *twl-st-heur'-def*[*simp*] *learned-clss-count-twl-st-heur*[*simp*]
unfolding *cdcl-twl-stgy-restart-prog-early-wl-heur-alt-def*
cdcl-twl-stgy-restart-prog-early-wl-alt-def
apply (*intro frefI nres-relI*)
apply (*refine-rcg*
restart-prog-wl-D-heur-restart-prog-wl-D[*THEN fref-to-Down-curry4*]
cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D[*THEN fref-to-Down*]
unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D'[*THEN fref-to-Down*]
WHILEIT-refine[**where** *R* = *?S*]
WHILEIT-refine[**where** *R* = ‹*?R*›])
subgoal using *r* **by** (*auto intro*!: *twl-st-heur-twl-st-heur-loopD*)
subgoal for *x y ebrk ebrka xa x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d*
using *cdcl-twl-stgy-restart-abs-wl-inv-NoneD*[*of y* ‹*fst* (*snd* (*snd x'*))›
‹*fst* (*snd* (*snd* (*snd x'*)))› ‹*fst* (*snd* (*snd* (*snd* (*snd x'*))))› ‹*snd* (*snd* (*snd* (*snd* (*snd x'*))))›]
unfolding *cdcl-twl-stgy-restart-abs-wl-heur-inv-def prod.case prod-rel-fst-snd-iff*
apply (*rule-tac x*=*y* **in** *exI*)
apply (*rule-tac x*= ‹*fst* (*snd* (*snd x'*))› **in** *exI*)
apply (*case-tac xa*; *case-tac x'*)
by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *fast*
subgoal by *auto*
apply (*rule twl-st-heur''*; *auto dest*!: *cdcl-twl-stgy-restart-abs-wl-inv-NoneD*; *fail*)
apply *assumption*
subgoal by *auto*
subgoal by *auto*
apply (*rule twl-st-heur'''*; *assumption*)
subgoal by (*auto simp*: *isasat-fast-def unat64-max-def snat64-max-def unat32-max-def*)
apply (*rule H'''*; *assumption*)
subgoal for *x y ebrk ebrka xa x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i S Sa*
T Ta xb x'a x1j x2j x1k x2k xc x'b x1l x2l x1m x2m x1n x2n x1o x2o ebrkb ebrkc
unfolding *get-conflict-wl-is-None*

by (*subst get-conflict-wl-is-None-heur-get-conflict-wl-is-None*[*THEN fref-to-Down-unRET-Id*, *of* - ‹*x1l*›])
clarsimp-all
subgoal by *auto*
subgoal by (*subst* (*asm*)*twl-st-heur-loop-def*) *force*
subgoal by *auto*
subgoal by *auto*
subgoal for *x y ebrk ebrka xa x′ x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g T Ta xb x′a*
using *cdcl-twl-stgy-restart-abs-wl-inv-NoneD*[*of y* ‹*fst* (*snd x′a*)› ‹*fst* (*snd* (*snd x′a*))› ‹*fst* (*snd* (*snd* (*snd x′a*)))› ‹*snd* (*snd* (*snd* (*snd x′a*)))›]
unfolding *cdcl-twl-stgy-restart-abs-wl-heur-inv2-def prod.case prod-rel-fst-snd-iff*
case-prod-beta
apply (*rule-tac x*= ‹*x1b*› **in** *exI*)
apply (*rule-tac x*= ‹*fst* (*snd x′a*)› **in** *exI*)
apply (*case-tac xb*; *case-tac x′a*)
by *auto*
subgoal by *auto*
apply (*rule twl-st-heur′′*)
subgoal by (*auto dest*!: *cdcl-twl-stgy-restart-abs-wl-inv-NoneD*)
subgoal by (*auto dest*!: *cdcl-twl-stgy-restart-abs-wl-inv-NoneD*)
apply *assumption*
apply (*rule twl-st-heur′′′*; *assumption*)
apply (*rule H4*; *assumption*)
subgoal for *x y ebrk ebrka xa x′ x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g T Ta xb x′a x1h x2h x1i x2i x1j x2j x1k x2k x1l x2l x1m x2m x1n x2n x1o x2o S Sa Tb Tc xc x′b x1p x2p x1q x2q xd x′c x1r x2r x1s x2s x1t x2t x1u x2u x1v x2v x1w x2w*
unfolding *get-conflict-wl-is-None*
by (*subst get-conflict-wl-is-None-heur-get-conflict-wl-is-None*[*THEN fref-to-Down-unRET-Id*, *of* - *x1r*])
clarsimp-all
subgoal by *auto*
subgoal by *auto*
done
qed

lemma *mark-unused-st-heur*:
assumes
‹(*S*, *T*) ∈ *twl-st-heur-restart*› **and**
‹*C* ∈# *dom-m* (*get-clauses-wl T*)›
shows ‹(*mark-unused-st-heur C S*, *T*) ∈ *twl-st-heur-restart*›
using *assms*
apply (*cases S*; *cases T*)
apply (*simp add*: *twl-st-heur-restart-def mark-unused-st-heur-def all-init-atms-def*[*symmetric*])
apply (*auto simp*: *twl-st-heur-restart-def mark-garbage-heur-def mark-garbage-wl-def learned-clss-l-l-fmdrop size-remove1-mset-If*
simp: *all-init-atms-def all-init-lits-def*
simp del: *all-init-atms-def*[*symmetric*]
intro!: *valid-arena-mark-unused*
dest!: *in-set-butlastD in-vdom-m-fmdropD*
elim!: *in-set-upd-cases*)
done

lemma *mark-to-delete-clauses-wl-D-heur-is-Some-iff*:
‹D = *Some* $C \longleftrightarrow D \neq$ *None* $\wedge$ ((*the* D) = C)›
by *auto*

lemma *cdcl-twl-stgy-restart-prog-bounded-wl-heur-alt-def*:
‹*cdcl-twl-stgy-restart-prog-bounded-wl-heur* S_0 = *do* {
ebrk ← *RETURN* (¬*isasat-fast* S_0);
(*ebrk*, *brk*, T, n) ←
WHILE$_T$$^{\lambda(ebrk,\ brk,\ T,\ last\text{-}GC,\ last\text{-}Restart,\ n).\ cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}abs\text{-}wl\text{-}heur\text{-}inv\ S_0\ (brk,\ T,\ last\text{-}GC,\ last\text{-}Restart,\ n)}$
(λ(*ebrk*, *brk*, -). ¬*brk* ∧ ¬*ebrk*)
(λ(*ebrk*, *brk*, S, *last-GC*, *last-Restart*, n).
do {
ASSERT(¬*brk* ∧ ¬*ebrk*);
ASSERT(*isasat-fast* S);
S ← *convert-to-full-state-wl-heur* S;
T ← *unit-propagation-outer-loop-wl-D-heur* S;
ASSERT(*isasat-fast* T);
(*brk*, T) ← *cdcl-twl-o-prog-wl-D-heur* T;
ASSERT(*isasat-fast-relaxed2* T n);
(T, *last-GC*, *last-Restart*, n) ← *restart-prog-wl-D-heur* T *last-GC last-Restart* n *brk*;
T ← *update-all-phases* T;
ASSERT(*isasat-fast-relaxed* T);
ebrk ← *RETURN* (¬(*isasat-fast* T ∧ n < *unat64-max*));
RETURN (*ebrk*, *brk* ∨ ¬*get-conflict-wl-is-None-heur* T, T, *last-GC*, *last-Restart*, n)
})
(*ebrk*, *False*, S_0::*isasat*, *learned-clss-count* S_0, *learned-clss-count* S_0, 0);
RETURN (*ebrk*, T)
}›
unfolding *cdcl-twl-stgy-restart-prog-bounded-wl-heur-def bind-to-let-conv Let-def*
convert-to-full-state-wl-heur-def **by** *auto*

lemma *cdcl-twl-stgy-restart-prog-bounded-wl-heur-cdcl-twl-stgy-restart-prog-bounded-wl-D*:
assumes r: ‹$r \leq$ *unat64-max*›
shows ‹(*cdcl-twl-stgy-restart-prog-bounded-wl-heur*, *cdcl-twl-stgy-restart-prog-bounded-wl*) ∈
twl-st-heur''' r →$_f$ ⟨*bool-rel* ×$_r$ *twl-st-heur-loop*⟩*nres-rel*›
proof −
have *cdcl-twl-stgy-restart-prog-bounded-wl-alt-def*:
‹*cdcl-twl-stgy-restart-prog-bounded-wl* S_0 = *do* {
ebrk ← *RES UNIV*;
(*ebrk*, *brk*, T, *last-GC*, *last-Restart*, n) ← *WHILE*$_T$$^{cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}abs\text{-}wl\text{-}inv\ S_0\ o\ snd}$
(λ(*ebrk*, *brk*, -). ¬*brk* ∧ ¬*ebrk*)
(λ(-, *brk*, S, *last-GC*, *last-Restart*,n).
do {
ASSERT (¬ *brk*);
S ← *convert-to-full-state-wl* S;
T ← *unit-propagation-outer-loop-wl* S;
(*brk*, T) ← *cdcl-twl-o-prog-wl* T;
(T, *last-GC*, *last-Restart*, n) ← *restart-prog-wl* T *last-GC last-Restart* n *brk*;
T ← *RETURN* (*id* T);
ebrk ← *RES UNIV*;
RETURN (*ebrk*, *brk* ∨ *get-conflict-wl* $T \neq$ *None*, T, *last-GC*, *last-Restart*, n)
})
(*ebrk*, *False*, S_0::*nat twl-st-wl*, *size* (*get-all-learned-clss-wl* S_0),
size (*get-all-learned-clss-wl* S_0), 0);
RETURN (*ebrk*, T)

```
  }› for S₀
  unfolding cdcl-twl-stgy-restart-prog-bounded-wl-def nres-monad1 Let-def bind-to-let-conv
 convert-to-full-state-wl-def
  by (auto intro!: bind-cong[OF refl])

 have [refine0]: ‹RETURN (¬(isasat-fast x ∧ n < unat64-max)) ≤ ⇓
    {(b, b'). b = b' ∧ (b = (¬(isasat-fast x ∧ n < unat64-max)))} (RES UNIV)›
     ‹RETURN (¬isasat-fast x) ≤ ⇓
    {(b, b'). b = b' ∧ (b = (¬(isasat-fast x ∧ 0 < unat64-max)))} (RES UNIV)›
  for x n
  by (auto intro: RETURN-RES-refine simp: unat64-max-def)
 have [refine0]: ‹isasat-fast-slow x1e
    ≤ ⇓ {(S, S'). S = x1e ∧ S' = x1b}
  (RETURN x1b)›
  for x1e x1b
 proof −
  show ?thesis
   unfolding isasat-fast-slow-def by auto
 qed
 have twl-st-heur'': ‹(x1e, x1b) ∈ twl-st-heur ⟹
  (x1e, x1b)
  ∈ twl-st-heur''
     (dom-m (get-clauses-wl x1b))
     (length (get-clauses-wl-heur x1e))
      (get-learned-count x1e)›
  for x1e x1b
  by (auto simp: twl-st-heur'-def)

 have twl-st-heur''': ‹(x1e, x1b) ∈ twl-st-heur'' 𝒟 r lcount ⟹
  (x1e, x1b)
  ∈ {(S, Taa).
       (S, Taa) ∈ twl-st-heur ∧
       length (get-clauses-wl-heur S) = r ∧
       learned-clss-count S = clss-size-allcount lcount}›
  for x1e x1b r 𝒟 lcount
  by (auto simp: twl-st-heur'-def clss-size-allcount-def learned-clss-count-def clss-size-lcountU0-def
    clss-size-lcount-def clss-size-lcountUE-def clss-size-lcountUS-def clss-size-lcountUEk-def
    split: prod.splits)
 have H: ‹(xb, x'a)
  ∈ bool-rel ×_f
    twl-st-heur'''' (length (get-clauses-wl-heur x1e) + MAX-HEADER-SIZE+1 + unat32-max div 2)
⟹
  x'a = (x1f, x2f) ⟹
  xb = (x1g, x2g) ⟹
  (x1g, x1f) ∈ bool-rel ⟹
  (x2e, x2b) ∈ nat-rel ⟹
  (((x2g, x2e), x1g), (x2f, x2b), x1f)
  ∈ twl-st-heur''' (length (get-clauses-wl-heur x2g)) ×_f
    nat-rel ×_f
    bool-rel› for x y ebrk ebrka xa x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e T Ta xb
     x'a x1f x2f x1g x2g
  by auto
 let ?R = ‹{((ebrk, brk, T, last-GC, last-Rephase, n), ebrk', brk', T',
       last-GC', last-Rephase', n').
       ebrk = ebrk' ∧
       brk = brk' ∧
```

$(T,\ T') \in \textit{twl-st-heur-loop} \wedge$
$(\neg \textit{brk} \longrightarrow (n = n' \wedge \textit{last-GC}' = \textit{last-GC} \wedge \textit{last-Rephase}' = \textit{last-Rephase})) \wedge$
$(\neg\ \textit{ebrk} \longrightarrow \textit{isasat-fast}\ T \wedge n < \textit{unat64-max}) \wedge$
$\textit{length}\ (\textit{get-clauses-wl-heur}\ T) \leq \textit{unat64-max}\}\rangle$

have *H4*: ‹$\bigwedge x\ y\ \textit{ebrk}\ \textit{ebrka}\ \textit{xa}\ x'\ \textit{x1}\ \textit{x2}\ \textit{x1a}\ \textit{x2a}\ \textit{x1b}\ \textit{x2b}\ \textit{x1c}\ \textit{x2c}\ \textit{x1d}\ \textit{x2d}\ \textit{x1e}\ \textit{x2e}\ \textit{x1f}\ \textit{x2f}\ \textit{x1g}\ \textit{x2g}\ \textit{x1h}\ \textit{x2h}\ \textit{x1i}\ \textit{x2i}\ S\ \textit{Sa}$
$T\ \textit{Ta}\ \textit{xb}\ x'a\ \textit{x1j}\ \textit{x2j}\ \textit{x1k}\ \textit{x2k}.$
$(x,\ y) \in \textit{twl-st-heur}'''\ r \Longrightarrow$
$(\textit{xa},\ x') \in ?R \Longrightarrow$
$\textit{x2c} = (\textit{x1d},\ \textit{x2d}) \Longrightarrow$
$\textit{x2b} = (\textit{x1c},\ \textit{x2c}) \Longrightarrow$
$\textit{x2a} = (\textit{x1b},\ \textit{x2b}) \Longrightarrow$
$\textit{x2} = (\textit{x1a},\ \textit{x2a}) \Longrightarrow$
$x' = (\textit{x1},\ \textit{x2}) \Longrightarrow$
$\textit{x2h} = (\textit{x1i},\ \textit{x2i}) \Longrightarrow$
$\textit{x2g} = (\textit{x1h},\ \textit{x2h}) \Longrightarrow$
$\textit{x2f} = (\textit{x1g},\ \textit{x2g}) \Longrightarrow$
$\textit{x2e} = (\textit{x1f},\ \textit{x2f}) \Longrightarrow$
$\textit{xa} = (\textit{x1e},\ \textit{x2e}) \Longrightarrow$
$\neg\ \textit{x1a} \Longrightarrow$
$\neg\ \textit{x1f} \wedge \neg\ \textit{x1e} \Longrightarrow$
$\textit{isasat-fast}\ \textit{x1g} \Longrightarrow$
$(S,\ \textit{Sa})$
$\in \textit{twl-st-heur}''\ (\textit{dom-m}\ (\textit{get-clauses-wl}\ \textit{x1b}))\ (\textit{length}\ (\textit{get-clauses-wl-heur}\ \textit{x1g}))$
$(\textit{get-learned-count}\ \textit{x1g}) \Longrightarrow$
$(T,\ \textit{Ta})$
$\in \textit{twl-st-heur}''\ (\textit{dom-m}\ (\textit{get-clauses-wl}\ \textit{Sa}))\ (\textit{length}\ (\textit{get-clauses-wl-heur}\ S))\ (\textit{get-learned-count}\ S)$
$\Longrightarrow$
$\textit{isasat-fast}\ T \Longrightarrow$
$(\textit{xb},\ x'a)$
$\in \textit{bool-rel} \times_f$
$\textit{twl-st-heur}''''\textit{uu}\ (\textit{length}\ (\textit{get-clauses-wl-heur}\ S) + 3 + 1 + \textit{unat32-max div}\ 2)$
$(\textit{Suc}\ (\textit{clss-size-allcount}\ (\textit{get-learned-count}\ S))) \Longrightarrow$
$x'a = (\textit{x1j},\ \textit{x2j}) \Longrightarrow$
$\textit{xb} = (\textit{x1k},\ \textit{x2k}) \Longrightarrow$
$\textit{isasat-fast-relaxed2}\ \textit{x2k}\ \textit{x2i} \Longrightarrow$
$(((((\textit{x2k},\ \textit{x1h}),\ \textit{x1i}),\ \textit{x2i}),\ \textit{x1k}),\ (((\textit{x2j},\ \textit{x1c}),\ \textit{x1d}),\ \textit{x2d}),\ \textit{x1j})$
$\in \textit{twl-st-heur}''''u\ (\textit{length}\ (\textit{get-clauses-wl-heur}\ \textit{x2k}))\ (\textit{learned-clss-count}\ \textit{x2k}) \times_f$
$\textit{nat-rel} \times_f$
$\textit{nat-rel} \times_f$
$\textit{nat-rel} \times_f$
$\textit{bool-rel}\rangle$
by *simp*

have *H5*:
‹$(\textit{xc},\ x'b) \in \textit{restart-prog-wl-heur-rel}\ (\textit{length}\ (\textit{get-clauses-wl-heur}\ \textit{x2k}))\ (\textit{learned-clss-count}\ \textit{x2k}) \Longrightarrow$
$\textit{x2m} = (\textit{x1n},\ \textit{x2n}) \Longrightarrow$
$\textit{x2l} = (\textit{x1m},\ \textit{x2m}) \Longrightarrow$
$x'b = (\textit{x1l},\ \textit{x2l}) \Longrightarrow$
$\textit{x2p} = (\textit{x1q},\ \textit{x2q}) \Longrightarrow$
$\textit{x2o} = (\textit{x1p},\ \textit{x2p}) \Longrightarrow$
$\textit{xc} = (\textit{x1o},\ \textit{x2o}) \Longrightarrow$
$(\textit{x1o},\ \textit{x1l})$
$\in \textit{twl-st-heur-loop}''''\textit{uu}\ \ (\textit{length}\ (\textit{get-clauses-wl-heur}\ \textit{x2k}))\ (\textit{learned-clss-count}\ \textit{x2k})\rangle$
for *x2k x2m x1n x2n x2l x1m x'b x1l x2p x1q x2q x2o x1o xc x1p*

by *auto*

show *?thesis*
supply[[*goals-limit=1*]] *isasat-fast-length-leD*[*dest*] *twl-st-heur'-def*[*simp*] *learned-clss-count-twl-st-heur*[*simp*]
unfolding *cdcl-twl-stgy-restart-prog-bounded-wl-heur-alt-def*
cdcl-twl-stgy-restart-prog-bounded-wl-alt-def
apply (*intro frefI nres-relI*)

apply (*refine-rcg*
restart-prog-wl-D-heur-restart-prog-wl-D[*THEN fref-to-Down-curry4*]
cdcl-twl-o-prog-wl-D-heur-cdcl-twl-o-prog-wl-D[*THEN fref-to-Down*]
unit-propagation-outer-loop-wl-D-heur-unit-propagation-outer-loop-wl-D'[*THEN fref-to-Down*]
update-all-phases-Pair
convert-to-full-state-wl-heur
WHILEIT-refine[**where** *R* = ‹*?R*›])
subgoal using *r* **by** (*auto simp*: *snat64-max-def isasat-fast-def unat32-max-def*
dest: *twl-st-heur-twl-st-heur-loopD*)
subgoal for *x y ebrk ebrka xa x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d*
using *cdcl-twl-stgy-restart-abs-wl-inv-NoneD*[*of y* ‹*fst* (*snd* (*snd x'*))›
‹*fst* (*snd* (*snd* (*snd x'*)))› ‹*fst* (*snd* (*snd* (*snd* (*snd x'*))))› ‹*snd* (*snd* (*snd* (*snd* (*snd x'*))))›]
unfolding *cdcl-twl-stgy-restart-abs-wl-heur-inv-def prod.case prod-rel-fst-snd-iff*
apply (*rule-tac x=y* **in** *exI*)
apply (*rule-tac x=* ‹*fst* (*snd* (*snd x'*))› **in** *exI*)
apply (*case-tac xa*; *case-tac x'*)
by (*auto intro*!: *twl-st-heur-twl-st-heur-loopD*)
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *snat64-max-def isasat-fast-def unat32-max-def*)
subgoal by *auto*
subgoal by *fast*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto dest*!: *cdcl-twl-stgy-restart-abs-wl-inv-NoneD*)
apply (*rule twl-st-heur''*)
subgoal by *simp*
subgoal by (*auto dest*: *get-learned-count-learned-clss-countD simp*: *isasat-fast-def*)
apply (*rule twl-st-heur'''*; *assumption*)
subgoal by (*auto simp*: *isasat-fast-def unat64-max-def unat32-max-def snat64-max-def*
isasat-fast-relaxed-def isasat-fast-relaxed2-def)
apply (*rule H4*; *assumption*)
apply (*rule H5*; *assumption*)
subgoal
by (*auto simp*: *isasat-fast-def unat64-max-def unat32-max-def snat64-max-def*
isasat-fast-relaxed-def)
subgoal for *x y ebrk ebrka xa x' x1 x2 x1a x2a x1b x2b x1c x2c x1d x2d x1e x2e x1f x2f x1g x2g x1h x2h x1i x2i S Sa*
T Ta xb x'a x1j x2j x1k x2k xc x'b x1l x2l x1m x2m x1n x2n x1o x2o x1p x2p x1q x2q Tb Tc
unfolding *get-conflict-wl-is-None*
apply (*subst get-conflict-wl-is-None-heur-get-conflict-wl-is-None*[*THEN fref-to-Down-unRET-Id*, *of - Tc*])
apply *fast*
by (*auto simp*: *isasat-fast-def unat64-max-def unat32-max-def snat64-max-def*
dest!: *twl-st-heur-twl-st-heur-loopD*)
subgoal by *auto*
done

qed

end
theory *IsaSAT-Garbage-Collect-LLVM*
imports *IsaSAT-Restart-Heuristics-Defs IsaSAT-Setup-LLVM*
IsaSAT-VMTF-State-LLVM IsaSAT-Rephase-State-LLVM
IsaSAT-Arena-Sorting-LLVM
IsaSAT-Show-LLVM
begin

sepref-register *length-ll extra-information-mark-to-delete nth-rll*
LEARNED

lemma *isasat-GC-clauses-prog-copy-wl-entry-alt-def*:
‹*isasat-GC-clauses-prog-copy-wl-entry* = (λ*N0 W A* (*N'*, *aivdom*). *do* {
ASSERT(*nat-of-lit A* < *length W*);
ASSERT(*length* (*W* ! *nat-of-lit A*) ≤ *length N0*);
let le = *length* (*W* ! *nat-of-lit A*);
(*i*, *N*, *N'*, *aivdom*) ← *WHILE*$_T$
(λ(*i*, *N*, *N'*, *aivdom*). *i* < *le*)
(λ(*i*, *N*, (*N'*, *aivdom*)). *do* {
ASSERT(*i* < *length* (*W* ! *nat-of-lit A*));
let (*C*, -, -) = *W* ! *nat-of-lit A* ! *i*;
ASSERT(*arena-is-valid-clause-vdom N C*);
let st = *arena-status N C*;
if st ≠ *DELETED then do* {
ASSERT(*arena-is-valid-clause-idx N C*);
ASSERT(*length N'* +
(*if arena-length N C* > *4 then MAX-HEADER-SIZE else MIN-HEADER-SIZE*) +
arena-length N C ≤ *length N0*);
ASSERT(*length N* = *length N0*);
ASSERT(*length* (*get-vdom-aivdom aivdom*) < *length N0*);
ASSERT(*length* (*get-avdom-aivdom aivdom*) < *length N0*);
ASSERT(*length* (*get-ivdom-aivdom aivdom*) < *length N0*);
ASSERT(*length* (*get-tvdom-aivdom aivdom*) < *length N0*);
let D = *length N'* + (*if arena-length N C* > *4 then MAX-HEADER-SIZE else MIN-HEADER-SIZE*);
N' ← *fm-mv-clause-to-new-arena C N N'*;
ASSERT(*mark-garbage-pre* (*N*, *C*));
RETURN (*i*+*1*, *extra-information-mark-to-delete N C*, *N'*,
(*if st* = *LEARNED then add-learned-clause-aivdom-strong D aivdom else add-init-clause-aivdom-strong D aivdom*))
} *else RETURN* (*i*+*1*, *N*, (*N'*, *aivdom*))
}) (*0*, *N0*, (*N'*, *aivdom*));
RETURN (*N*, (*N'*, *aivdom*))
})›
proof −
have *H*: ‹(*let* (*a*, -, -) = *c in f a*) = (*let a* = *fst c in f a*)› **for** *a c f*
by (*cases c*) (*auto simp*: *Let-def*)
show *?thesis*
unfolding *isasat-GC-clauses-prog-copy-wl-entry-def H*
..
qed

sepref-def *isasat-GC-clauses-prog-copy-wl-entry-code*
is ‹*uncurry3 isasat-GC-clauses-prog-copy-wl-entry*›
:: ‹[λ(((*N*, -), -), -). *length N* ≤ *snat64-max*]$_a$

```
    arena-fast-assn^d *_a watchlist-fast-assn^k *_a unat-lit-assn^k *_a
      (arena-fast-assn ×_a aivdom-assn)^d →
    (arena-fast-assn ×_a (arena-fast-assn ×_a aivdom-assn))›
  supply [[goals-limit=1]] if-splits[split] length-ll-def[simp]
  unfolding isasat-GC-clauses-prog-copy-wl-entry-alt-def nth-rll-def[symmetric]
    length-ll-def[symmetric] if-conn(4)
  apply (annot-snat-const ‹TYPE(64)›)
  by sepref

sepref-register isasat-GC-clauses-prog-copy-wl-entry

lemma shorten-taken-op-list-list-take:
  ‹W[L := []] = op-list-list-take W L 0›
  by (auto simp:)

sepref-def isasat-GC-clauses-prog-single-wl-code
  is ‹uncurry3 isasat-GC-clauses-prog-single-wl›
  :: ‹[λ(((N, -), -), A). A ≤ unat32-max div 2 ∧ length N ≤ snat64-max]_a
    arena-fast-assn^d *_a (arena-fast-assn ×_a aivdom-assn)^d *_a watchlist-fast-assn^d *_a atom-assn^k →
    (arena-fast-assn ×_a (arena-fast-assn ×_a aivdom-assn) ×_a watchlist-fast-assn)›
  supply [[goals-limit=1]]
  unfolding isasat-GC-clauses-prog-single-wl-def
    shorten-taken-op-list-list-take
  apply (annot-snat-const ‹TYPE(64)›)
  by sepref

sepref-register isasat-GC-clauses-prog-wl2 isasat-GC-clauses-prog-single-wl
sepref-def isasat-GC-clauses-prog-wl2-code
  is ‹uncurry isasat-GC-clauses-prog-wl2›
  :: ‹[λ(-, (N, -)). length N ≤ snat64-max]_a
    (heuristic-bump-assn)^k *_a
    (arena-fast-assn ×_a (arena-fast-assn ×_a aivdom-assn) ×_a watchlist-fast-assn)^d →
    (arena-fast-assn ×_a (arena-fast-assn ×_a aivdom-assn) ×_a watchlist-fast-assn)›
  supply [[goals-limit=1]]
  unfolding isasat-GC-clauses-prog-wl2-def prod.case atom.fold-option
  by sepref

lemma isasat-GC-clauses-prog-wl-alt-def:
  ‹isasat-GC-clauses-prog-wl = (λS_0. do {
    let (vm, S) = extract-vmtf-wl-heur S_0;
    let (N', S) = extract-arena-wl-heur S;
    ASSERT (N' = get-clauses-wl-heur S_0);
    let (W', S) = extract-watchlist-wl-heur S;
    let (vdom, S) = extract-vdom-wl-heur S;
    let (old-arena, S) = extract-old-arena-wl-heur S;
    ASSERT(old-arena = []);
    (N, (N', vdom), WS) ← isasat-GC-clauses-prog-wl2 vm
      (N', (old-arena, empty-aivdom vdom), W');
    let S = update-watchlist-wl-heur WS S;
    let S = update-arena-wl-heur N' S;
    let S = update-old-arena-wl-heur (take 0 N) S;
    let S = update-vmtf-wl-heur vm S;
    let (stats, S) = extract-stats-wl-heur S;
    let S = update-stats-wl-heur (incr-GC stats) S;
    let S = update-vdom-wl-heur vdom S;
    let (heur, S) = extract-heur-wl-heur S;
```

let heur = heuristic-reluctant-untrigger (set-zero-wasted heur);
let S = update-heur-wl-heur heur S;
RETURN S
})›
by (*auto simp*: *isasat-GC-clauses-prog-wl-def state-extractors*
Let-def intro!: *ext bind-cong*[*OF refl*]
split: *isasat-int-splits*)

sepref-register *isasat-GC-clauses-prog-wl rewatch-heur-st*
sepref-def *isasat-GC-clauses-prog-wl-code*
is ‹*isasat-GC-clauses-prog-wl*›
:: ‹[λ*S*. *length* (*get-clauses-wl-heur S*) ≤ *snat64-max*]$_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
supply [[*goals-limit=1*]]
unfolding *isasat-GC-clauses-prog-wl-alt-def*
atom.fold-option
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *rewatch-heur-st-pre-alt-def*:
‹*rewatch-heur-st-pre S* ⟷ (∀ *i* ∈ *set* (*get-tvdom S*). *i* ≤ *snat64-max*)›
by (*auto simp*: *rewatch-heur-st-pre-def all-set-conv-nth*)

definition *rewatch-heur-and-reorder-vdom* **where**
‹*rewatch-heur-and-reorder-vdom vdom* = *rewatch-heur-and-reorder* (*get-tvdom-aivdom vdom*)›

sepref-def *rewatch-heur-and-reorder-code*
is ‹*uncurry2* (*rewatch-heur-and-reorder-vdom*)›
:: ‹[λ((*vdom*, *arena*), *W*). (∀ *x* ∈ *set* (*get-tvdom-aivdom vdom*). *x* ≤ *snat64-max*) ∧ *length arena* ≤ *snat64-max* ∧
length (*get-tvdom-aivdom vdom*) ≤ *snat64-max*]$_a$
aivdom-assn^k *$*_a$ *arena-fast-assn*^{k} *$*_a$ *watchlist-fast-assn*^{d} → *watchlist-fast-assn*›
supply [[*goals-limit=1*]]
arena-lit-pre-le-snat64-max[*dest*] *arena-is-valid-clause-idx-le-unat64-max*[*dest*]
supply [*simp*] = *append-ll-def*
supply [*dest*] = *arena-lit-implI*(*1*)
unfolding *rewatch-heur-and-reorder-def Let-def PR-CONST-def rewatch-heur-and-reorder-vdom-def*
rewatch-heur-vdom-def[*symmetric*]
by *sepref*

lemma *rewatch-heur-and-reorder-st-alt-def*:
‹*rewatch-heur-and-reorder-st* = (λS_0. *do* {
let (*vdom*, *S*) = *extract-vdom-wl-heur* S_0;
ASSERT (*vdom* = *get-aivdom* S_0);
let (*N*, *S*) = *extract-arena-wl-heur S*;
ASSERT (*N* = *get-clauses-wl-heur* S_0);
let (*W*, *S*) = *extract-watchlist-wl-heur S*;
ASSERT(*length* (*get-tvdom-aivdom vdom*) ≤ *length N*);
W ← *rewatch-heur-and-reorder* (*get-tvdom-aivdom vdom*) *N W*;
RETURN (*update-watchlist-wl-heur W* (*update-arena-wl-heur N* (*update-vdom-wl-heur vdom S*)))
})›
by (*auto simp*: *rewatch-heur-and-reorder-st-def state-extractors split*: *isasat-int-splits intro*!: *ext*)

sepref-register *rewatch-heur-and-reorder rewatch-heur-and-reorder-vdom*
sepref-def *rewatch-heur-st-code*
is ‹*rewatch-heur-and-reorder-st*›

:: ‹[λS. rewatch-heur-st-pre S ∧ length (get-clauses-wl-heur S) ≤ snat64-max]$_a$ isasat-bounded-assn^d → isasat-bounded-assn›
supply [[goals-limit=1]] append-ll-def[simp]
unfolding rewatch-heur-and-reorder-st-alt-def rewatch-heur-st-pre-alt-def rewatch-heur-vdom-def[symmetric] rewatch-heur-and-reorder-vdom-def[symmetric]
by sepref

sepref-register isasat-GC-clauses-wl-D

sepref-register rewatch-heur-and-reorder-st
sepref-def isasat-GC-clauses-wl-D-code
is ‹uncurry isasat-GC-clauses-wl-D›
:: ‹[λ(b, S). length (get-clauses-wl-heur S) ≤ snat64-max]$_a$ bool1-assn^k *$_a$ isasat-bounded-assn^d → isasat-bounded-assn›
supply [[goals-limit=1]] isasat-GC-clauses-wl-D-rewatch-pre[intro!]
unfolding isasat-GC-clauses-wl-D-def
by sepref

sepref-register access-avdom-at

experiment
begin
export-llvm
isasat-GC-clauses-wl-D-code
end

end
theory IsaSAT-Restart-Simp-LLVM
imports IsaSAT-Restart-Heuristics-LLVM IsaSAT-Garbage-Collect-LLVM
IsaSAT-Other-LLVM IsaSAT-Propagate-Conflict-LLVM IsaSAT-Inprocessing-LLVM IsaSAT-Restart-LLVM
begin

sepref-register cdcl-twl-mark-clauses-to-delete mark-to-delete-clauses-GC-wl-D-heur

sepref-def cdcl-twl-restart-wl-heur-fast-code
is ‹cdcl-twl-restart-wl-heur›
:: ‹[λS. length (get-clauses-wl-heur S) ≤ snat64-max]$_a$ isasat-bounded-assn^d → isasat-bounded-assn›
unfolding cdcl-twl-restart-wl-heur-def
supply [[goals-limit = 1]]
by sepref

sepref-def cdcl-twl-mark-clauses-to-delete-fast-code
is ‹cdcl-twl-mark-clauses-to-delete›
:: ‹[λS. length (get-clauses-wl-heur S) ≤ snat64-max]$_a$ isasat-bounded-assn^d → isasat-bounded-assn›
unfolding cdcl-twl-mark-clauses-to-delete-def
supply [[goals-limit = 1]]
by sepref

sepref-def cdcl-twl-full-restart-wl-D-GC-heur-prog-fast-code
is ‹cdcl-twl-full-restart-wl-D-GC-heur-prog›
:: ‹[λS. length (get-clauses-wl-heur S) ≤ snat64-max ∧ learned-clss-count S ≤ unat64-max]$_a$ isasat-bounded-assn^d → isasat-bounded-assn›
supply [[goals-limit = 1]]
unfolding cdcl-twl-full-restart-wl-D-GC-heur-prog-def
apply (annot-unat-const ‹TYPE(32)›)

by *sepref*

sepref-def *cdcl-twl-full-restart-wl-D-inprocess-heur-prog-fast-code*
is ‹*cdcl-twl-full-restart-wl-D-inprocess-heur-prog*›
:: ‹$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq snat64\text{-}max \wedge learned\text{-}clss\text{-}count\ S \leq unat64\text{-}max]_a$
$isasat\text{-}bounded\text{-}assn^d \to isasat\text{-}bounded\text{-}assn$›
supply [[*goals-limit* = *1*]]
supply [*simp*] = *isasat-fast-relaxed-def*
unfolding *cdcl-twl-full-restart-wl-D-inprocess-heur-prog-def*
apply (*annot-unat-const* ‹*TYPE*(*32*)›)
by *sepref*

sepref-register *restart-required-heur cdcl-twl-restart-wl-heur*
cdcl-twl-full-restart-wl-D-inprocess-heur-prog

sepref-def *restart-prog-wl-D-heur-fast-code*
is ‹*uncurry4 restart-prog-wl-D-heur*›
:: ‹$[\lambda((((S,\ -),\ -),\ n),\ -).\ isasat\text{-}fast\text{-}relaxed2\ S\ n]_a$
$isasat\text{-}bounded\text{-}assn^d *_a uint64\text{-}nat\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a$
$bool1\text{-}assn^k \to isasat\text{-}bounded\text{-}assn \times_a uint64\text{-}nat\text{-}assn \times_a uint64\text{-}nat\text{-}assn \times_a uint64\text{-}nat\text{-}assn$›
unfolding *restart-prog-wl-D-heur-def isasat-fast-relaxed-def isasat-fast-relaxed2-def*
supply [[*goals-limit* = *1*]]
apply (*annot-unat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *cdcl-twl-stgy-restart-prog-bounded-wl-heurI1*:
assumes
‹*isasat-fast a1'b*›
‹*learned-clss-count a2'e* ≤ *Suc* (*learned-clss-count a1'b*)›
shows ‹*learned-clss-count a2'e* ≤ *unat64-max*›
using *assms*
by (*auto simp*: *isasat-fast-def*)

lemma *cdcl-twl-stgy-restart-prog-bounded-wl-heurI2*:
‹*isasat-fast x* ⟹ *learned-clss-count x* ≤ *unat64-max*›
by (*auto simp*: *isasat-fast-def*)

lemma *cdcl-twl-stgy-restart-prog-bounded-wl-heurI3*:
assumes
‹*isasat-fast S*›
‹*length* (*get-clauses-wl-heur T*) ≤ *length* (*get-clauses-wl-heur S*)›
‹*learned-clss-count T* ≤ (*learned-clss-count S*)›
shows ‹*isasat-fast T*›
using *assms*
by (*auto simp*: *isasat-fast-def*)

sepref-register *cdcl-twl-stgy-restart-prog-bounded-wl-heur*
sepref-def *cdcl-twl-stgy-restart-prog-wl-heur-fast-code*
is ‹*cdcl-twl-stgy-restart-prog-bounded-wl-heur*›
:: ‹$[\lambda S.\ isasat\text{-}fast\ S]_a\ isasat\text{-}bounded\text{-}assn^d \to bool1\text{-}assn \times_a isasat\text{-}bounded\text{-}assn$›
unfolding *cdcl-twl-stgy-restart-prog-bounded-wl-heur-def*
supply [[*goals-limit* = *1*]] *isasat-fast-countD*[*dest*]
supply [*intro*] = *cdcl-twl-stgy-restart-prog-bounded-wl-heurI2*
supply [*sepref-bounds-simps del*] = *unat32-max-def snat32-max-def unat64-max-def snat64-max-def*
apply (*annot-unat-const* ‹*TYPE*(*64*)›)

by *sepref*

experiment
begin
export-llvm *opts-reduction-st-fast-code*
opts-restart-st-fast-code
get-conflict-count-since-last-restart-heur-fast-code
get-fast-ema-heur-fast-code
get-slow-ema-heur-fast-code
get-learned-count-fast-code
upper-restart-bound-reached-fast-impl
minimum-number-between-restarts-impl
restart-required-heur-fast-code
cdcl-twl-full-restart-wl-D-GC-heur-prog-fast-code
cdcl-twl-restart-wl-heur-fast-code
cdcl-twl-mark-clauses-to-delete-fast-code
cdcl-twl-local-restart-wl-D-heur-fast-code
get-conflict-wl-is-None-fast-code
end

end
theory *IsaSAT-Defs*
imports *IsaSAT-CDCL-Defs IsaSAT-Restart-Simp-Defs IsaSAT-Initialisation*
begin

Chapter 24

Full IsaSAT

We now combine all the previous definitions to prove correctness of the complete SAT solver:

1. We initialise the arena part of the state;
2. Then depending on the options and the number of clauses, we either use the bounded version or the unbounded version. Once have if decided which one, we initiale the watch lists;
3. After that, we can run the CDCL part of the SAT solver;
4. Finally, we extract the trail from the state.

   Remark that the statistics and the options are unchecked: the number of propagations might overflows (but they do not impact the correctness of the whole solver). Similar restriction applies on the options: setting the options might not do what you expect to happen, but the result will still be correct.

definition *extract-model-of-state-heur* **where**
‹*extract-model-of-state-heur U* = *Some* (*fst* (*get-trail-wl-heur U*))›

definition *convert-state* **where**
‹*convert-state* - *S* = *S*›

definition *learned-clss-count-init* :: ‹*twl-st-wl-heur-init* ⇒ *nat*› **where**
‹*learned-clss-count-init S* = *clss-size-lcount* (*get-learned-count-init S*) +
clss-size-lcountUE (*get-learned-count-init S*) +
clss-size-lcountUEk (*get-learned-count-init S*) + *clss-size-lcountUS* (*get-learned-count-init S*) +
clss-size-lcountU0 (*get-learned-count-init S*)›

definition *isasat-fast-init* :: ‹*twl-st-wl-heur-init* ⇒ *bool*› **where**
‹*isasat-fast-init S* ⟷
(*length* (*get-clauses-wl-heur-init S*) ≤ *snat64-max* − (*unat32-max div 2* + *MAX-HEADER-SIZE*+*1*)
∧
learned-clss-count-init S < *unat64-max*)›

definition *empty-init-code* :: ‹*bool* × - *list* × *isasat-stats*› **where**
‹*empty-init-code* = (*True*, [], *empty-stats*)›

definition *empty-conflict-code* :: ‹(*bool* × - *list* × *isasat-stats*) *nres*› **where**

‹*empty-conflict-code* = *do*{
let M0 = [];
RETURN (*False*, *M0*, *empty-stats*)}›

definition *extract-model-of-state-stat* :: ‹*isasat* ⇒ *bool* × *nat literal list* × *isasat-stats*› **where**
‹*extract-model-of-state-stat U* =
(*False*, (*fst* (*get-trail-wl-heur U*)), (*get-stats-heur U*))›

definition *extract-state-stat* :: ‹*isasat* ⇒ *bool* × *nat literal list* × *isasat-stats*› **where**
‹*extract-state-stat U* =
(*True*, [], (*get-stats-heur U*))›

definition *empty-conflict* :: ‹*nat literal list option*› **where**
‹*empty-conflict* = *Some* []›

definition *IsaSAT-bounded-heur* :: ‹*opts* ⇒ *nat clause-l list* ⇒ (*bool* × (*bool* × *nat literal list* × *isasat-stats*)) *nres*› **where**
‹*IsaSAT-bounded-heur opts CS* = *do*{
- ← *RETURN* (*IsaSAT-Profile.start-initialisation*);
ASSERT(*isasat-input-bounded* (*mset-set* (*extract-atms-clss CS* {})));
ASSERT(∀ *C*∈*set CS*. ∀ *L*∈*set C*. *nat-of-lit L* ≤ *unat32-max*);
let $\mathcal{A}_{in}'$ = *mset-set* (*extract-atms-clss CS* {});
ASSERT(*isasat-input-bounded* $\mathcal{A}_{in}'$);
ASSERT(*distinct-mset* $\mathcal{A}_{in}'$);
let $\mathcal{A}_{in}''$ = *virtual-copy* $\mathcal{A}_{in}'$;
let b = *opts-unbounded-mode opts*;
S ← *init-state-wl-heur-fast* $\mathcal{A}_{in}'$;
(*T*::*twl-st-wl-heur-init*) ← *init-dt-wl-heur-b CS S*;
let T = *convert-state* $\mathcal{A}_{in}''$ *T*;
- ← *RETURN* (*IsaSAT-Profile.stop-initialisation*);
if isasat-fast-init T ∧ ¬*is-failed-heur-init T*
then do {
if ¬*get-conflict-wl-is-None-heur-init T*
then RETURN (*False*, *empty-init-code*)
else if CS = [] *then do* {*stat* ← *empty-conflict-code*; *RETURN* (*False*, *stat*)}
else do {
- ← *RETURN* (*IsaSAT-Profile.start-initialisation*);
ASSERT($\mathcal{A}_{in}'' \neq \{\#\}$);
ASSERT(*isasat-input-bounded-nempty* $\mathcal{A}_{in}''$);
- ← *isasat-information-banner T*;
ASSERT(*rewatch-heur-st-fast-pre T*);
T ← *rewatch-heur-st-init T*;
ASSERT(*isasat-fast-init T*);
T ← *finalise-init-code opts* (*T*::*twl-st-wl-heur-init*);
- ← *RETURN* (*IsaSAT-Profile.stop-initialisation*);
ASSERT(*isasat-fast T*);
(*b*, *U*) ← *cdcl-twl-stgy-restart-prog-bounded-wl-heur T*;
RETURN (*b*, *if* ¬*b* ∧ *get-conflict-wl-is-None-heur U then extract-model-of-state-stat U*
else extract-state-stat U)
}
}
else RETURN (*True*, *empty-init-code*)
}›

end

theory *IsaSAT*
imports *IsaSAT-CDCL IsaSAT-Restart-Simp IsaSAT-Initialisation*
IsaSAT-Defs
begin

24.1 Correctness Relation

We cannot use *cdcl-twl-stgy-restart* since we do not always end in a final state for *cdcl-twl-stgy*.

abbreviation *conclusive-TWL-run-bounded* **where**
‹*conclusive-TWL-run-bounded S* ≡ *partial-conclusive-TWL-run S*›

Before introducing the pragmatic CDCL version, we expressed the specification all the level up to Weidenbach's CDCL, but now we stop at the pragmatic CDCL. Technically, we could actually skip the part on the calculus and simply use the conclusive part (no conflict and model, conflict and unsat), but this version is nicer on paper to highlight the refinement approach.

definition *conclusive-CDCL-state* :: ‹*'v clauses* ⇒ *'v prag-st* ⇒ *'v prag-st* ⇒ *bool*› **where**
‹*conclusive-CDCL-state CS T U* ⟷
(*pget-conflict U* = *None* ⟶ (*consistent-interp* (*lits-of-l* (*pget-trail U*)) ∧
pget-trail U ⊨*as* (*set-mset CS*))) ∧
(*pget-conflict U* ≠ *None* ⟶ (*CS* ≠ {#} ∧ *count-decided* (*pget-trail U*) = *0* ∧
unsatisfiable (*set-mset CS*)))›

lemmas *cdcl-twl-stgy-restart-restart-prog-spec* = *cdcl-twl-stgy-restart-prog-spec*

lemmas *cdcl-twl-stgy-restart-prog-bounded-spec* = *cdcl-twl-stgy-prog-bounded-spec*

lemma *rtranclp-pcdcl-satisfiable-iff*:
assumes
‹*pcdcl*** *S T*› **and**
‹*pcdcl-all-struct-invs S*› **and**
ent-init: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-learned-clauses-entailed-by-init* (*state-of S*)›
shows
‹*satisfiable* (*set-mset* (*pget-all-init-clss S*)) ⟷ *satisfiable* (*set-mset* (*pget-all-init-clss T*))›
using *assms*
apply (*induction rule*: *rtranclp-induct*)
subgoal by *auto*
subgoal
by (*auto dest*!: *pcdcl-satisfiable-iff intro*: *rtranclp-pcdcl-all-struct-invs*
Pragmatic-CDCL.rtranclp-pcdcl-entailed-by-init)
done

lemma *pcdcl-stgy-restart-satisfiable-iff*:
‹*pcdcl-stgy-restart* (*R1*, *R2*, *S*, *m*, *n*, *g*) (*R1'*, *R2'*, *T*, *m'*, *n'*, *g'*) ⟹
pcdcl-all-struct-invs S ⟹ *cdcl*$_W$*-restart-mset.cdcl*$_W$*-learned-clauses-entailed-by-init* (*state-of S*) ⟹
satisfiable (*set-mset* (*pget-all-init-clss T*)) ⟷ *satisfiable* (*set-mset* (*pget-all-init-clss S*))›
apply (*induction* ‹(*R1*, *R2*, *S*, *m*, *n*, *g*)› ‹(*R1'*, *R2'*, *T*, *m'*, *n'*, *g'*)› *rule*: *pcdcl-stgy-restart.induct*)
apply (*auto dest*: *pcdcl-restart-only-entailed-iff pcdcl-restart-entailed-iff*
dest: *rtranclp-pcdcl-stgy-pcdcl pcdcl-tcore-stgy-pcdcl-stgy'*
rtranclp-pcdcl-stgy-pcdcl rtranclp-pcdcl-satisfiable-iff)[]
apply (*meson rtranclp-pcdcl-inprocessing-satisfiable-iff rtranclp-pcdcl-restart-inprocessing rtranclp-pcdcl-stgy-pcdcl rtranclp-trans*)
apply (*meson satisfiable-carac true-cls-mset-true-clss-iff*(*2*) *twl-restart-ops.pcdcl-restart-only-entailed-iff*)
by *simp*

lemma *rtranclp-pcdcl-restart-satisfiable-iff*:
‹*pcdcl-stgy-restart*** (*R1*, *R2*, *S*, *m*, *n*, *g*) (*R1'*, *R2'*, *T*, *m'*, *n'*, *g'*) ⟹
pcdcl-all-struct-invs S ⟹ *cdcl*$_W$*-restart-mset.cdcl*$_W$*-learned-clauses-entailed-by-init* (*state-of S*) ⟹
satisfiable (*set-mset* (*pget-all-init-clss T*)) ⟷ *satisfiable* (*set-mset* (*pget-all-init-clss S*))›
apply (*induction rule*: *rtranclp-induct*[*of r* ‹(-, -, -, -, -, -)› ‹(-, -, -, -, -, -)›, *split-format*(*complete*), *of* **for** *r*])
subgoal by *auto*
subgoal for *T U*
by (*simp add*: *pcdcl-stgy-restart-satisfiable-iff pcdcl-stgy-restart-same-init-vars*
rtranclp-pcdcl-stgy-restart-entailed-by-init rtranclp-pcdcl-stgy-restart-pcdcl-all-struct-invs)
done

fun *pinit-state* :: ‹*'v clauses* ⇒ *'v prag-st*› **where**
‹*pinit-state N* = ([], *N*, {#}, *None*, {#}, {#}, {#}, {#}, {#}, {#})›

lemma *get-all-clss-alt-def*:
‹*get-all-clss S* = *get-all-init-clss S* + *get-all-learned-clss S*›
by (*cases S*) (*auto simp*: *ac-simps clauses-def*)

lemma *conclusive-TWL-run-conclusive-CDCL-state*:
assumes
struct: ‹*twl-struct-invs S*› **and**
stgy: ‹*twl-stgy-invs S*› **and**
init: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-learned-clauses-entailed-by-init* (*state*$_W$*-of S*)›
shows ‹*conclusive-TWL-run S*
≤ ⇓ (*br pstate*$_W$*-of* (λ-. *True*))
(*SPEC*
(*conclusive-CDCL-state* (*get-all-init-clss S*) (*pstate*$_W$*-of S*)))›
unfolding *conclusive-TWL-run-def*
proof (*rule RES-refine*)
fix *s*
assume ‹*s* ∈ {*Ta*.
∃ *T0 T0' n' m*$_0$ *n*$_0$.
*cdcl-twl-stgy-restart*** (*S*, *S*, *S*, *m*$_0$, *n*$_0$, *True*) (*T0*, *T0'*, *Ta*, *n'*) ∧
final-twl-state Ta}›
then obtain *T0 T0' m' n' f' m*$_0$ *n*$_0$ **where**
st: ‹*cdcl-twl-stgy-restart*** (*S*, *S*, *S*, *m*$_0$, *n*$_0$, *True*) (*T0*, *T0'*, *s*, *m'*, *n'*, *f'*)› **and**
final: ‹*final-twl-state s*›
by *fast*
have *pcdcl-invs*: ‹*pcdcl-all-struct-invs* (*pstate*$_W$*-of S*)›
using *struct* **unfolding** *twl-struct-invs-def* **by** *auto*
from *rtranclp-cdcl-twl-stgy-restart-pcdcl*[*OF st struct struct struct init init init*]
have *pcdcl*: ‹*pcdcl-stgy-restart*** (*pstate*$_W$*-of S*, *pstate*$_W$*-of S*, *pstate*$_W$*-of S*, *m*$_0$, *n*$_0$, *True*)
(*pstate*$_W$*-of T0*, *pstate*$_W$*-of T0'*, *pstate*$_W$*-of s*, *m'*, *n'*, *f'*)› **.**
have *struct-invs-s*: ‹*twl-struct-invs s*›
using *rtranclp-cdcl-twl-stgy-restart-twl-struct-invs*[*OF st*] *struct init*
by *auto*
have *alien*: ‹*cdcl*$_W$*-restart-mset.no-strange-atm* (*state*$_W$*-of s*)›
using *struct-invs-s* **unfolding** *twl-struct-invs-def pcdcl-all-struct-invs-def*
cdcl$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-M-level-inv-def*
by *auto*
then have *atms-eq*: ‹*atms-of-mm* (*get-all-clss s*) = *atms-of-mm* (*pget-all-init-clss* (*pstate*$_W$*-of s*))›
unfolding *cdcl*$_W$*-restart-mset.no-strange-atm-def*
by (*cases s*) (*auto*)
have *pinvs*: ‹*pcdcl-all-struct-invs* (*pstate*$_W$*-of s*)›

using *struct-invs-s* **unfolding** *twl-struct-invs-def* **by** *auto*
have *stgy-invs-s*: ‹*twl-stgy-invs s*›
using *rtranclp-cdcl-twl-stgy-restart-twl-stgy-invs*[*OF st*] *stgy struct init*
by (*auto simp*: *twl-restart-inv-def pcdcl-stgy-restart-inv-def*
twl-struct-invs-def)
have *H1*: ‹*consistent-interp* (*lits-of-l* (*get-trail s*))›
‹*get-trail s* ⊨*asm get-all-init-clss S*›
if *confl*: ‹*get-conflict s* = *None*›
proof −
have ‹*no-step cdcl-twl-stgy s*›
using *confl final* **unfolding** *final-twl-state-def*
by *blast*
then have ‹*no-step pcdcl-core-stgy* (*pstate*$_W$ *-of s*)›
by (*rule no-step-cdcl-twl-stgy-no-step-cdcl*$_W$*-stgy*[*OF - struct-invs-s*])
then have *nss*: ‹*no-step cdcl*$_W$*-restart-mset.cdcl*$_W$*-stgy* (*state-of* (*pstate*$_W$ *-of s*))›
by (*rule no-step-pcdcl-core-stgy-is-cdcl-stgy*)
(*use struct-invs-s* **in** ‹*auto simp*: *twl-struct-invs-def pcdcl-all-struct-invs-def*›)
show ‹*consistent-interp* (*lits-of-l* (*get-trail s*))›
using *struct-invs-s* **unfolding** *twl-struct-invs-def pcdcl-all-struct-invs-def*
cdcl$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-M-level-inv-def*
by *auto*
moreover **{**
have *nss′*: ‹*no-step cdcl*$_W$*-restart-mset.cdcl*$_W$ (*state*$_W$ *-of s*)›
using *nss* **by** (*auto dest*: *cdcl*$_W$*-restart-mset.cdcl*$_W$*-Ex-cdcl*$_W$*-stgy*)
have ‹*total-over-set* (*lits-of-l* (*get-trail s*))
(*atms-of-mm* (*pget-all-init-clss* (*pstate*$_W$ *-of s*)))›
using *cdcl*$_W$*-restart-mset.no-step-cdcl*$_W$*-total*[*of* ‹(*state*$_W$ *-of s*)›, *OF nss′*]
confl struct-invs-s atms-eq **unfolding** *twl-struct-invs-def pcdcl-all-struct-invs-def*
cdcl$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def total-over-m-def*
by *auto*
}
ultimately show ‹*get-trail s* ⊨*asm get-all-init-clss S*›
using *struct-invs-s rtranclp-pcdcl-restart-entailed-iff*[*OF pcdcl*, *of* ‹*lits-of-l* (*get-trail s*)›]
cdcl$_W$*-restart-mset.cdcl*$_W$*-stgy-final-state-conclusive2*[*OF nss*] *confl init pcdcl-invs*
unfolding *twl-struct-invs-def pcdcl-all-struct-invs-def*
cdcl$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*
by (*auto simp*: *get-all-clss-alt-def true-annots-true-cls*)
qed

have *H2*: ‹*cdcl*$_W$*-restart-mset.clauses* (*state-of* (*pstate*$_W$ *-of s*)) ⊨*pm T*›
if ‹*conflicting* (*state*$_W$ *-of s*) = *Some T*›
for *T*
using *struct-invs-s that* **unfolding** *twl-struct-invs-def pcdcl-all-struct-invs-def*
cdcl$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*
cdcl$_W$*-restart-mset.cdcl*$_W$*-learned-clause-def*
by *auto*
then have *H3*: ‹*get-conflict s* ≠ *None* ⟹ *get-all-init-clss s* ≠ {#}›
using *rtranclp-pcdcl-stgy-restart-entailed-by-init*[*OF pcdcl pcdcl-invs*] *init*
alien
by (*force simp*: *get-all-clss-alt-def cdcl*$_W$*-restart-mset.no-strange-atm-def*
cdcl$_W$*-restart-mset.cdcl*$_W$*-learned-clauses-entailed-by-init-def empty-entails-novars-iff*)
have *H4*:
‹*unsatisfiable* (*set-mset* (*get-all-init-clss S*))›
if *confl*: ‹*get-conflict s* ≠ *None*› ‹*count-decided* (*get-trail s*) = *0*›
proof −
have ‹*unsatisfiable* (*set-mset* (*init-clss* (*state*$_W$ *-of s*)))›

```
    by (rule conflict-of-level-unsatisfiable[of ‹state_W-of s›])
     (use confl struct-invs-s init
       rtranclp-pcdcl-stgy-restart-entailed-by-init[OF pcdcl pcdcl-invs]
      in ‹auto simp: twl-struct-invs-def pcdcl-all-struct-invs-def
       cdcl_W-restart-mset.cdcl_W-all-struct-inv-def›)
   then show ‹unsatisfiable (set-mset (get-all-init-clss S))›
    using rtranclp-pcdcl-restart-satisfiable-iff[OF pcdcl]
     rtranclp-pcdcl-stgy-restart-same-init-vars[OF pcdcl] struct-invs-s init pcdcl-invs
    by (auto simp: satisfiable-def total-over-m-def twl-struct-invs-def)
 qed

 have H5: False
  if confl: ‹get-conflict s ≠ None› ‹count-decided (get-trail s) ≠ 0›
 proof −
  have ‹no-step cdcl-twl-stgy s›
   using confl final unfolding final-twl-state-def
   by auto

  then have ‹no-step pcdcl-core-stgy (pstate_W-of s)›
   by (rule no-step-cdcl-twl-stgy-no-step-cdcl_W-stgy[OF - struct-invs-s])
  then have nss: ‹no-step cdcl_W-restart-mset.cdcl_W-stgy (state-of (pstate_W-of s))›
   by (rule no-step-pcdcl-core-stgy-is-cdcl-stgy)
    (use struct-invs-s in ‹auto simp: twl-struct-invs-def pcdcl-all-struct-invs-def›)
  show False
   using struct-invs-s rtranclp-pcdcl-restart-entailed-iff[OF pcdcl, of ‹lits-of-l (get-trail s)›]
    cdcl_W-restart-mset.cdcl_W-stgy-final-state-conclusive2[OF nss] confl
    stgy-invs-s
   unfolding twl-struct-invs-def pcdcl-all-struct-invs-def
    cdcl_W-restart-mset.cdcl_W-all-struct-inv-def
    twl-stgy-invs-def cdcl_W-restart-mset.cdcl_W-stgy-invariant-def
    cdcl_W-restart-mset.conflict-non-zero-unless-level-0-def
   by (auto simp: get-all-clss-alt-def true-annots-true-cls)
 qed

 show ‹∃ s'∈Collect (conclusive-CDCL-state (get-all-init-clss S) (pstate_W-of S)).
   (s, s') ∈ (br pstate_W-of (λ-. True))›
   apply (rule-tac x = ‹pstate_W-of s› in bexI)
   apply (solves ‹auto simp: br-def›)
   unfolding conclusive-CDCL-state-def mem-Collect-eq
   apply (cases ‹count-decided (get-trail s) = 0›)
   apply (use H1 H2 H3 H4 H5 in force)+
   done
qed

definition init-state-l :: ‹'v twl-st-l-init› where
 ‹init-state-l = (([], fmempty, None, {#}, {#},{#}, {#}, {#}, {#}, {#}, {#}, {#}), {#})›

definition to-init-state-l :: ‹nat twl-st-l-init ⇒ nat twl-st-l-init› where
 ‹to-init-state-l S = S›

definition init-state0 :: ‹'v twl-st-init› where
 ‹init-state0 = (([], {#}, {#}, None, {#}, {#}, {#}, {#}, {#}, {#}, {#}, {#}), {#})›

definition to-init-state0 :: ‹nat twl-st-init ⇒ nat twl-st-init› where
 ‹to-init-state0 S = S›
```

lemma *init-dt-pre-init*:
shows ‹*init-dt-pre CS* (*to-init-state-l init-state-l*)›
unfolding *init-dt-pre-def to-init-state-l-def init-state-l-def*
by (*rule exI*[*of* - ‹(([], {#}, {#}, *None*, {#}, {#}, {#}, {#}, {#}, {#}, {#}, {#}), {#})›])
(*auto simp*: *twl-st-l-init-def twl-init-invs*)

This is the specification of the SAT solver:

definition *SAT* :: ‹*nat clauses* ⇒ *nat prag-st nres*› **where**
‹*SAT CS* = *do*{
let T = *pinit-state CS*;
SPEC (*conclusive-CDCL-state CS T*)
}›

definition *init-dt-spec0* :: ‹'*v clause-l list* ⇒ '*v twl-st-init* ⇒ '*v twl-st-init* ⇒ *bool*› **where**
‹*init-dt-spec0 CS SOC T'* ⟷
(
twl-struct-invs-init T' ∧
clauses-to-update-init T' = {#} ∧
(∀ *s*∈*set* (*get-trail-init T'*). ¬*is-decided s*) ∧
(*get-conflict-init T'* = *None* ⟶
literals-to-update-init T' = *uminus* ‘# *lit-of* ‘# *mset* (*get-trail-init T'*)) ∧
(*remdups-mset* ‘# *mset* ‘# *mset CS* + *clause* ‘# (*get-init-clauses-init SOC*) + *other-clauses-init SOC* +
get-unit-init-clauses-init SOC + *get-init-clauses0-init SOC* + *get-subsumed-init-clauses-init SOC* +
get-init-clauses0-init SOC =
clause ‘# (*get-init-clauses-init T'*) + *other-clauses-init T'* +
get-unit-init-clauses-init T' + *get-subsumed-init-clauses-init T'* + *get-init-clauses0-init T'*) ∧
get-learned-clauses0-init SOC = *get-learned-clauses0-init T'* ∧
get-learned-clauses-init SOC = *get-learned-clauses-init T'* ∧
get-subsumed-learned-clauses-init SOC = *get-subsumed-learned-clauses-init T'* ∧
get-learned-clauses0-init SOC = *get-learned-clauses0-init T'* ∧
get-unit-learned-clauses-init T' = *get-unit-learned-clauses-init SOC* ∧
twl-stgy-invs (*fst T'*) ∧
(*other-clauses-init T'* ≠ {#} ⟶ *get-conflict-init T'* ≠ *None*) ∧
({#} ∈# *mset* ‘# *mset CS* ⟶ *get-conflict-init T'* ≠ *None*) ∧
(*get-conflict-init SOC* ≠ *None* ⟶ *get-conflict-init SOC* = *get-conflict-init T'*))›

24.2 Refinements of the Whole SAT Solver

We do not add the refinement steps in separate files, since the form is very specific to the SAT solver we want to generate (and needs to be updated if it changes).

definition *SAT0* :: ‹*nat clause-l list* ⇒ *nat twl-st nres*› **where**
‹*SAT0 CS* = *do*{
b ← *SPEC*(λ-::*bool*. *True*);
if b then do {
let S = *init-state0*;
T ← *SPEC* (*init-dt-spec0 CS* (*to-init-state0 S*));
let T = *fst T*;
if get-conflict T ≠ *None*
then RETURN T
else if CS = [] *then RETURN* (*fst init-state0*)
else do {
ASSERT (*extract-atms-clss CS* {} ≠ {});

```
   ASSERT (clauses-to-update T = {#});
        ASSERT(clause `# (get-clauses T) + unit-clss T + subsumed-clauses T + get-all-clauses0 T =
           remdups-mset `# mset `# mset CS);
         ASSERT(get-learned-clss T = {#});
         ASSERT(subsumed-learned-clss T = {#});
         cdcl-twl-stgy-restart-prog T
       }
    }
    else do {
       let S = init-state0;
       T ← SPEC (init-dt-spec0 CS (to-init-state0 S));
       failed ← SPEC (λ- :: bool. True);
       if failed then do {
         T ← SPEC (init-dt-spec0 CS (to-init-state0 S));
         let T = fst T;
         if get-conflict T ≠ None
         then RETURN T
         else if CS = [] then RETURN (fst init-state0)
         else do {
           ASSERT (extract-atms-clss CS {} ≠ {});
           ASSERT (clauses-to-update T = {#});
           ASSERT(clause `# (get-clauses T) + unit-clss T + subsumed-clauses T + get-all-clauses0 T
=
             remdups-mset `# mset `# mset CS);
           ASSERT(get-learned-clss T = {#});
           cdcl-twl-stgy-restart-prog T
       }
       } else do {
         let T = fst T;
         if get-conflict T ≠ None
         then RETURN T
         else if CS = [] then RETURN (fst init-state0)
         else do {
           ASSERT (extract-atms-clss CS {} ≠ {});
           ASSERT (clauses-to-update T = {#});
           ASSERT(clause `# (get-clauses T) + unit-clss T + subsumed-clauses T + get-all-clauses0 T
=
             remdups-mset `# mset `# mset CS);
           ASSERT(get-learned-clss T = {#});
           cdcl-twl-stgy-restart-prog-early T
         }
       }
     }
  }›

lemma pget-all-init-clss-pstate_W-of-init:
  ‹pget-all-init-clss (pstate_W-of-init T) = get-subsumed-init-clauses-init T +
  get-unit-init-clauses-init T+
  get-init-clauses0-init T +
                  other-clauses-init T+
                   clause `# (get-init-clauses-init T)›
  by (cases T) auto


lemma [twl-st-init,simp]:
  ‹pget-trail (pstate_W-of-init T) = get-trail-init T›
```

‹pget-conflict (pstate$_W$-of-init T) = get-conflict-init T›
‹pget-all-learned-clss (pstate$_W$-of-init T) = clause `# get-learned-clauses-init T+ get-unit-learned-clauses-init T + get-learned-clauses0-init T +
get-subsumed-learned-clauses-init T›
‹get-init-learned-clss (fst T) = get-unit-learned-clauses-init T›
‹subsumed-learned-clauses (fst T) = get-subsumed-learned-clauses-init T›
‹get-learned-clss (fst T) = get-learned-clauses-init T›
‹get-conflict (fst T) = get-conflict-init T›
by (*solves* ‹*cases T*; *auto*›)+

lemma *get-all-init-clss-alt-init-def*:
‹*get-all-init-clss* (*fst T*) = *clauses* (*get-init-clauses-init T*) +
get-unit-init-clauses-init T + *get-subsumed-init-clauses-init T* + *get-init-clauses0-init T*›
by (*cases T*) (*auto simp*:)

lemma *satisfiable-remdups-iff*:
‹*satisfiable* ((λ*x*. *remdups-mset* (*mset x*)) ` *CS*) ⟷ *satisfiable* (*mset* ` *CS*)›
by (*auto simp flip*: *satisfiable-carac simp*: *true-clss-def*)

lemma *true-annots-remdups-mset*:
‹*I* ⊨*as remdups-mset* ` *C* ⟷ *I* ⊨*as C*›
by (*simp add*: *true-annot-def true-annots-def*)

lemma *SAT0-SAT*:
shows ‹*SAT0 CS* ≤ ⇓ {(*S*, *T*). *T* = *pstate*$_W$-*of S*} (*SAT* (*mset `# mset CS*))›
proof −
have *conflict-during-init*: ‹*RETURN* (*fst T*)
≤ ⇓ {(*S*, *T*). *T* = *pstate*$_W$-*of S*}
(*SPEC* (*conclusive-CDCL-state* (*mset `# mset CS*)
(*pinit-state* (*mset `# mset CS*))))›
if
spec: ‹*T* ∈ *Collect* (*init-dt-spec0 CS* (*to-init-state0 init-state0*))› **and**
confl: ‹*get-conflict* (*fst T*) ≠ *None*›
for *T*
proof −
let *?CS* = ‹*remdups-mset `# mset `# mset CS*›
have
struct-invs: ‹*twl-struct-invs-init T*› **and**
‹*clauses-to-update-init T* = {#}› **and**
count-dec: ‹∀ *s*∈*set* (*get-trail-init T*). ¬ *is-decided s*› **and**
‹*get-conflict-init T* = *None* ⟶
literals-to-update-init T =
uminus `# lit-of `# mset (*get-trail-init T*)› **and**
clss: ‹*?CS* +
clause `# get-init-clauses-init (*to-init-state0 init-state0*) +
other-clauses-init (*to-init-state0 init-state0*) +
get-unit-init-clauses-init (*to-init-state0 init-state0*) +
get-init-clauses0-init (*to-init-state0 init-state0*) +
get-subsumed-init-clauses-init (*to-init-state0 init-state0*) +
get-init-clauses0-init (*to-init-state0 init-state0*) =
clause `# get-init-clauses-init T + *other-clauses-init T* +
get-unit-init-clauses-init T + *get-subsumed-init-clauses-init T* +
get-init-clauses0-init T› **and**
learned: ‹*get-learned-clauses-init* (*to-init-state0 init-state0*) =
get-learned-clauses-init T›

```
    ‹get-unit-learned-clauses-init T =
      get-unit-learned-clauses-init (to-init-state0 init-state0)›
    ‹get-subsumed-learned-clauses-init T =
      get-subsumed-learned-clauses-init (to-init-state0 init-state0)›
    ‹get-learned-clauses0-init T =
      get-learned-clauses0-init (to-init-state0 init-state0)› and
  ‹twl-stgy-invs (fst T)› and
  ‹other-clauses-init T ≠ {#} ⟶ get-conflict-init T ≠ None› and
  ‹{#} ∈# mset `# mset CS ⟶ get-conflict-init T ≠ None› and
  ‹get-conflict-init (to-init-state0 init-state0) ≠ None ⟶
   get-conflict-init (to-init-state0 init-state0) = get-conflict-init T›
  using spec unfolding init-dt-wl-spec-def init-dt-spec0-def
    Set.mem-Collect-eq apply −
  apply normalize-goal+
  by metis+

have count-dec: ‹count-decided (get-trail (fst T)) = 0›
  using count-dec unfolding count-decided-0-iff by (auto simp: twl-st-init
    twl-st-wl-init)

have le: ‹cdcl_W-restart-mset.cdcl_W-learned-clause (state_W-of-init T)› and
  all-struct-invs:
    ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (state_W-of-init T)›
  using struct-invs unfolding twl-struct-invs-init-def
     cdcl_W-restart-mset.cdcl_W-all-struct-inv-def pcdcl-all-struct-invs-def
     state_W-of-init.simps[symmetric]
  by fast+
have ‹cdcl_W-restart-mset.cdcl_W-conflicting (state_W-of-init T)›
  using struct-invs unfolding twl-struct-invs-init-def
    cdcl_W-restart-mset.cdcl_W-all-struct-inv-def pcdcl-all-struct-invs-def
     state_W-of-init.simps[symmetric]
  by fast
have ‹unsatisfiable (set-mset (remdups-mset `# mset `# mset (rev CS)))›
  using conflict-of-level-unsatisfiable[OF all-struct-invs] count-dec confl
    learned le clss
  by (auto simp: clauses-def mset-take-mset-drop-mset′ twl-st-init twl-st-wl-init
      image-image to-init-state0-def init-state0-def ac-simps pcdcl-all-struct-invs-def
      cdcl_W-restart-mset.cdcl_W-learned-clauses-entailed-by-init-def ac-simps
twl-st-l-init pget-all-init-clss-pstate_W-of-init)
then have unsat[simp]: ‹unsatisfiable (mset ` set CS)›
  using satisfiable-remdups-iff[of ‹set CS›]
  by auto
then have [simp]: ‹CS ≠ []›
  by (auto simp del: unsat)
show ?thesis
  unfolding conclusive-CDCL-state-def
  apply (rule RETURN-SPEC-refine)
  apply (rule exI[of - ‹pstate_W-of (fst T)›])
  apply (intro conjI)
  subgoal
    by auto
  subgoal
    using struct-invs learned count-dec clss confl
    by (clarsimp simp: twl-st-init twl-st-wl-init twl-st-l-init)
  subgoal
    using struct-invs learned count-dec clss confl
```

```
      by (clarsimp simp: twl-st-init twl-st-wl-init twl-st-l-init)
    done
 qed
 have empty-clauses: ‹RETURN (fst init-state0)
≤ ⇓ {(S, T). T = pstate_W-of S}
   (SPEC
    (conclusive-CDCL-state (mset ‘# mset CS)
     (pinit-state (mset ‘# mset CS))))›
   if
    ‹T ∈ Collect (init-dt-spec0 CS (to-init-state0 init-state0))› and
    ‹¬ get-conflict (fst T) ≠ None› and
    ‹CS = []›
   for T
 proof −
   have [dest]: ‹cdcl_W-restart-mset.cdcl_W ([], {#}, {#}, None) (a, aa, ab, b) ⟹ False›
    for a aa ab b
    by (metis cdcl_W-restart-mset.cdcl_W.cases cdcl_W-restart-mset.cdcl_W-stgy.conflict'
     cdcl_W-restart-mset.cdcl_W-stgy.propagate' cdcl_W-restart-mset.other'
cdcl_W-stgy-cdcl_W-init-state-empty-no-step init-state.simps)
   show ?thesis
    by (rule RETURN-RES-refine)
     (use that in ‹auto simp: conclusive-CDCL-state-def init-state0-def›)
 qed

 have extract-atms-clss-nempty: ‹extract-atms-clss CS {} ≠ {}›
   if
    ‹T ∈ Collect (init-dt-spec0 CS (to-init-state0 init-state0))› and
    ‹¬ get-conflict (fst T) ≠ None› and
    ‹CS ≠ []›
   for T
 proof −
   show ?thesis
    using that
    by (cases T; cases CS)
     (auto simp: init-state0-def to-init-state0-def init-dt-spec0-def
      extract-atms-clss-alt-def)
 qed

 have cdcl-twl-stgy-restart-prog: ‹cdcl-twl-stgy-restart-prog (fst T)
≤ ⇓ {(S, T). T = pstate_W-of S}
   (SPEC
    (conclusive-CDCL-state (mset ‘# mset CS)
     (pinit-state (mset ‘# mset CS)))› (is ?G1) and
    cdcl-twl-stgy-restart-prog-early: ‹cdcl-twl-stgy-restart-prog-early (fst T)
≤ ⇓ {(S, T). T = pstate_W-of S}
   (SPEC
    (conclusive-CDCL-state (mset ‘# mset CS)
     (pinit-state (mset ‘# mset CS))))› (is ?G2)
   if
    spec: ‹T ∈ Collect (init-dt-spec0 CS (to-init-state0 init-state0))› and
    confl: ‹¬ get-conflict (fst T) ≠ None› and
    CS-nempty[simp]: ‹CS ≠ []› and
    ‹extract-atms-clss CS {} ≠ {}› and
   ‹clause ‘# get-clauses (fst T) + unit-clss (fst T) + subsumed-clauses (fst T) + get-all-clauses0 (fst
T) =
     remdups-mset ‘# mset ‘# mset CS› and
```

```
    ‹get-learned-clss (fst T) = {#}›
  for T
proof −
  let ?CS = ‹remdups-mset ‘# mset ‘# mset CS›
  have
    struct-invs: ‹twl-struct-invs-init T› and
    clss-to-upd: ‹clauses-to-update-init T = {#}› and
    count-dec: ‹∀ s∈set (get-trail-init T). ¬ is-decided s› and
    ‹get-conflict-init T = None ⟶
     literals-to-update-init T =
     uminus ‘# lit-of ‘# mset (get-trail-init T)› and
    clss: ‹?CS +
       clauses (get-init-clauses-init (to-init-state0 init-state0)) +
       other-clauses-init (to-init-state0 init-state0) +
       get-unit-init-clauses-init (to-init-state0 init-state0) +
       get-init-clauses0-init (to-init-state0 init-state0) +
       get-subsumed-init-clauses-init (to-init-state0 init-state0) +
       get-init-clauses0-init (to-init-state0 init-state0) =
       clauses (get-init-clauses-init T) + other-clauses-init T +
       get-unit-init-clauses-init T +
       get-subsumed-init-clauses-init T +
       get-init-clauses0-init T› and
    learned: ‹get-learned-clauses-init (to-init-state0 init-state0) =
        get-learned-clauses-init T›
      ‹get-unit-learned-clauses-init T =
        get-unit-learned-clauses-init (to-init-state0 init-state0)›
      ‹get-subsumed-learned-clauses-init T =
        get-subsumed-learned-clauses-init (to-init-state0 init-state0)›
      ‹get-learned-clauses0-init T =
        get-learned-clauses0-init (to-init-state0 init-state0)› and
    stgy-invs: ‹twl-stgy-invs (fst T)› and
    oth: ‹other-clauses-init T ≠ {#} ⟶ get-conflict-init T ≠ None› and
    ‹{#} ∈# mset ‘# mset CS ⟶ get-conflict-init T ≠ None› and
    ‹get-conflict-init (to-init-state0 init-state0) ≠ None ⟶
     get-conflict-init (to-init-state0 init-state0) = get-conflict-init T›
    using spec unfolding init-dt-wl-spec-def init-dt-spec0-def
      Set.mem-Collect-eq apply −
    apply normalize-goal+
    by metis+
  have struct-invs: ‹twl-struct-invs (fst T)›
    by (rule twl-struct-invs-init-twl-struct-invs)
      (use struct-invs oth confl in ‹auto simp: twl-st-init›)
  have clss-to-upd: ‹clauses-to-update (fst T) = {#}›
    using clss-to-upd by (auto simp: twl-st-init)

  have init: ‹cdcl_W-restart-mset.cdcl_W-learned-clauses-entailed-by-init (state-of (pstate_W-of (fst T)))›
    using learned
    apply (auto simp: cdcl_W-restart-mset.cdcl_W-learned-clauses-entailed-by-init-def
      get-all-learned-clss-alt-def init-state0-def to-init-state0-def)
    by (smt add.right-neutral get-learned-clauses0.elims get-learned-clauses0-init.simps
      get-unit-learned-clauses-init.elims prod.sel(1) set-mset-union union-trus-clss-clss)
  have CS': ‹(λx. remdups-mset (mset x)) ‘ set CS  = clause ‘ set-mset (get-init-clauses-init T) ∪
set-mset (get-unit-init-clauses-init T) ∪
      set-mset (get-subsumed-init-clauses-init T) ∪ set-mset (get-init-clauses0-init T)›
    using arg-cong[OF clss, of set-mset] oth confl
    by (cases ‹other-clauses-init T = {#}›)
```

```
    (auto 5 3 simp: init-state0-def to-init-state0-def)
  have conclusive-le: ‹conclusive-TWL-run (fst T)
  ≤ ⇓ {(S, T). T = pstate_W-of S}
    (SPEC
    (conclusive-CDCL-state (mset `# mset CS) (pinit-state (mset `# mset CS))))›
    using satisfiable-remdups-iff[of ‹set CS›]
    apply −
    apply (rule order-trans[OF conclusive-TWL-run-conclusive-CDCL-state[of ‹fst T›]])
    using conclusive-TWL-run-conclusive-CDCL-state[of ‹fst T›] clss oth
    apply (cases ‹other-clauses-init T = {#}›)
    apply (auto simp: br-def struct-invs stgy-invs init get-all-init-clss-alt-init-def
      to-init-state0-def init-state0-def conclusive-CDCL-state-def CS' true-clss-def
      image-image  true-annots-remdups-mset[of - ‹mset ` set CS›, symmetric]
      intro!: ref-two-step'')
    done
  show ?G1
    apply (rule cdcl-twl-stgy-restart-restart-prog-spec[THEN order-trans])
        apply (rule struct-invs; fail)
       apply (rule stgy-invs; fail)
      apply (rule clss-to-upd; fail)
     apply (use confl in fast; fail)
    apply (rule init[unfolded state_W-of-def[symmetric]]; fail)
    apply (rule conclusive-le)
    done
  show ?G2
    apply (rule cdcl-twl-stgy-restart-prog-early-spec[THEN order-trans])
        apply (rule struct-invs; fail)
       apply (rule stgy-invs; fail)
      apply (rule clss-to-upd; fail)
     apply (use confl in fast; fail)
    apply (rule init[unfolded state_W-of-def[symmetric]]; fail)
    apply (rule conclusive-le)
    done
qed

show ?thesis
  unfolding SAT0-def SAT-def
  apply (refine-vcg lhs-step-If)
  subgoal for b T
    by (rule conflict-during-init)
  subgoal by (rule empty-clauses)
  subgoal for b T
    by (rule extract-atms-clss-nempty)
  subgoal for b T
    by (cases T)
      (auto simp: init-state0-def to-init-state0-def init-dt-spec0-def
        extract-atms-clss-alt-def)
  subgoal for b T
    by (cases T)
      (auto simp: init-state0-def to-init-state0-def init-dt-spec0-def
        extract-atms-clss-alt-def)
  subgoal for b T
    by (cases T)
      (auto simp: init-state0-def to-init-state0-def init-dt-spec0-def
        extract-atms-clss-alt-def)
  subgoal for b T
```

```
    by (cases T)
      (auto simp: init-state0-def to-init-state0-def init-dt-spec0-def
        extract-atms-clss-alt-def)
  subgoal for b T
    by (rule cdcl-twl-stgy-restart-prog)
  subgoal for b T
    by (rule conflict-during-init)
  subgoal by (rule empty-clauses)
  subgoal for b T
    by (rule extract-atms-clss-nempty)
  subgoal premises p for b - - T
    using p(6−)
    by (cases T)
      (auto simp: init-state0-def to-init-state0-def init-dt-spec0-def
        extract-atms-clss-alt-def)
  subgoal premises p for b - - T
    using p(6−)
    by (cases T)
      (auto simp: init-state0-def to-init-state0-def init-dt-spec0-def
        extract-atms-clss-alt-def)
  subgoal premises p for b - - T
    using p(6−)
    by (cases T)
      (auto simp: init-state0-def to-init-state0-def init-dt-spec0-def
        extract-atms-clss-alt-def)
  subgoal for b T
    by (rule cdcl-twl-stgy-restart-prog)
  subgoal for b T
    by (rule conflict-during-init)
  subgoal by (rule empty-clauses)
  subgoal for b T
    by (rule extract-atms-clss-nempty)
  subgoal for b T
    by (cases T)
      (auto simp: init-state0-def to-init-state0-def init-dt-spec0-def
        extract-atms-clss-alt-def)
  subgoal for b T
    by (cases T)
      (auto simp: init-state0-def to-init-state0-def init-dt-spec0-def
        extract-atms-clss-alt-def)
  subgoal for b T
    by (cases T)
      (auto simp: init-state0-def to-init-state0-def init-dt-spec0-def
        extract-atms-clss-alt-def)
  subgoal for b T
    by (rule cdcl-twl-stgy-restart-prog-early)
  done
qed

definition SAT-l :: ‹nat clause-l list ⇒ nat twl-st-l nres› where
  ‹SAT-l CS = do{
    b ← SPEC(λ-::bool. True);
    if b then do {
        let S = init-state-l;
        T ← init-dt CS (to-init-state-l S);
        let T = fst T;
```

```
      if get-conflict-l T ≠ None
      then RETURN T
      else if CS = [] then RETURN (fst init-state-l)
      else do {
        ASSERT (extract-atms-clss CS {} ≠ {});
  ASSERT (clauses-to-update-l T = {#});
        ASSERT(mset `# ran-mf (get-clauses-l T) + get-unit-clauses-l T +
          get-subsumed-clauses-l T + get-clauses0-l T = remdups-mset `# mset `# mset CS);
        ASSERT(learned-clss-l (get-clauses-l T) = {#});
        cdcl-twl-stgy-restart-prog-l T
      }
    }
    else do {
      let S = init-state-l;
      T ← init-dt CS (to-init-state-l S);
      failed ← SPEC (λ- :: bool. True);
      if failed then do {
        T ← init-dt CS (to-init-state-l S);
        let T = fst T;
        if get-conflict-l T ≠ None
        then RETURN T
        else if CS = [] then RETURN (fst init-state-l)
        else do {
          ASSERT (extract-atms-clss CS {} ≠ {});
          ASSERT (clauses-to-update-l T = {#});
          ASSERT(mset `# ran-mf (get-clauses-l T) + get-unit-clauses-l T +
              get-subsumed-clauses-l T + get-clauses0-l T = remdups-mset `# mset `# mset CS);
          ASSERT(learned-clss-l (get-clauses-l T) = {#});
          cdcl-twl-stgy-restart-prog-l T
        }
      } else do {
        let T = fst T;
        if get-conflict-l T ≠ None
        then RETURN T
        else if CS = [] then RETURN (fst init-state-l)
        else do {
          ASSERT (extract-atms-clss CS {} ≠ {});
          ASSERT (clauses-to-update-l T = {#});
          ASSERT(mset `# ran-mf (get-clauses-l T) + get-unit-clauses-l T +
            get-subsumed-clauses-l T  + get-clauses0-l T = remdups-mset `# mset `# mset CS);
          ASSERT(learned-clss-l (get-clauses-l T) = {#});
          cdcl-twl-stgy-restart-prog-early-l T
        }
      }
    }
  }⟩

lemma [twl-st-l-init, simp]:
  assumes ⟨(s, x) ∈ twl-st-l-init⟩
  shows ⟨get-learned-clauses0-init x = get-learned-clauses0-l-init s⟩
    ⟨get-init-clauses0-init x = get-init-clauses0-l-init s⟩
  using assms by (auto simp: twl-st-l-init-def)


lemma SAT-l-SAT0:
  shows ⟨SAT-l CS ≤ ⇓ {(T,T'). (T, T') ∈ twl-st-l None} (SAT0 CS)⟩
```

```
proof −
  have inj: ‹inj (uminus :: - literal ⇒ -)›
    by (auto simp: inj-on-def)
  have [simp]: ‹{#− lit-of x. x ∈# A#} = {#− lit-of x. x ∈# B#} ⟷
    {#lit-of x. x ∈# A#} = {#lit-of x. x ∈# B#}› for A B :: ‹(nat literal, nat literal,
          nat) annotated-lit multiset›
    unfolding multiset.map-comp[unfolded comp-def, symmetric]
    apply (subst inj-image-mset-eq-iff[of uminus])
    apply (rule inj)
    by (auto simp: inj-on-def)[]
  have get-unit-twl-st-l: ‹(s, x) ∈ twl-st-l-init ⟹ get-learned-unit-clauses-l-init s = {#} ⟹
      learned-clss-l (get-clauses-l-init s) = {#} ⟹
      get-subsumed-learned-clauses-l-init s = {#} ⟹
    {#mset (fst x). x ∈# ran-m (get-clauses-l-init s)#} +
    (get-unit-clauses-l-init s + get-subsumed-init-clauses-l-init s) =
    clause `# get-init-clauses-init x + get-unit-init-clauses-init x +
      get-subsumed-init-clauses-init x› for s x
    apply (cases s; cases x)
    apply (auto simp: twl-st-l-init-def mset-take-mset-drop-mset')
    by (metis (mono-tags, lifting) add.right-neutral all-clss-l-ran-m)

  have init-dt-pre: ‹init-dt-pre CS (to-init-state-l init-state-l)›
    by (rule init-dt-pre-init)

  have init-dt-spec0: ‹init-dt CS (to-init-state-l init-state-l)
      ≤ ⇓{((T),T'). (T, T') ∈ twl-st-l-init ∧ twl-list-invs (fst T) ∧
          clauses-to-update-l (fst T) = {#}}
        (SPEC (init-dt-spec0 CS (to-init-state0 init-state0)))›
    apply (rule init-dt-full[THEN order-trans])
    subgoal by (rule init-dt-pre)
    subgoal
      apply (rule RES-refine)
      unfolding init-dt-spec-def Set.mem-Collect-eq init-dt-spec0-def
        to-init-state-l-def init-state-l-def
        to-init-state0-def init-state0-def
      apply normalize-goal+
      apply (rule-tac x=x in bexI)
      subgoal for s x by (auto simp: twl-st-l-init)
      subgoal for s x
        unfolding Set.mem-Collect-eq
        by (simp-all add: twl-st-init twl-st-l-init twl-st-l-init-no-decision-iff get-unit-twl-st-l)
      done
    done

  have init-state0: ‹(fst init-state-l, fst init-state0) ∈ {(T, T'). (T, T') ∈ twl-st-l None}›
    by (auto simp: twl-st-l-def init-state0-def init-state-l-def)
  show ?thesis
    unfolding SAT-l-def SAT0-def
    apply (refine-vcg init-dt-spec0)
    subgoal by auto
    subgoal by (auto simp: twl-st-l-init twl-st-init)
    subgoal by (auto simp: twl-st-l-init-alt-def)
    subgoal by auto
    subgoal by (rule init-state0)
    subgoal for b ba T Ta
      unfolding all-clss-lf-ran-m[symmetric] image-mset-union to-init-state0-def init-state0-def
```

by (*cases T*; *cases Ta*)
(*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset′*
init-dt-spec0-def)
subgoal for *b ba T Ta*
unfolding *all-clss-lf-ran-m*[*symmetric*] *image-mset-union*
by (*cases T*; *cases Ta*)
(*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset′*
, *auto simp*: *ac-simps*)
subgoal for *b ba T Ta*
by (*cases T*; *cases Ta*)
(*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset′*)
subgoal for *b ba T Ta*
by (*rule cdcl-twl-stgy-restart-prog-l-cdcl-twl-stgy-restart-prog*[*THEN fref-to-Down*, *of* - ‹*fst Ta*›,
THEN order-trans])
(*auto simp*: *twl-st-l-init-alt-def mset-take-mset-drop-mset′ intro*!: *conc-fun-R-mono*)
subgoal by (*auto simp*: *twl-st-l-init twl-st-init*)
subgoal by (*auto simp*: *twl-st-l-init twl-st-init*)
subgoal by (*auto simp*: *twl-st-l-init-alt-def*)
subgoal by *auto*
subgoal by (*rule init-state0*)
subgoal for *b ba* - - - - *T Ta*
unfolding *all-clss-lf-ran-m*[*symmetric*] *image-mset-union to-init-state0-def init-state0-def*
by (*cases T*; *cases Ta*)
(*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset′*
init-dt-spec0-def)
subgoal for *b ba* - - - - *T Ta*
unfolding *all-clss-lf-ran-m*[*symmetric*] *image-mset-union*
by (*cases T*; *cases Ta*) (*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset′*
, *auto simp*: *ac-simps*)
subgoal for *b ba* - - - - *T Ta*
by (*cases T*; *cases Ta*) (*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset′*)
subgoal for *b ba* - - - - *T Ta*
by (*rule cdcl-twl-stgy-restart-prog-l-cdcl-twl-stgy-restart-prog*[*THEN fref-to-Down*, *of* - ‹*fst Ta*›,
THEN order-trans])
(*auto simp*: *twl-st-l-init-alt-def intro*!: *conc-fun-R-mono*)
subgoal by (*auto simp*: *twl-st-l-init twl-st-init*)
subgoal by (*auto simp*: *twl-st-l-init-alt-def*)
subgoal by *auto*
subgoal by (*rule init-state0*)
subgoal by *auto*
subgoal for *b ba T Ta*
unfolding *all-clss-lf-ran-m*[*symmetric*] *image-mset-union*
by (*cases T*; *cases Ta*) (*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset′*
, *auto simp*: *ac-simps*)
subgoal for *b ba T Ta*
by (*cases T*; *cases Ta*) (*auto simp*: *twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset′*)
subgoal for *b ba T Ta*
apply (*rule order-trans*)
apply (*rule cdcl-twl-stgy-restart-prog-early-l-cdcl-twl-stgy-restart-abs-early-l*[*THEN fref-to-Down*, *of*
- ‹*fst T*›])
apply *fast*
apply (*solves simp*)
apply (*subst Down-id-eq*)
apply (*rule cdcl-twl-stgy-restart-abs-early-l-cdcl-twl-stgy-restart-abs-early*[*THEN fref-to-Down*, *of* -
‹*fst Ta*›,
THEN order-trans])

```
    apply (auto simp: twl-st-l-init-alt-def intro!: conc-fun-R-mono)
    done
  done
qed

definition SAT-wl :: ‹nat clause-l list ⇒ nat twl-st-wl nres› where
  ‹SAT-wl CS = do{
    ASSERT(isasat-input-bounded (mset-set (extract-atms-clss CS {})));
    let 𝒜in' = extract-atms-clss CS {};
    b ← SPEC(λ-::bool. True);
    if b then do {
        let S = init-state-wl;
        T ← init-dt-wl' CS (to-init-state S);
        T ← rewatch-st (from-init-state T);
        if get-conflict-wl T ≠ None
        then RETURN T
        else if CS = [] then RETURN (([], fmempty, None, {#}, {#}, {#}, {#}, {#}, {#}, {#},
{#}, λ-. undefined))
        else do {
   ASSERT (extract-atms-clss CS {} ≠ {});
   ASSERT(isasat-input-bounded-nempty (mset-set 𝒜in'));
   ASSERT(mset ‘# ran-mf (get-clauses-wl T) + get-unit-clauses-wl T +
           get-subsumed-clauses-wl T + get-clauses0-wl T  = remdups-mset ‘# mset ‘# mset CS);
   ASSERT(learned-clss-l (get-clauses-wl T) = {#});
   cdcl-twl-stgy-restart-prog-wl (finalise-init T)
        }
    }
    else do {
        let S = init-state-wl;
        T ← init-dt-wl' CS (to-init-state S);
        let T = from-init-state T;
        failed ← SPEC (λ- :: bool. True);
        if failed then do {
          let S = init-state-wl;
          T ← init-dt-wl' CS (to-init-state S);
          T ← rewatch-st (from-init-state T);
          if get-conflict-wl T ≠ None
          then RETURN T
         else if CS = [] then RETURN (([], fmempty, None, {#}, {#}, {#}, {#}, {#}, {#}, {#},
{#},λ-. undefined))
          else do {
            ASSERT (extract-atms-clss CS {} ≠ {});
            ASSERT(isasat-input-bounded-nempty (mset-set 𝒜in'));
            ASSERT(mset ‘# ran-mf (get-clauses-wl T) + get-unit-clauses-wl T +
              get-subsumed-clauses-wl T + get-clauses0-wl T  = remdups-mset ‘# mset ‘# mset CS);
            ASSERT(learned-clss-l (get-clauses-wl T) = {#});
            cdcl-twl-stgy-restart-prog-wl (finalise-init T)
          }
        } else do {
          if get-conflict-wl T ≠ None
          then RETURN T
         else if CS = [] then RETURN (([], fmempty, None, {#}, {#}, {#}, {#}, {#}, {#}, {#},
{#},λ-. undefined))
          else do {
            ASSERT (extract-atms-clss CS {} ≠ {});
            ASSERT(isasat-input-bounded-nempty (mset-set 𝒜in'));
```

```
        ASSERT(mset ‘# ran-mf (get-clauses-wl T) + get-unit-clauses-wl T +
          get-subsumed-clauses-wl T  + get-clauses0-wl T  = remdups-mset ‘# mset ‘# mset CS);
        ASSERT(learned-clss-l (get-clauses-wl T) = {#});
        T ← rewatch-st (finalise-init T);
        cdcl-twl-stgy-restart-prog-early-wl T
      }
    }
  }
}›

lemma SAT-l-alt-def:
  ‹SAT-l CS = do{
    A ← RETURN (); ////////
    b ← SPEC(λ-::bool. True);
    if b then do {
        let S = init-state-l;
        A ← RETURN (); ////////////
        T ← init-dt CS (to-init-state-l S);  ////////
        let T = fst T;
        if get-conflict-l T ≠ None
        then RETURN T
        else if CS = [] then RETURN (fst init-state-l)
        else do {
          ASSERT (extract-atms-clss CS {} ≠ {});
    ASSERT (clauses-to-update-l T = {#});
          ASSERT(mset ‘# ran-mf (get-clauses-l T) + get-unit-clauses-l T +
            get-subsumed-clauses-l T + get-clauses0-l T = remdups-mset ‘# mset ‘# mset CS);
          ASSERT(learned-clss-l (get-clauses-l T) = {#});
          cdcl-twl-stgy-restart-prog-l T
        }
    }
    else do {
        let S = init-state-l;
        A ← RETURN (); ////////////
        T ← init-dt CS (to-init-state-l S);
        failed ← SPEC (λ- :: bool. True);
        if failed then do {
          let S = init-state-l;
          A ← RETURN (); ////////////
          T ← init-dt CS (to-init-state-l S);
          let T = T;
          if get-conflict-l-init T ≠ None
          then RETURN (fst T)
          else if CS = [] then RETURN (fst init-state-l)
          else do {
            ASSERT (extract-atms-clss CS {} ≠ {});
            ASSERT (clauses-to-update-l (fst T) = {#});
            ASSERT(mset ‘# ran-mf (get-clauses-l (fst T)) + get-unit-clauses-l (fst T) +
               get-subsumed-clauses-l (fst T)+ get-clauses0-l (fst T)  = remdups-mset ‘# mset ‘# mset
CS);
            ASSERT(learned-clss-l (get-clauses-l (fst T)) = {#});
            let T = fst T;
            cdcl-twl-stgy-restart-prog-l T
          }
        } else do {
```

```
        let T = T;
        if get-conflict-l-init T ≠ None
        then RETURN (fst T)
        else if CS = [] then RETURN (fst init-state-l)
        else do {
          ASSERT (extract-atms-clss CS {} ≠ {});
          ASSERT (clauses-to-update-l (fst T) = {#});
          ASSERT(mset `# ran-mf (get-clauses-l (fst T)) + get-unit-clauses-l (fst T) +
              get-subsumed-clauses-l (fst T) + get-clauses0-l (fst T) = remdups-mset `# mset `# mset
CS);
          ASSERT(learned-clss-l (get-clauses-l (fst T)) = {#});
          let T = fst T;
          cdcl-twl-stgy-restart-prog-early-l T
        }
      }
    }
  }›
  unfolding SAT-l-def by (auto cong: if-cong Let-def twl-st-l-init)

lemma init-dt-wl-full-init-dt-wl-spec-full:
  assumes ‹init-dt-wl-pre CS S› and ‹init-dt-pre CS S'› and
    ‹(S, S') ∈ state-wl-l-init›
  shows ‹init-dt-wl-full CS S ≤ ⇓ {(S, S'). (fst S, fst S') ∈ state-wl-l None} (init-dt CS S')›
proof −
  have init-dt-wl: ‹init-dt-wl CS S ≤ SPEC (λT. RETURN T ≤ ⇓ state-wl-l-init (init-dt CS S') ∧
     init-dt-wl-spec CS S T)›
    apply (rule SPEC-rule-conjI)
    apply (rule order-trans)
    apply (rule init-dt-wl-init-dt[of S S'])
    subgoal by (rule assms)
    apply (rule no-fail-spec-le-RETURN-itself)
    subgoal
      apply (rule SPEC-nofail)
      apply (rule order-trans)
      apply (rule ref-two-step')
      apply (rule init-dt-full)
      using assms by (auto simp: conc-fun-RES init-dt-wl-pre-def)
    subgoal
      apply (rule order-trans)
      apply (rule init-dt-wl-init-dt-wl-spec)
      apply (rule assms)
      apply simp
      done
    done

  show ?thesis
    unfolding init-dt-wl-full-def
    apply (rule specify-left)
    apply (rule init-dt-wl)
    subgoal for x
      apply (cases x, cases ‹fst x›)
      apply (simp only: prod.case fst-conv)
      apply normalize-goal+
      apply (rule specify-left)
      apply (rule-tac M =aa and N=ba and C=c and NE=d and UE=e and NEk=f and UEk=g
and NS=h and
```

```
      US=i and Q=l in
  rewatch-correctness[OF - init-dt-wl-spec-rewatch-pre])
    subgoal by rule
    apply (assumption)
    apply (auto)[3]
    apply (cases ‹init-dt CS S'›)
    apply (auto simp: RETURN-RES-refine-iff state-wl-l-def state-wl-l-init-def
      state-wl-l-init'-def)
    done
  done
qed

lemma init-dt-wl-pre:
  shows ‹init-dt-wl-pre CS (to-init-state init-state-wl)›
  unfolding init-dt-wl-pre-def to-init-state-def init-state-wl-def
  apply (rule exI[of - ‹(([], fmempty, None, {#}, {#}, {#}, {#}, {#}, {#}, {#}, {#}, {#}),
{#})›])
  apply (intro conjI)
   apply (auto simp: init-dt-pre-def state-wl-l-init-def state-wl-l-init'-def)[]
  unfolding init-dt-pre-def
  apply (rule exI[of - ‹(([], {#}, {#}, None, {#}, {#}, {#}, {#}, {#}, {#}, {#}, {#}), {#})›])
  by (auto simp: init-dt-pre-def state-wl-l-init-def state-wl-l-init'-def
    twl-st-l-init-def twl-init-invs)[]


lemma SAT-wl-SAT-l:
  assumes
    bounded: ‹isasat-input-bounded (mset-set (⋃C∈set CS. atm-of ‘ set C))›
  shows ‹SAT-wl CS ≤ ⇓ {(T,T'). (T, T') ∈ state-wl-l None} (SAT-l CS)›
proof −
  have extract-atms-clss: ‹(extract-atms-clss CS {}, ()) ∈ {(x, -). x = extract-atms-clss CS {}}›
    by auto
  have init-dt-wl-pre: ‹init-dt-wl-pre CS (to-init-state init-state-wl)›
    by (rule init-dt-wl-pre)

  have init-rel: ‹(to-init-state init-state-wl, to-init-state-l init-state-l)
    ∈ state-wl-l-init›
    by (auto simp: init-dt-pre-def state-wl-l-init-def state-wl-l-init'-def
      twl-st-l-init-def twl-init-invs to-init-state-def init-state-wl-def
      init-state-l-def to-init-state-l-def)[]

  — The following stlightly strange theorem allows to reuse the definition and the correctness of
init-dt-wl-heur-full, which was split in the definition for purely refinement-related reasons.
  define init-dt-wl-rel where
    ‹init-dt-wl-rel S ≡ ({(T, T'). RETURN T ≤ init-dt-wl' CS S ∧ T' = ()})› for S
  have init-dt-wl':
    ‹init-dt-wl' CS S ≤  SPEC (λc. (c, ()) ∈ (init-dt-wl-rel S))›
    if
      ‹init-dt-wl-pre CS S› and
      ‹(S, S') ∈ state-wl-l-init›
      for S S'
  proof −
    have [simp]: ‹(U, U') ∈ ({(T, T'). RETURN T ≤ init-dt-wl' CS S ∧ remove-watched T = T'} O
        state-wl-l-init) ⟷ ((U, U') ∈ {(T, T'). remove-watched T = T'} O
        state-wl-l-init ∧ RETURN U ≤ init-dt-wl' CS S)› for S S' U U'
      by auto
```

```
    have H: ‹A ≤ ⇓ ({(S, S'). P S S'}) B ⟷ A ≤ ⇓ ({(S, S'). RETURN S ≤ A ∧ P S S'}) B›
      for A B P R
      by (simp add: pw-conc-inres pw-conc-nofail pw-le-iff p2rel-def)
    have nofail: ‹nofail (init-dt-wl' CS S)›
      apply (rule SPEC-nofail)
      apply (rule order-trans)
      apply (rule init-dt-wl'-spec[unfolded conc-fun-RES])
      using that by auto
    have H: ‹A ≤ ⇓ ({(S, S'). P S S'} O R) B ⟷ A ≤ ⇓ ({(S, S'). RETURN S ≤ A ∧ P S S'} O
R) B›
      for A B P R
      by (smt Collect-cong H case-prod-cong conc-fun-chain)
    show ?thesis
      unfolding init-dt-wl-rel-def
      using that
      by (auto simp: nofail no-fail-spec-le-RETURN-itself)
  qed

  have rewatch-st: ‹rewatch-st (from-init-state T) ≤
  ⇓ ({(S, S'). (S, fst S') ∈ state-wl-l None ∧ correct-watching S ∧
        literals-are-ℒ_in (all-atms-st (finalise-init S)) (finalise-init S)})
    (init-dt CS (to-init-state-l init-state-l))›
   (is ‹- ≤ ⇓ ?rewatch -›)
  if ‹(extract-atms-clss CS {}, 𝒜) ∈ {(x, -). x = extract-atms-clss CS {}}› and
     ‹(T, Ta) ∈ init-dt-wl-rel (to-init-state init-state-wl)›
    for T Ta and 𝒜 :: unit
  proof −
    have le-wa: ‹⇓ {(T, T'). T = append-empty-watched T'} A =
      (do {S ← A; RETURN (append-empty-watched S)})› for A R
      by (cases A)
        (auto simp: conc-fun-RES RES-RETURN-RES image-iff)
    have init': ‹init-dt-pre CS (to-init-state-l init-state-l)›
      by (rule init-dt-pre-init)
    have H: ‹do {T ← RETURN T; rewatch-st (from-init-state T)} ≤
       ⇓{(S', T'). S' = fst T'} (init-dt-wl-full CS (to-init-state init-state-wl))›
      using that unfolding init-dt-wl-full-def init-dt-wl-rel-def init-dt-wl'-def apply −
      apply (rule bind-refine[of - ‹{(T', T''). T' = append-empty-watched T''}›])
      apply (subst le-wa)
      apply (auto simp: rewatch-st-def from-init-state-def intro!: bind-refine[of - Id])
      done
    have [intro]: ‹correct-watching-init (af, ag, None, ai, aj, NEk, UEk, NS, US, N0, U0, {#}, ba) ⟹
      blits-in-ℒ_in (af, ag, ah, ai, aj, NEk, UEk, NS, US, N0, U0, ak, ba)› for af ag ah ai aj ak ba NS
US N0 U0 NEk UEk
      by (auto simp: correct-watching-init.simps blits-in-ℒ_in-def
        all-blits-are-in-problem-init.simps all-lits-st-def all-lits-def
  in-ℒ_all-atm-of-𝒜_in in-all-lits-of-mm-ain-atms-of-iff
  atm-of-all-lits-of-mm)

    have ‹rewatch-st (from-init-state T)
    ≤ ⇓ {(S, S'). (S, fst S') ∈ state-wl-l None}
      (init-dt CS (to-init-state-l init-state-l))›
     apply (rule H[simplified, THEN order-trans])
     apply (rule order-trans)
     apply (rule ref-two-step')
     apply (rule init-dt-wl-full-init-dt-wl-spec-full)
     subgoal by (rule init-dt-wl-pre)
```

apply (*rule init'*)
subgoal by (*auto simp*: *to-init-state-def init-state-wl-def to-init-state-l-def init-state-l-def state-wl-l-init-def state-wl-l-init'-def*)
by (*auto intro*!: *conc-fun-R-mono simp*: *conc-fun-chain*)

moreover have ‹*rewatch-st* (*from-init-state T*) ≤ *SPEC* (λ*S*. *correct-watching S* ∧ *literals-are-*$\mathcal{L}_{in}$ (*all-atms-st* (*finalise-init S*)) (*finalise-init S*))›
apply (*rule H*[*simplified*, *THEN order-trans*])
apply (*rule order-trans*)
apply (*rule ref-two-step'*)
apply (*rule Watched-Literals-Watch-List-Initialisation.init-dt-wl-full-init-dt-wl-spec-full*)
subgoal by (*rule init-dt-wl-pre*)
by (*auto simp*: *conc-fun-RES init-dt-wl-spec-full-def correct-watching-init-correct-watching finalise-init-def literals-are-*$\mathcal{L}_{in}$*-def is-*$\mathcal{L}_{all}$*-def* $\mathcal{L}_{all}$*-all-atms-all-lits simp flip*: *all-lits-st-alt-def IsaSAT-Setup.all-lits-st-alt-def*)
ultimately show *?thesis*
by (*rule add-invar-refineI-P*)
qed
have *cdcl-twl-stgy-restart-prog-wl-D*: ‹*cdcl-twl-stgy-restart-prog-wl* (*finalise-init U*) ≤ ⇓ {(*T*, *T'*). (*T*, *T'*) ∈ *state-wl-l None*} (*cdcl-twl-stgy-restart-prog-l* (*fst U'*))›
if
‹(*extract-atms-clss CS* {}, (*A*::*unit*)) ∈ {(*x*, -). *x* = *extract-atms-clss CS* {}}› **and**
UU': ‹(*U*, *U'*) ∈ *?rewatch*› **and**
‹¬ *get-conflict-wl U* ≠ *None*› **and**
‹¬ *get-conflict-l* (*fst U'*) ≠ *None*› **and**
‹*CS* ≠ []› **and**
‹*CS* ≠ []› **and**
‹*extract-atms-clss CS* {} ≠ {}› **and**
‹*clauses-to-update-l* (*fst U'*) = {#}› **and**
‹*mset* '# *ran-mf* (*get-clauses-l* (*fst U'*)) + *get-unit-clauses-l* (*fst U'*) + *get-subsumed-clauses-l* (*fst U'*) + *get-clauses0-l* (*fst U'*) = *remdups-mset* '# *mset* '# *mset CS*› **and**
‹*learned-clss-l* (*get-clauses-l* (*fst U'*)) = {#}› **and**
‹*extract-atms-clss CS* {} ≠ {}› **and**
‹*isasat-input-bounded-nempty* (*mset-set* (*extract-atms-clss CS* {}))› **and**
‹*mset* '# *ran-mf* (*get-clauses-wl U*) + *get-unit-clauses-wl U* + *get-subsumed-clauses-wl U* + *get-clauses0-wl U* = *remdups-mset* '# *mset* '# *mset CS*›
for $\mathcal{A}$ *T Ta U U'*
proof −
have *1*: ‹ {(*T*, *T'*). (*T*, *T'*) ∈ *state-wl-l None*} = *state-wl-l None*›
by *auto*
have *lits*: ‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st* (*finalise-init U*)) (*finalise-init U*)›
using *UU'* **by** (*auto simp*: *finalise-init-def*)
show *?thesis*
apply (*rule cdcl-twl-stgy-restart-prog-wl-spec*[*unfolded fref-param1*, *THEN fref-to-Down*, *THEN order-trans*])
apply *fast*
using *UU'* **by** (*auto simp*: *finalise-init-def*)
qed

have *conflict-during-init*:
‹(([], *fmempty*, *None*, {#}, {#}, {#}, {#}, {#}, {#}, {#}, {#}, λ-. *undefined*), *fst init-state-l*) ∈ {(*T*, *T'*). (*T*, *T'*) ∈ *state-wl-l None*}›
by (*auto simp*: *init-state-l-def state-wl-l-def*)

have *init-init-dt*: ‹*RETURN* (*from-init-state T*)
≤ ⇓ ({(*S*, *S'*). *S* = *fst S'*} *O* {(*S* :: *nat twl-st-wl-init-full*, *S'* :: *nat twl-st-wl-init*).
remove-watched S = *S'*} *O state-wl-l-init*)
(*init-dt CS* (*to-init-state-l init-state-l*))›
(**is** ‹- ≤ ⇓ *?init-dt* - ›)
if
‹(*extract-atms-clss CS* {}, (*A*::*unit*)) ∈ {(*x*, -). *x* = *extract-atms-clss CS* {}}› **and**
‹(*T*, *Ta*) ∈ *init-dt-wl-rel* (*to-init-state init-state-wl*)›
for *A T Ta*
proof −
have *1*: ‹*RETURN T* ≤ *init-dt-wl' CS* (*to-init-state init-state-wl*)›
using *that* **by** (*auto simp*: *init-dt-wl-rel-def from-init-state-def*)
have *2*: ‹*RETURN* (*from-init-state T*) ≤ ⇓ {(*S*, *S'*). *S* = *fst S'*} (*RETURN T*)›
by (*auto simp*: *RETURN-refine from-init-state-def*)
have *2*: ‹*RETURN* (*from-init-state T*) ≤ ⇓ {(*S*, *S'*). *S* = *fst S'*} (*init-dt-wl' CS* (*to-init-state init-state-wl*))›
apply (*rule 2*[*THEN order-trans*])
apply (*rule ref-two-step'*)
apply (*rule 1*)
done
show *?thesis*
apply (*rule order-trans*)
apply (*rule 2*)
unfolding *conc-fun-chain*[*symmetric*]
apply (*rule ref-two-step'*)
unfolding *conc-fun-chain*
apply (*rule init-dt-wl'-init-dt*)
apply (*rule init-dt-wl-pre*)
subgoal by (*auto simp*: *to-init-state-def init-state-wl-def to-init-state-l-def*
init-state-l-def state-wl-l-init-def state-wl-l-init'-def)
done
qed

have *rewatch-st-fst*: ‹*rewatch-st* (*finalise-init* (*from-init-state T*))
≤ *SPEC* (λ*c*. (*c*, *fst Ta*) ∈ {(*S*, *T*). (*S*, *T*) ∈ *state-wl-l None* ∧ *correct-watching S* ∧ *blits-in-*$\mathcal{L}_{in}$ *S*})›
(**is** ‹- ≤ *SPEC ?rewatch*›)
if

‹(*extract-atms-clss CS* {}, *A*) ∈ {(*x*, -). *x* = *extract-atms-clss CS* {}}› **and**
T: ‹(*T*, *A'*) ∈ *init-dt-wl-rel* (*to-init-state init-state-wl*)› **and**
T-Ta: ‹(*from-init-state T*, *Ta*)
∈ {(*S*, *S'*). *S* = *fst S'*} *O*
{(*S*, *S'*). *remove-watched S* = *S'*} *O state-wl-l-init*› **and**
‹¬ *get-conflict-wl* (*from-init-state T*) ≠ *None*› **and**
‹¬ *get-conflict-l-init Ta* ≠ *None*›
for *A b ba T A' Ta bb bc*
proof −
have *1*: ‹*RETURN T* ≤ *init-dt-wl' CS* (*to-init-state init-state-wl*)›
using *T* **unfolding** *init-dt-wl-rel-def* **by** *auto*
have *2*: ‹*RETURN T* ≤ ⇓ {(*S*, *S'*). *remove-watched S* = *S'*}
(*SPEC* (*init-dt-wl-spec CS* (*to-init-state init-state-wl*)))›
using *order-trans*[*OF 1 init-dt-wl'-spec*[*OF init-dt-wl-pre*]] **.**

have *empty-watched*: ‹*get-watched-wl* (*finalise-init* (*from-init-state T*)) = (λ-. [])›
using *1 2 init-dt-wl'-spec*[*OF init-dt-wl-pre*]

```
    by (cases T; cases ‹init-dt-wl CS (init-state-wl, {#})›)
     (auto simp: init-dt-wl-spec-def RETURN-RES-refine-iff
      finalise-init-def from-init-state-def state-wl-l-init-def
state-wl-l-init'-def to-init-state-def to-init-state-l-def
      init-state-l-def init-dt-wl'-def RES-RETURN-RES)

  have 1: ‹length (aa ∝ x) ≥ 2› ‹distinct (aa ∝ x)›
    if
      struct: ‹twl-struct-invs-init
        ((af,
        {#TWL-Clause (mset (watched-l (fst x))) (mset (unwatched-l (fst x)))
        . x ∈# init-clss-l aa#},
        {#}, y, ac, {#}, NS, US, N0, U0, {#}, ae),
       OC)› and
x: ‹x ∈# dom-m aa› and
learned: ‹learned-clss-l aa = {#}›
for af aa y ac ae x OC NS US N0 U0
  proof −
    have irred: ‹irred aa x›
      using that by (cases ‹fmlookup aa x›) (auto simp: ran-m-def dest!: multi-member-split
 split: if-splits)
    have ‹Multiset.Ball
({#TWL-Clause (mset (watched-l (fst x))) (mset (unwatched-l (fst x)))
. x ∈# init-clss-l aa#} +
{#})
struct-wf-twl-cls›
using struct unfolding twl-struct-invs-init-def fst-conv twl-st-inv.simps
by fast
    then show ‹length (aa ∝ x) ≥ 2› ‹distinct (aa ∝ x)›
      using x learned in-ran-mf-clause-inI[OF x, of True] irred
by (auto simp: mset-take-mset-drop-mset' dest!: multi-member-split[of x]
 split: if-splits)
  qed
  have min-len: ‹x ∈# dom-m (get-clauses-wl (finalise-init (from-init-state T))) ⟹
    distinct (get-clauses-wl (finalise-init (from-init-state T)) ∝ x) ∧
    2 ≤ length (get-clauses-wl (finalise-init (from-init-state T)) ∝ x)›
    for x
    using 2
    by (cases T)
     (auto simp: init-dt-wl-spec-def RETURN-RES-refine-iff
      finalise-init-def from-init-state-def state-wl-l-init-def
state-wl-l-init'-def to-init-state-def to-init-state-l-def
      init-state-l-def init-dt-wl'-def RES-RETURN-RES
      init-dt-spec-def init-state-wl-def twl-st-l-init-def
      intro: 1)

  show ?thesis
    apply (rule rewatch-st-correctness[THEN order-trans])
    subgoal by (rule empty-watched)
    subgoal by (rule min-len)
    subgoal using T-Ta by (auto simp: finalise-init-def
       state-wl-l-init-def state-wl-l-init'-def state-wl-l-def
 correct-watching-init-correct-watching
 correct-watching-init-blits-in-\mathcal{L}_{in})
    done
qed
```

have *cdcl-twl-stgy-restart-prog-wl-D2*: ‹*cdcl-twl-stgy-restart-prog-wl* U'
$\leq \Downarrow \{(T,\ T').\ (T,\ T') \in$ *state-wl-l None*$\}$
(*cdcl-twl-stgy-restart-prog-l* (*fst* T'))› (**is** *?A*) **and**
cdcl-twl-stgy-restart-prog-early-wl-D2: ‹*cdcl-twl-stgy-restart-prog-early-wl* U'
$\leq \Downarrow \{(T,\ T').\ (T,\ T') \in$ *state-wl-l None*$\}$
(*cdcl-twl-stgy-restart-prog-early-l* (*fst* T'))› (**is** *?B*)

if
U': ‹(U', *fst* T') $\in \{(S,\ T).\ (S,\ T) \in$ *state-wl-l None* $\wedge$ *correct-watching* $S \wedge$ *blits-in-*$\mathcal{L}_{in}$ $S\}$›
for $\mathcal{A}$ b b' T $\mathcal{A}'$ T' c c' U'
proof −
have *1*: ‹ $\{(T,\ T').\ (T,\ T') \in$ *state-wl-l None*$\}$ = *state-wl-l None*›
by *auto*
have *lits*: ‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st* (U')) (U')›
using U' **by** (*auto simp*: *finalise-init-def correct-watching.simps*)
show *?A*
apply (*rule cdcl-twl-stgy-restart-prog-wl-spec*[*unfolded fref-param1*, *THEN fref-to-Down*, *THEN order-trans*])
apply *fast*
using U' **by** (*auto simp*: *finalise-init-def*)
show *?B*
apply (*rule cdcl-twl-stgy-restart-prog-early-wl-spec*[*unfolded fref-param1*, *THEN fref-to-Down*, *THEN order-trans*])
apply *fast*
using U' **by** (*auto simp*: *finalise-init-def*)
qed
have *all-le*: ‹$\forall$ *C*$\in$*set CS*. $\forall$ *L*$\in$*set C*. *nat-of-lit L* $\leq$ *unat32-max*›
proof (*intro ballI*)
fix *C L*
assume ‹*C* $\in$ *set CS*› **and** ‹*L* $\in$ *set C*›
then have ‹*L* $\in\#$ $\mathcal{L}_{all}$ (*mset-set* ($\bigcup$ *C*$\in$*set CS*. *atm-of* ‘ *set C*))›
by (*auto simp*: *in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$)
then show ‹*nat-of-lit L* $\leq$ *unat32-max*›
using *assms* **by** *auto*
qed
have [*simp*]: ‹(*Tc*, *fst Td*) $\in$ *state-wl-l None* $\Longrightarrow$
get-conflict-l-init Td = *get-conflict-wl Tc*› **for** *Tc Td*
by (*cases Tc*; *cases Td*; *auto simp*: *state-wl-l-def*)
show *?thesis*
unfolding *SAT-wl-def SAT-l-alt-def*
apply (*refine-vcg extract-atms-clss init-dt-wl'* *init-rel*)
subgoal using *assms* **unfolding** *extract-atms-clss-alt-def* **by** *auto*
subgoal by *auto*
subgoal by (*rule init-dt-wl-pre*)
apply (*rule rewatch-st*; *assumption*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*rule conflict-during-init*)
subgoal using *bounded* **by** (*auto simp*: *isasat-input-bounded-nempty-def extract-atms-clss-alt-def*
simp del: *isasat-input-bounded-def*)
subgoal by *auto*
subgoal by *auto*
subgoal for $\mathcal{A}$ b *ba* T *Ta* U U'
by (*rule cdcl-twl-stgy-restart-prog-wl-D*)

subgoal by (*rule init-dt-wl-pre*)
apply (*rule init-init-dt*; *assumption*)
subgoal by *auto*
subgoal by (*rule init-dt-wl-pre*)
apply (*rule rewatch-st*; *assumption*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*rule conflict-during-init*)
subgoal using *bounded* **by** (*auto simp*: *isasat-input-bounded-nempty-def extract-atms-clss-alt-def*
simp del: *isasat-input-bounded-def*)
subgoal by *auto*
subgoal by *auto*
subgoal for $\mathcal{A}$ *b ba T Ta U U'*
unfolding *twl-st-l-init*[*symmetric*]
by (*rule cdcl-twl-stgy-restart-prog-wl-D*)
subgoal by (*auto simp*: *from-init-state-def state-wl-l-init-def state-wl-l-init'-def*)
subgoal for $\mathcal{A}$ *b ba T Ta U U'*
by (*cases U'*; *cases U*)
(*auto simp*: *from-init-state-def state-wl-l-init-def state-wl-l-init'-def*
state-wl-l-def)
subgoal by (*auto simp*: *from-init-state-def state-wl-l-init-def state-wl-l-init'-def*)
subgoal by (*rule conflict-during-init*)

subgoal using *bounded* **by** (*auto simp*: *isasat-input-bounded-nempty-def extract-atms-clss-alt-def*
simp del: *isasat-input-bounded-def*)
subgoal for $\mathcal{A}$ *b ba U* $\mathcal{A}'$ *T' bb bc*
by (*cases U*; *cases T'*)
(*auto simp*: *state-wl-l-init-def state-wl-l-init'-def*)
subgoal for $\mathcal{A}$ *b ba T* $\mathcal{A}'$ *T' bb bc*
by (*auto simp*: *state-wl-l-init-def state-wl-l-init'-def*)
apply (*rule rewatch-st-fst*; *assumption*)
subgoal by (*rule cdcl-twl-stgy-restart-prog-early-wl-D2*)
done
qed

definition *extract-model-of-state* **where**
‹*extract-model-of-state U* = *Some* (*map lit-of* (*get-trail-wl U*))›

definition *extract-stats* **where**
[*simp*]: ‹*extract-stats U* = *None*›

definition *extract-stats-init* **where**
[*simp*]: ‹*extract-stats-init* = *None*›

definition *IsaSAT* :: ‹*nat clause-l list* ⇒ *nat literal list option nres*› **where**
‹*IsaSAT CS* = *do*{
S ← *SAT-wl CS*;
RETURN (*if get-conflict-wl S* = *None then extract-model-of-state S else extract-stats S*)
}›

lemma *IsaSAT-alt-def*:
‹*IsaSAT CS* = *do*{
ASSERT(*isasat-input-bounded* (*mset-set* (*extract-atms-clss CS* {})));
let $\mathcal{A}_{in}'$ = *extract-atms-clss CS* {};

```
- ← RETURN ();
b ← SPEC(λ-::bool. True);
if b then do {
    let S = init-state-wl;
    T ← init-dt-wl' CS (to-init-state S);
    T ← rewatch-st (from-init-state T);
    if get-conflict-wl T ≠ None
    then RETURN (extract-stats T)
    else if CS = [] then RETURN (Some [])
    else do {
      ASSERT (extract-atms-clss CS {} ≠ {});
      ASSERT(isasat-input-bounded-nempty (mset-set 𝒜_in'));
      ASSERT(mset `# ran-mf (get-clauses-wl T) + get-unit-clauses-wl T +
        get-subsumed-clauses-wl T + get-clauses0-wl T = remdups-mset `# mset `# mset CS);
      ASSERT(learned-clss-l (get-clauses-wl T) = {#});
T ← RETURN (finalise-init T);
      S ← cdcl-twl-stgy-restart-prog-wl (T);
      RETURN (if get-conflict-wl S = None then extract-model-of-state S else extract-stats S)
    }
}
else do {
    let S = init-state-wl;
    T ← init-dt-wl' CS (to-init-state S);
    failed ← SPEC (λ- :: bool. True);
    if failed then do {
      let S = init-state-wl;
      T ← init-dt-wl' CS (to-init-state S);
      T ← rewatch-st (from-init-state T);
      if get-conflict-wl T ≠ None
      then RETURN (extract-stats T)
      else if CS = [] then RETURN (Some [])
      else do {
        ASSERT (extract-atms-clss CS {} ≠ {});
        ASSERT(isasat-input-bounded-nempty (mset-set 𝒜_in'));
        ASSERT(mset `# ran-mf (get-clauses-wl T) + get-unit-clauses-wl T +
          get-subsumed-clauses-wl T + get-clauses0-wl T = remdups-mset `# mset `# mset CS);
        ASSERT(learned-clss-l (get-clauses-wl T) = {#});
        let T = finalise-init T;
        S ← cdcl-twl-stgy-restart-prog-wl T;
        RETURN (if get-conflict-wl S = None then extract-model-of-state S else extract-stats S)
      }
    } else do {
      let T = from-init-state T;
      if get-conflict-wl T ≠ None
      then RETURN (extract-stats T)
      else if CS = [] then RETURN (Some [])
      else do {
        ASSERT (extract-atms-clss CS {} ≠ {});
        ASSERT(isasat-input-bounded-nempty (mset-set 𝒜_in'));
        ASSERT(mset `# ran-mf (get-clauses-wl T) + get-unit-clauses-wl T +
          get-subsumed-clauses-wl T + get-clauses0-wl T = remdups-mset `# mset `# mset CS);
        ASSERT(learned-clss-l (get-clauses-wl T) = {#});
        T ← rewatch-st T;
 T ← RETURN (finalise-init T);
        S ← cdcl-twl-stgy-restart-prog-early-wl T;
        RETURN (if get-conflict-wl S = None then extract-model-of-state S else extract-stats S)
```

```
        }
      }
    }
  }› (is ‹?A = ?B›) for CS opts
proof −
  have H: ‹A = B ⟹ A ≤ ⇓ Id B› for A B
    by auto
  have 1: ‹?A ≤ ⇓ Id ?B›
    unfolding IsaSAT-def SAT-wl-def nres-bind-let-law If-bind-distrib nres-monad-laws
      Let-def finalise-init-def
    apply (refine-vcg)
    subgoal by auto
    apply (rule H; solves auto)
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by (auto simp: extract-model-of-state-def)
    subgoal by auto
    subgoal by auto
    apply (rule H; solves auto)
    subgoal by auto
    subgoal by auto
    apply (rule H; solves auto)
    subgoal by auto

    subgoal by auto
    subgoal by auto
    subgoal by (auto simp: extract-model-of-state-def)
    subgoal by auto
    subgoal by auto
    apply (rule H; solves auto)
    subgoal by (auto simp: extract-model-of-state-def)
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by (auto simp: extract-model-of-state-def)
    subgoal by auto
    subgoal by auto
    apply (rule H; solves auto)
    apply (rule H; solves auto)
    subgoal by auto
    done

  have 2: ‹?B ≤ ⇓ Id ?A›
    unfolding IsaSAT-def SAT-wl-def nres-bind-let-law If-bind-distrib nres-monad-laws
      Let-def finalise-init-def
    apply (refine-vcg)
    subgoal by auto
    apply (rule H; solves auto)
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by (auto simp: extract-model-of-state-def)
    subgoal by auto
    subgoal by auto
    apply (rule H; solves auto)
```

subgoal by *auto*
subgoal by *auto*
apply (*rule H*; *solves auto*)
subgoal by *auto*

subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *extract-model-of-state-def*)
subgoal by *auto*
subgoal by *auto*
apply (*rule H*; *solves auto*)
subgoal by (*auto simp*: *extract-model-of-state-def*)
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by (*auto simp*: *extract-model-of-state-def*)
subgoal by *auto*
subgoal by *auto*
apply (*rule H*; *solves auto*)
apply (*rule H*; *solves auto*)
subgoal by *auto*
done

show *?thesis*
using *1 2* **by** *simp*
qed

lemma *extract-model-of-state-stat-alt-def*:
‹*extract-model-of-state-stat U* =
(*let* - = *print-trail-st2 U in*
(*False*, (*fst* (*get-trail-wl-heur U*)), (*get-stats-heur U*)))›
unfolding *extract-model-of-state-stat-def print-trail-st2-def*
by *auto*

definition *IsaSAT-use-fast-mode* **where**
‹*IsaSAT-use-fast-mode* = *True*›

definition *IsaSAT-heur* :: ‹*opts* ⇒ *nat clause-l list* ⇒ (*bool* × *nat literal list* × *isasat-stats*) *nres*› **where**
‹*IsaSAT-heur opts CS* = *do*{
ASSERT(*isasat-input-bounded* (*mset-set* (*extract-atms-clss CS* {})));
ASSERT(∀ *C*∈*set CS*. ∀ *L*∈*set C*. *nat-of-lit L* ≤ *unat32-max*);
let $\mathcal{A}_{in}'$ = *mset-set* (*extract-atms-clss CS* {});
ASSERT(*isasat-input-bounded* $\mathcal{A}_{in}'$);
ASSERT(*distinct-mset* $\mathcal{A}_{in}'$);
let $\mathcal{A}_{in}''$ = *virtual-copy* $\mathcal{A}_{in}'$;
let b = *opts-unbounded-mode opts*;
if b
then do {
S ← *init-state-wl-heur* $\mathcal{A}_{in}'$;
(*T*::*twl-st-wl-heur-init*, -) ← *init-dt-wl-heur True CS* (*S*, []);
T ← *rewatch-heur-st-init T*;
let T = *convert-state* $\mathcal{A}_{in}''$ *T*;
if ¬*get-conflict-wl-is-None-heur-init T*

```
    then RETURN (empty-init-code)
    else if CS = [] then empty-conflict-code
    else do {
      ASSERT(𝒜_in'' ≠ {#});
      ASSERT(isasat-input-bounded-nempty 𝒜_in'');
      - ← isasat-information-banner T;
      T ← finalise-init-code opts (T::twl-st-wl-heur-init);
      U ← cdcl-twl-stgy-restart-prog-wl-heur T;
      RETURN (if get-conflict-wl-is-None-heur U then extract-model-of-state-stat U
        else extract-state-stat U)
    }
}
else do {
    S ← init-state-wl-heur-fast 𝒜_in';
    (T::twl-st-wl-heur-init, -) ← init-dt-wl-heur False CS (S, []);
    let failed = is-failed-heur-init T ∨ ¬isasat-fast-init T;
    if failed then do {
      let 𝒜_in' = mset-set (extract-atms-clss CS {});
      S ← init-state-wl-heur 𝒜_in';
      (T::twl-st-wl-heur-init, -) ← init-dt-wl-heur True CS (S, []);
      let T = convert-state 𝒜_in'' T;
      T ← rewatch-heur-st-init T;
      if ¬get-conflict-wl-is-None-heur-init T
      then RETURN (empty-init-code)
      else if CS = [] then empty-conflict-code
      else do {
       ASSERT(𝒜_in'' ≠ {#});
       ASSERT(isasat-input-bounded-nempty 𝒜_in'');
       - ← isasat-information-banner T;
       T ← finalise-init-code opts (T::twl-st-wl-heur-init);
       U ← cdcl-twl-stgy-restart-prog-wl-heur T;
       RETURN (if get-conflict-wl-is-None-heur U then extract-model-of-state-stat U
         else extract-state-stat U)
      }
    }
    else do {
      let T = convert-state 𝒜_in'' T;
      if ¬get-conflict-wl-is-None-heur-init T
      then RETURN (empty-init-code)
      else if CS = [] then empty-conflict-code
      else do {
        ASSERT(𝒜_in'' ≠ {#});
        ASSERT(isasat-input-bounded-nempty 𝒜_in'');
        - ← isasat-information-banner T;
        ASSERT(rewatch-heur-st-fast-pre T);
        T ← rewatch-heur-st-init T;
        ASSERT(isasat-fast-init T);
        T ← finalise-init-code opts (T::twl-st-wl-heur-init);
        ASSERT(isasat-fast T);
        U ← cdcl-twl-stgy-restart-prog-early-wl-heur T;
        RETURN (if get-conflict-wl-is-None-heur U then extract-model-of-state-stat U
          else extract-state-stat U)
      }
    }
  }
}›
```

lemma *fref-to-Down-unRET-uncurry0-SPEC*:
  assumes ‹(λ-. (*f*), λ-. (*RETURN g*)) ∈ [*P*]$_f$ *unit-rel* → ⟨*B*⟩*nres-rel*› **and** ‹*P* ()›
  shows ‹*f* ≤ *SPEC* (λ*c*. (*c*, *g*) ∈ *B*)›
proof −
  have [*simp*]: ‹*RES* (*B*$^{-1}$ ‘‘ {*g*}) = *SPEC* (λ*c*. (*c*, *g*) ∈ *B*)›
    by *auto*
  show *?thesis*
    using *assms*
    unfolding *fref-def uncurry-def nres-rel-def RETURN-def*
    by (*auto simp*: *conc-fun-RES Image-iff*)
qed

lemma *fref-to-Down-unRET-SPEC*:
  assumes ‹(*f*, *RETURN o g*) ∈ [*P*]$_f$ *A* → ⟨*B*⟩*nres-rel*› **and**
    ‹*P y*› **and**
    ‹(*x*, *y*) ∈ *A*›
  shows ‹*f x* ≤ *SPEC* (λ*c*. (*c*, *g y*) ∈ *B*)›
proof −
  have [*simp*]: ‹*RES* (*B*$^{-1}$ ‘‘ {*g*}) = *SPEC* (λ*c*. (*c*, *g*) ∈ *B*)› **for** *g*
    by *auto*
  show *?thesis*
    using *assms*
    unfolding *fref-def uncurry-def nres-rel-def RETURN-def*
    by (*auto simp*: *conc-fun-RES Image-iff*)
qed

lemma *fref-to-Down-unRET-curry-SPEC*:
  assumes ‹(*uncurry f*, *uncurry* (*RETURN oo g*)) ∈ [*P*]$_f$ *A* → ⟨*B*⟩*nres-rel*› **and**
    ‹*P* (*x*, *y*)› **and**
    ‹((*x*′, *y*′), (*x*, *y*)) ∈ *A*›
  shows ‹*f x*′ *y*′ ≤ *SPEC* (λ*c*. (*c*, *g x y*) ∈ *B*)›
proof −
  have [*simp*]: ‹*RES* (*B*$^{-1}$ ‘‘ {*g*}) = *SPEC* (λ*c*. (*c*, *g*) ∈ *B*)› **for** *g*
    by *auto*
  show *?thesis*
    using *assms*
    unfolding *fref-def uncurry-def nres-rel-def RETURN-def*
    by (*auto simp*: *conc-fun-RES Image-iff*)
qed

lemma *all-lits-of-mm-empty-iff*: ‹*all-lits-of-mm A* = {#} ⟷ (∀ *C* ∈# *A*. *C* = {#})›
  apply (*induction A*)
  subgoal by *auto*
  subgoal by (*auto simp*: *all-lits-of-mm-add-mset*)
  done

lemma *all-lits-of-mm-extract-atms-clss*:
  ‹*L* ∈# (*all-lits-of-mm* (*mset* ‘# *mset CS*)) ⟷ *atm-of L* ∈ *extract-atms-clss CS* {}›
  by (*induction CS*)
    (*auto simp*: *extract-atms-clss-alt-def all-lits-of-mm-add-mset*
    in-all-lits-of-m-ain-atms-of-iff)

lemma *IsaSAT-heur-alt-def*:
  ‹*IsaSAT-heur opts CS* = *do*{

```
ASSERT(isasat-input-bounded (mset-set (extract-atms-clss CS {})));
ASSERT(∀ C∈set CS. ∀ L∈set C. nat-of-lit L ≤ unat32-max);
let 𝒜in' = mset-set (extract-atms-clss CS {});
ASSERT(isasat-input-bounded 𝒜in');
ASSERT(distinct-mset 𝒜in');
let 𝒜in'' = virtual-copy 𝒜in';
let b = opts-unbounded-mode opts;
if b
then do {
    S ← init-state-wl-heur 𝒜in';
    (T::twl-st-wl-heur-init, -) ←  init-dt-wl-heur True CS (S, []);
    T ← rewatch-heur-st-init T;
    let T = convert-state 𝒜in'' T;
    if ¬get-conflict-wl-is-None-heur-init T
    then RETURN (empty-init-code)
    else if CS = [] then empty-conflict-code
    else do {
      ASSERT(𝒜in'' ≠ {#});
      ASSERT(isasat-input-bounded-nempty 𝒜in'');
      T ← finalise-init-code opts (T::twl-st-wl-heur-init);
      U ← cdcl-twl-stgy-restart-prog-wl-heur T;
      RETURN (if get-conflict-wl-is-None-heur U then extract-model-of-state-stat U
        else extract-state-stat U)
    }
}
else do {
    S ← init-state-wl-heur 𝒜in';
    (T::twl-st-wl-heur-init, -) ← init-dt-wl-heur False CS (S, []);
    failed ← RETURN (is-failed-heur-init T ∨ ¬isasat-fast-init T);
    if failed then do {
      S ← init-state-wl-heur 𝒜in';
     (T::twl-st-wl-heur-init, -) ← init-dt-wl-heur True CS (S, []);
      T ← rewatch-heur-st-init T;
      let T = convert-state 𝒜in'' T;
      if ¬get-conflict-wl-is-None-heur-init T
      then RETURN (empty-init-code)
      else if CS = [] then empty-conflict-code
      else do {
       ASSERT(𝒜in'' ≠ {#});
       ASSERT(isasat-input-bounded-nempty 𝒜in'');
       T ← finalise-init-code opts (T::twl-st-wl-heur-init);
       U ← cdcl-twl-stgy-restart-prog-wl-heur T;
       RETURN (if get-conflict-wl-is-None-heur U then extract-model-of-state-stat U
         else extract-state-stat U)
     }
    }
    else do {
      let T = convert-state 𝒜in'' T;
      if ¬get-conflict-wl-is-None-heur-init T
      then RETURN (empty-init-code)
      else if CS = [] then empty-conflict-code
      else do {
        ASSERT(𝒜in'' ≠ {#});
        ASSERT(isasat-input-bounded-nempty 𝒜in'');
        ASSERT(rewatch-heur-st-fast-pre T);
        T ← rewatch-heur-st-init T;
```

```
          ASSERT(isasat-fast-init T);
          T ← finalise-init-code opts (T::twl-st-wl-heur-init);
          ASSERT(isasat-fast T);
          U ← cdcl-twl-stgy-restart-prog-early-wl-heur T;
          RETURN (if get-conflict-wl-is-None-heur U then extract-model-of-state-stat U
            else extract-state-stat U)
        }
      }
    }
  }›
  by (auto simp: init-state-wl-heur-fast-def IsaSAT-heur-def isasat-init-fast-slow-alt-def
    convert-state-def isasat-information-banner-def IsaSAT-Profile.start-def IsaSAT-Profile.stop-def
    cong: if-cong)

abbreviation rewatch-heur-st-rewatch-st-rel where
  ‹rewatch-heur-st-rewatch-st-rel CS U V ≡
    {(S,T). (S, T) ∈ twl-st-heur-parsing (mset-set (extract-atms-clss CS {})) True ∧
        get-clauses-wl-heur-init S = get-clauses-wl-heur-init U ∧
  get-conflict-wl-heur-init S = get-conflict-wl-heur-init U ∧
        get-learned-count-init S = get-learned-count-init U ∧
        get-clauses-wl (fst T) = get-clauses-wl (fst V) ∧
  get-conflict-wl (fst T) = get-conflict-wl (fst V) ∧
  get-subsumed-init-clauses-wl (fst T) = get-subsumed-init-clauses-wl (fst V) ∧
  get-subsumed-learned-clauses-wl (fst T) = get-subsumed-learned-clauses-wl (fst V) ∧
  get-learned-clauses0-wl (fst T) = get-learned-clauses0-wl (fst V) ∧
  get-unkept-unit-init-clss-wl (fst T) = get-unkept-unit-init-clss-wl (fst V) ∧
  get-kept-unit-init-clss-wl (fst T) = get-kept-unit-init-clss-wl (fst V) ∧
  get-unkept-unit-learned-clss-wl (fst T) = get-unkept-unit-learned-clss-wl (fst V) ∧
  get-kept-unit-learned-clss-wl (fst T) = get-kept-unit-learned-clss-wl (fst V) ∧
  get-unit-clauses-wl (fst T) = get-unit-clauses-wl (fst V) ∧
  get-clauses0-wl (fst T) = get-clauses0-wl (fst V)} O {(S, T). S = (T, {#})}›

lemma rewatch-heur-st-rewatch-st:
  assumes
    ‹(x, V)
    ∈ {((S, -), T).
    (S, T)
    ∈ twl-st-heur-parsing-no-WL (mset-set (extract-atms-clss CS {})) True O
    {(S, T). S = remove-watched T ∧ get-watched-wl (fst T) = (λ-. [])}}› and
    ‹x = (U, C)›
  shows ‹rewatch-heur-st-init U ≤
    ⇓(rewatch-heur-st-rewatch-st-rel CS U V)
        (rewatch-st (from-init-state V))›
proof −
  let ?𝒜 = ‹(mset-set (extract-atms-clss CS {}))›
  have UV: ‹(U, V)
      ∈ twl-st-heur-parsing-no-WL (mset-set (extract-atms-clss CS {})) True O
        {(S, T). S = remove-watched T ∧ get-watched-wl (fst T) = (λ-. [])}›
    using assms(1) unfolding assms(2) by simp
  obtain M' arena D' j W' vm φ clvls cach lbd vdom M N D NE UE NS US Q W OC failed N0 U0
NEk UEk where

    V: ‹V = ((M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, Q, W), OC)›
    by (cases V) auto
  let ?vdom = ‹(get-vdom-heur-init U)›
  let ?W' = ‹get-watchlist-wl-heur-init U›
```

```
  let ?arena = ‹get-clauses-wl-heur-init U›
  have valid: ‹valid-arena ?arena N (set ?vdom)› and
    dist: ‹distinct ?vdom› and
    vdom-N: ‹mset ?vdom = dom-m N› and
    watched: ‹(?W', W) ∈ ⟨Id⟩map-fun-rel (D₀ ?𝒜)› and
    lall: ‹literals-are-in-ℒ_in-mm ?𝒜 (mset `# ran-mf N)› and
    vdom: ‹vdom-m ?𝒜 W N ⊆ set-mset (dom-m N)›
    using UV by (auto simp: twl-st-heur-parsing-no-WL-def V distinct-mset-dom
      empty-watched-def vdom-m-def literals-are-in-ℒ_in-mm-def
      all-lits-of-mm-union
      simp flip: distinct-mset-mset-distinct)

  show ?thesis
    using UV
    unfolding rewatch-heur-st-def rewatch-st-def rewatch-heur-st-init-def
    apply (simp only: prod.simps from-init-state-def fst-conv nres-monad1 V)
    apply refine-vcg
    subgoal by (auto simp: twl-st-heur-parsing-no-WL-def dest: valid-arena-vdom-subset)
    apply (rule rewatch-heur-rewatch[OF valid - dist - watched lall])
    subgoal by simp
    subgoal using vdom-N[symmetric] by simp
    subgoal  by (auto simp: vdom-m-def)
    subgoal by (auto simp: V twl-st-heur-parsing-def Collect-eq-comp'
      twl-st-heur-parsing-no-WL-def ac-simps)
    done
qed

lemma rewatch-heur-st-rewatch-st2:
  assumes
    T: ‹(U, V)
    ∈ twl-st-heur-parsing-no-WL (mset-set (extract-atms-clss CS {})) True O
      {(S, T). S = remove-watched T ∧ get-watched-wl (fst T) = (λ-. [])}›
  shows ‹rewatch-heur-st-init
         (convert-state (virtual-copy (mset-set (extract-atms-clss CS {}))) U)
        ≤ ⇓ ({(S,T). (S, T) ∈ twl-st-heur-parsing (mset-set (extract-atms-clss CS {})) True ∧
        get-clauses-wl-heur-init S = get-clauses-wl-heur-init U ∧
  get-conflict-wl-heur-init S = get-conflict-wl-heur-init U ∧
        get-clauses-wl (fst T) = get-clauses-wl (fst V) ∧
  get-conflict-wl (fst T) = get-conflict-wl (fst V) ∧
  get-unit-clauses-wl (fst T) = get-unit-clauses-wl (fst V)} O {(S, T). S = (T, {#})})
          (rewatch-st (from-init-state V))›
proof −
  have
    UV: ‹((U, []), V) ∈ {((S, -), T).
    (S, T)
    ∈ twl-st-heur-parsing-no-WL (mset-set (extract-atms-clss CS {})) True O
    {(S, T). S = remove-watched T ∧ get-watched-wl (fst T) = (λ-. [])}}›
    using T by (auto simp: twl-st-heur-parsing-no-WL-def)
  then show ?thesis
    unfolding convert-state-def finalise-init-def id-def rewatch-heur-st-fast-def
    by (rule rewatch-heur-st-rewatch-st[of ‹(U, [])› V, THEN order-trans])
      (auto intro!: conc-fun-R-mono simp: Collect-eq-comp'
        twl-st-heur-parsing-def)
qed
```

lemma *rewatch-heur-st-rewatch-st3*:
assumes
T: ‹(*U*, *V*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *False O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}› **and**
failed: ‹¬*is-failed-heur-init U*›
shows ‹*rewatch-heur-st-init*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *U*)
≤ ⇓ (*rewatch-heur-st-rewatch-st-rel CS U V*)
(*rewatch-st* (*from-init-state V*))›
proof −
have
UV: ‹((*U*, []), *V*)
∈ {((*S*, -), *T*).
(*S*, *T*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *True O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}}›
using *T failed* **by** (*fastforce simp*: *twl-st-heur-parsing-no-WL-def*)
then show *?thesis*
unfolding *convert-state-def finalise-init-def id-def rewatch-heur-st-fast-def*
by (*rule rewatch-heur-st-rewatch-st*[*of* ‹(*U*, [])› *V*, *THEN order-trans*])
(*auto intro*!: *conc-fun-R-mono simp*: *Collect-eq-comp′*
twl-st-heur-parsing-def)
qed

abbreviation *option-with-bool-rel* :: ‹((*bool* × *′a*) × *′a option*) *set*› **where**
‹*option-with-bool-rel* ≡ {((*b*, *s*), *s′*). (*b* = *is-None s′*) ∧ (¬*b* ⟶ *s* = *the s′*)}›

definition *model-stat-rel* :: ‹((*bool* × *nat literal list* × *′a*) × *nat literal list option*) *set*› **where**
‹*model-stat-rel* = {((*b*, *M′*, *s*), *M*). ((*b*, *rev M′*), *M*) ∈ *option-with-bool-rel*}›

lemma *twl-st-heur-parsing-no-WL-wl-twl-st-heur-parsing-no-WL-init*:
‹*inres* (*init-state-wl-heur* (*mset-set* (*extract-atms-clss CS* {}))) *Sa* ⟹
(*Sa*, *init-state-wl*)
∈ *twl-st-heur-parsing-no-WL-wl* (*mset-set* (*extract-atms-clss CS* {})) *True* ⟹
(*Sa*, *to-init-state init-state-wl*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *True*›
apply (*auto simp*: *twl-st-heur-parsing-no-WL-def*
twl-st-heur-parsing-no-WL-wl-def inres-def to-init-state-def
init-state-wl-def init-state-wl-heur-def)
apply (*auto simp add*: *RES-RES-RETURN-RES*
RES-RETURN-RES)
done

lemma *IsaSAT-heur-IsaSAT*:
‹*IsaSAT-heur b CS* ≤ ⇓*model-stat-rel* (*IsaSAT CS*)›
proof −
have *init-dt-wl-heur*: ‹*init-dt-wl-heur True CS* (*S*, []) ≤
⇓{((*S*, -), *T*). (*S*,*T*) ∈ *twl-st-heur-parsing-no-WL A True O* {(*S*, *T*). *S* = *remove-watched T* ∧
get-watched-wl (*fst T*) = (λ-. [])}}
(*init-dt-wl′ CS T*)›
if
‹*case* (*CS*, *T*) *of*
(*CS*, *S*) ⇒

(∀ C∈*set CS*. *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*))› **and**
‹((*CS*, *S*), *CS*, *T*) ∈ ⟨*Id*⟩*list-rel* ×$_f$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *True*›
for $\mathcal{A}$ *CS T S*
proof −
show *?thesis*
apply (*rule init-dt-wl-heur-init-dt-wl*[*THEN fref-to-Down-curry*, *of* $\mathcal{A}$ *CS T CS* ‹(*S*, [])›, *THEN order-trans*])
apply (*rule that*(*1*))
apply (*use that*(*2*) **in** *auto*; *fail*)
apply (*cases* ‹*init-dt-wl CS T*›)
apply (*force simp*: *init-dt-wl′-def RES-RETURN-RES conc-fun-RES Image-iff*)+
done
qed
have *init-dt-wl-heur-b*: ‹*init-dt-wl-heur False CS* (*S*, []) ≤
⇓{((*S*, -), *T*). (*S*,*T*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *False O* {(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}}
(*init-dt-wl′ CS T*)›
if
‹*case* (*CS*, *T*) *of*
(*CS*, *S*) ⇒
(∀ C∈*set CS*. *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset C*))› **and**
‹((*CS*, *S*), *CS*, *T*) ∈ ⟨*Id*⟩*list-rel* ×$_f$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *True*›
for $\mathcal{A}$ *CS T S*
proof −
show *?thesis*
apply (*rule init-dt-wl-heur-init-dt-wl*[*THEN fref-to-Down-curry*, *of* $\mathcal{A}$ *CS T CS* ‹(*S*, [])›, *THEN order-trans*])
apply (*rule that*(*1*))
using *that*(*2*) **apply** (*force simp*: *twl-st-heur-parsing-no-WL-def*)
apply (*cases* ‹*init-dt-wl CS T*›)
apply (*force simp*: *init-dt-wl′-def RES-RETURN-RES conc-fun-RES Image-iff*)+
done
qed
have *virtual-copy*: ‹(*virtual-copy* $\mathcal{A}$, ()) ∈ {($\mathcal{B}$, *c*). *c* = () ∧ $\mathcal{B}$ = $\mathcal{A}$}› **for** $\mathcal{B}$ $\mathcal{A}$
by (*auto simp*: *virtual-copy-def*)
have *input-le*: ‹∀ C∈*set CS*. ∀ L∈*set C*. *nat-of-lit L* ≤ *unat32-max*›
if ‹*isasat-input-bounded* (*mset-set* (*extract-atms-clss CS* {}))›
proof (*intro ballI*)
fix *C L*
assume ‹*C* ∈ *set CS*› **and** ‹*L* ∈ *set C*›
then have ‹*L* ∈# $\mathcal{L}_{all}$ (*mset-set* (*extract-atms-clss CS* {}))›
by (*auto simp*: *extract-atms-clss-alt-def in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$)
then show ‹*nat-of-lit L* ≤ *unat32-max*›
using *that* **by** *auto*
qed
have *lits-C*: ‹*literals-are-in-*$\mathcal{L}_{in}$ (*mset-set* (*extract-atms-clss CS* {})) (*mset C*)›
if ‹*C* ∈ *set CS*› **for** *C CS*
using *that*
by (*force simp*: *literals-are-in-*$\mathcal{L}_{in}$*-def in-*$\mathcal{L}_{all}$*-atm-of-*$\mathcal{A}_{in}$ *in-all-lits-of-m-ain-atms-of-iff extract-atms-clss-alt-def atm-of-eq-atm-of*)
have *init-state-wl-heur*: ‹*isasat-input-bounded* $\mathcal{A}$ ⟹
init-state-wl-heur $\mathcal{A}$ ≤ *SPEC* (λ*c*. (*c*, *init-state-wl*) ∈
{(*S*, *S′*). (*S*, *S′*) ∈ *twl-st-heur-parsing-no-WL-wl* $\mathcal{A}$ *True*})› **for** $\mathcal{A}$

apply (*rule init-state-wl-heur-init-state-wl*[*THEN fref-to-Down-unRET-uncurry0-SPEC*,
of $\mathcal{A}$, *THEN order-trans*])
apply (*auto*)
done

let *?TT* = ‹*rewatch-heur-st-rewatch-st-rel CS*›
have *get-conflict-wl-is-None-heur-init*: ‹(*Tb*, *Tc*) ∈ *?TT U V* ⟹
(¬ *get-conflict-wl-is-None-heur-init Tb*) = (*get-conflict-wl Tc* ≠ *None*)› **for** *Tb Tc U V*
by (*cases V*; *cases* ‹*get-conflict-wl-heur-init U*›) (*auto simp*: *twl-st-heur-parsing-def Collect-eq-comp'*
get-conflict-wl-is-None-heur-init-def
option-lookup-clause-rel-def)
have *get-conflict-wl-is-None-heur-init3*: ‹(*TC*, *Ta*)
∈ {((*T*,-), *Ta*). (*T*, *Ta*) ∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *False O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}} ⟹
(*failed*, *faileda*)
∈ {(*b*, *b'*). *b* = *b'* ∧ *b* = (*is-failed-heur-init T* ∨ ¬ *isasat-fast-init T*)} ⟹ ¬*failed* ⟹
TC = (*T*, *C*) ⟹
(¬ *get-conflict-wl-is-None-heur-init T*) = (*get-conflict-wl* (*fst Ta*) ≠ *None*)› **for** *TC C T Ta failed faileda*
by (*cases T*; *cases Ta*) (*auto simp*: *twl-st-heur-parsing-no-WL-def*
get-conflict-wl-is-None-heur-init-def
option-lookup-clause-rel-def)

have *banner*: ‹*isasat-information-banner*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *Tb*)
≤ *SPEC* (λ*c*. (*c*, ()) ∈ {(-, -). *True*})› **for** *Tb*
by (*auto simp*: *isasat-information-banner-def*)

have *finalise-init-code*: ‹*finalise-init-code b*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *Tb*)
≤ *SPEC* (λ*c*. (*c*, *finalise-init Tc*) ∈ *twl-st-heur*)› (**is** *?A*) **and**
finalise-init-code3: ‹*finalise-init-code b Tb*
≤ *SPEC* (λ*c*. (*c*, *finalise-init Tc*) ∈ *twl-st-heur*)› (**is** *?B*)
if
T: ‹(*Tb*, *Tc*) ∈ *?TT U V*› **and**
confl: ‹¬ *get-conflict-wl Tc* ≠ *None*› **and**
nempty: ‹*extract-atms-clss CS* {} ≠ {}› **and**
clss-CS: ‹*mset* '# *ran-mf* (*get-clauses-wl Tc*) + *get-unit-clauses-wl Tc* + *get-subsumed-clauses-wl Tc* + *get-clauses0-wl Tc* =
remdups-mset '# *mset* '# *mset CS*› **and**
learned: ‹*learned-clss-l* (*get-clauses-wl Tc*) = {#}›
for *ba S T Ta Tb Tc u v U V*
proof −
have *1*: ‹*get-conflict-wl Tc* = *None*›
using *confl* **by** *auto*

have ‹*set-mset* (*all-atms-st Tc*) ≠ {}›
using *clss-CS nempty*
unfolding *all-atms-st-alt-def all-lits-def all-lits-st-def*
by (*auto simp flip*: *all-atms-def*[*symmetric*] *simp*: *all-lits-def all-atms-st-def*
isasat-input-bounded-nempty-def extract-atms-clss-alt-def ac-simps
all-lits-of-mm-empty-iff)
then have *2*: ‹*all-atms-st Tc* ≠ {#}›
by *auto*
have *3*: ‹*set-mset* (*all-atms-st Tc*) = *set-mset* (*mset-set* (*extract-atms-clss CS* {}))›
using *clss-CS nempty*

unfolding *all-atms-st-alt-def all-lits-def all-lits-st-def*
by (*auto simp flip*: *all-atms-def*[*symmetric*] *all-lits-alt-def simp*: *ac-simps*
isasat-input-bounded-nempty-def
atm-of-all-lits-of-mm extract-atms-clss-alt-def atms-of-ms-def)
have *H*: ‹*A* = *B* ⟹ *x* ∈ *A* ⟹ *x* ∈ *B*› **for** *A B x*
by *auto*
have *H'*: ‹*A* = *B* ⟹ *A x* ⟹ *B x*› **for** *A B x*
by *auto*

note *cong* = *trail-pol-cong heuristic-rel-cong*
option-lookup-clause-rel-cong
vdom-m-cong[*THEN H*] *isasat-input-nempty-cong*[*THEN iffD1*]
isasat-input-bounded-cong[*THEN iffD1*]
cach-refinement-empty-cong[*THEN H'*]
phase-saving-cong[*THEN H'*]
$\mathcal{L}_{all}$*-cong*[*THEN H*]
D_0*-cong*[*THEN H*]
lookup-clause-rel-cong

have *4*: ‹(*convert-state* (*mset-set* (*extract-atms-clss CS* {})) *Tb*, *Tc*)
∈ *twl-st-heur-post-parsing-wl True*›
using *T nempty*
by (*clarsimp simp*: *twl-st-heur-parsing-def twl-st-heur-post-parsing-wl-def*
convert-state-def eq-commute[*of* ‹*mset* -› ‹*dom-m* -›] *all-atms-st-def all-lits-st-alt-def*[*symmetric*]
vdom-m-cong[*OF 3*[*symmetric*]] $\mathcal{L}_{all}$*-cong*[*OF 3*[*symmetric*]] *bump-heur-init-cong*[*OF 3*[*symmetric*]]
dest!: *cong*[*OF 3*[*symmetric*]])
(*simp-all add*: *add.assoc* $\mathcal{L}_{all}$*-all-atms-all-lits*
flip: *all-lits-def all-lits-alt-def2 all-lits-alt-def*)
show *?A*
by (*rule finalise-init-finalise-init*[*THEN fref-to-Down-unRET-curry-SPEC*, *of b*])
(*use 1 2 learned 4* **in** *auto*)
then show *?B* **unfolding** *convert-state-def* **by** *auto*
qed

have *get-conflict-wl-is-None-heur-init2*: ‹(*U*, *V*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *True O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])} ⟹
(¬ *get-conflict-wl-is-None-heur-init*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *U*)) =
(*get-conflict-wl* (*from-init-state V*) ≠ *None*)› **for** *U V*
by (*auto simp*: *twl-st-heur-parsing-def Collect-eq-comp'*
get-conflict-wl-is-None-heur-init-def twl-st-heur-parsing-no-WL-def
option-lookup-clause-rel-def convert-state-def from-init-state-def)

have *rewatch-heur-st-fast-pre*: ‹*rewatch-heur-st-fast-pre*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *T*)›
if
T: ‹(*T*, *Ta*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *True O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])}› **and**
length-le: ‹¬¬*isasat-fast-init* (*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *T*)›
for *uu ba S T Ta baa uua uub*
proof −
have ‹*valid-arena* (*get-clauses-wl-heur-init T*) (*get-clauses-wl* (*fst Ta*))
(*set* (*get-vdom-heur-init T*))›
using *T* **by** (*auto simp*: *twl-st-heur-parsing-no-WL-def*)

```
    then show ?thesis
      using length-le unfolding rewatch-heur-st-fast-pre-def convert-state-def
        isasat-fast-init-def unat64-max-def unat32-max-def
      by (auto dest: valid-arena-in-vdom-le-arena)
  qed
  have rewatch-heur-st-fast-pre2: ‹rewatch-heur-st-fast-pre
  (convert-state (virtual-copy (mset-set (extract-atms-clss CS {}))) T)›
    if
      T: ‹(T, Ta)
       ∈ twl-st-heur-parsing-no-WL (mset-set (extract-atms-clss CS {})) False O
  {(S, T). S = remove-watched T ∧ get-watched-wl (fst T) = (λ-. [])}› and
      length-le: ‹¬¬isasat-fast-init (convert-state (virtual-copy (mset-set (extract-atms-clss CS {}))) T)›
and
      failed: ‹¬is-failed-heur-init T›
    for uu ba S T Ta baa uua uub
  proof −
    have
      Ta: ‹(T, Ta)
     ∈ twl-st-heur-parsing-no-WL (mset-set (extract-atms-clss CS {})) True O
       {(S, T). S = remove-watched T ∧ get-watched-wl (fst T) = (λ-. [])}›
       using failed T by (cases T; cases Ta) (fastforce simp: twl-st-heur-parsing-no-WL-def)
    from rewatch-heur-st-fast-pre[OF this length-le]
    show ?thesis .
  qed
  have finalise-init-code2: ‹finalise-init-code b Tb
≤ SPEC (λc. (c, finalise-init Tc) ∈  {(S′, T′).
            (S′, T′) ∈ twl-st-heur ∧
            get-clauses-wl-heur-init Tb = get-clauses-wl-heur S′ ∧
           get-learned-count-init Tb = get-learned-count S′})›
     (is ‹- ≤ SPEC (λc. - ∈ ?P)›)
  if
    Ta: ‹(TC, Ta)
    ∈ {((T, -), Ta). (T, Ta) ∈ twl-st-heur-parsing-no-WL (mset-set (extract-atms-clss CS {})) False O
      {(S, T). S = remove-watched T ∧ get-watched-wl (fst T) = (λ-. [])}}› and
    confl: ‹¬ get-conflict-wl (from-init-state Ta) ≠ None› and
    ‹CS ≠ []› and
    nempty: ‹extract-atms-clss CS {} ≠ {}› and
    ‹isasat-input-bounded-nempty (mset-set (extract-atms-clss CS {}))› and
    clss-CS: ‹mset `# ran-mf (get-clauses-wl (from-init-state Ta)) +
    get-unit-clauses-wl (from-init-state Ta) + get-subsumed-clauses-wl (from-init-state Ta) +
     get-clauses0-wl (from-init-state Ta) =
     remdups-mset `# mset `# mset CS› and
    learned: ‹learned-clss-l (get-clauses-wl (from-init-state Ta)) = {#}› and
    ‹virtual-copy (mset-set (extract-atms-clss CS {})) ≠ {#}› and
    ‹isasat-input-bounded-nempty
      (virtual-copy (mset-set (extract-atms-clss CS {})))› and
    T: ‹(Tb, Tc) ∈ ?TT T Ta› and
    failed: ‹¬is-failed-heur-init T› and
    TC: ‹TC = (T, C)›
    for uu ba S T Ta baa uua uub V W b Tb Tc TC C
  proof −
    have
    Ta: ‹(T, Ta)
    ∈ twl-st-heur-parsing-no-WL (mset-set (extract-atms-clss CS {})) True O
      {(S, T). S = remove-watched T ∧ get-watched-wl (fst T) = (λ-. [])}›
    using failed Ta unfolding TC by (cases T; cases Ta) (fastforce simp: twl-st-heur-parsing-no-WL-def)
```

have *1*: ‹*get-conflict-wl Tc = None*›
using *confl T* **by** (*auto simp*: *from-init-state-def*)
have *Ta-Tc*: ‹*all-atms-st Tc = all-atms-st* (*from-init-state Ta*)›
using *T Ta*
unfolding *all-lits-alt-def mem-Collect-eq prod.case relcomp.simps*
all-atms-def add.assoc **apply** −
apply *normalize-goal+*
by (*auto simp flip*: *all-atms-def*[*symmetric*] *simp*: *all-atms-st-def all-lits-st-def*
twl-st-heur-parsing-no-WL-def twl-st-heur-parsing-def
from-init-state-def)
moreover have *3*: ‹*set-mset* (*all-atms-st* (*from-init-state Ta*)) = *set-mset* (*mset-set* (*extract-atms-clss CS* {}))›
using *clss-CS*
unfolding *mem-Collect-eq prod.case relcomp.simps all-atms-st-def*
all-atms-st-def all-atms-def all-lits-def
by (*simp add*: *ac-simps extract-atms-clss-alt-def*
atm-of-all-lits-of-mm atms-of-ms-def)
ultimately have *2*: ‹*all-atms-st Tc* ≠ {#}›
using *nempty*
by *auto*
have *3*: ‹*set-mset* (*all-atms-st Tc*) = *set-mset* (*mset-set* (*extract-atms-clss CS* {}))›
unfolding *Ta-Tc 3* **..**

have *H*: ‹*A = B* ⟹ *x* ∈ *A* ⟹ *x* ∈ *B*› **for** *A B x*
by *auto*
have *H′*: ‹*A = B* ⟹ *A x* ⟹ *B x*› **for** *A B x*
by *auto*

note *cong = trail-pol-cong heuristic-rel-cong*
option-lookup-clause-rel-cong
vdom-m-cong[*THEN H*] *isasat-input-nempty-cong*[*THEN iffD1*]
isasat-input-bounded-cong[*THEN iffD1*]
cach-refinement-empty-cong[*THEN H′*]
phase-saving-cong[*THEN H′*]
$\mathcal{L}_{all}$*-cong*[*THEN H*]
D_0*-cong*[*THEN H*]
lookup-clause-rel-cong

have *4*: ‹(*convert-state* (*mset-set* (*extract-atms-clss CS* {})) *Tb*, *Tc*)
∈ *twl-st-heur-post-parsing-wl True*›
using *T nempty*
by (*clarsimp simp*: *twl-st-heur-parsing-def twl-st-heur-post-parsing-wl-def all-atms-st-def*
convert-state-def eq-commute[*of* ‹*mset* ›› ‹*dom-m* ››] *all-lits-st-alt-def*[*symmetric*]
vdom-m-cong[*OF 3*[*symmetric*]] $\mathcal{L}_{all}$*-cong*[*OF 3*[*symmetric*]] *bump-heur-init-cong*[*OF 3*[*symmetric*]]
dest!: *cong*[*OF 3*[*symmetric*]])
(*simp-all add*: *add.assoc* $\mathcal{L}_{all}$*-all-atms-all-lits*
flip: *all-lits-def all-lits-alt-def2 all-lits-alt-def*)

show *?thesis*
apply (*rule finalise-init-finalise-init-full*[*unfolded conc-fun-RETURN*,
THEN order-trans])
by (*use 1 2 learned 4 T* **in** ‹*auto simp*: *from-init-state-def convert-state-def*›)
qed

have *isasat-fast*: ‹*isasat-fast Td*›

```
 if
   fast: ‹¬ ¬ isasat-fast-init
  (convert-state (virtual-copy (mset-set (extract-atms-clss CS {})))
    T)› and
   Tb: ‹(Tb, Tc) ∈ ?TT T Ta› and
   Td: ‹(Td, Te) ∈ (?P Tb)›
  for uu ba S T Ta baa uua uub Tb Tc Td Te
 proof −
  have ‹get-learned-count-init Tb = get-learned-count Td ⟹
   learned-clss-count-init Tb = learned-clss-count Td›
   by (cases Tb; cases Td; auto simp: learned-clss-count-init-def
     learned-clss-count-def clss-size-lcountU0-def clss-size-lcountUEk-def)
   moreover have ‹get-learned-count Td = get-learned-count-init T ⟹
   learned-clss-count Td = learned-clss-count-init T›
   by (cases Td; cases T; auto simp: learned-clss-count-init-def
     learned-clss-count-def clss-size-lcountUS-def clss-size-lcountUE-def
     clss-size-lcount-def clss-size-lcountUEk-def)
   ultimately show ?thesis
    using fast Td Tb unfolding mem-Collect-eq prod.case isasat-fast-init-def
    by (auto simp add: isasat-fast-def
      convert-state-def)
 qed
 define init-succesfull where ‹init-succesfull T = RETURN (is-failed-heur-init T ∨ ¬isasat-fast-init
T)› for T
 define init-succesfull2 where ‹init-succesfull2 = SPEC (λ- :: bool. True)›
 have [refine]: ‹init-succesfull T ≤ ⇓ {(b, b′). (b = b′) ∧ (b ⟷ is-failed-heur-init T ∨ ¬isasat-fast-init
T)}
     init-succesfull2› for T
  by (auto simp: init-succesfull-def init-succesfull2-def intro!: RETURN-RES-refine)
 show ?thesis
  supply [[goals-limit=1]]
  unfolding IsaSAT-heur-alt-def IsaSAT-alt-def init-succesfull-def[symmetric]
 apply (rewrite at ‹do {- ← init-dt-wl′ - -; - ← (¤ :: bool nres); If - - - }› init-succesfull2-def[symmetric])
  apply (refine-vcg virtual-copy init-state-wl-heur banner
    cdcl-twl-stgy-restart-prog-wl-heur-cdcl-twl-stgy-restart-prog-wl-D[THEN fref-to-Down])
  subgoal by (rule input-le)
  subgoal by (rule distinct-mset-mset-set)
  subgoal by auto
  subgoal by auto
  apply (rule init-dt-wl-heur[of ‹mset-set (extract-atms-clss CS {})›])
  subgoal by (auto simp: lits-C)
  subgoal by (simp add: twl-st-heur-parsing-no-WL-wl-twl-st-heur-parsing-no-WL-init)
  apply (rule rewatch-heur-st-rewatch-st)
   apply assumption+
  subgoal unfolding convert-state-def by (rule get-conflict-wl-is-None-heur-init)
  subgoal by (auto simp add: empty-init-code-def model-stat-rel-def)
  subgoal by simp
  subgoal by (auto simp add: empty-conflict-code-def model-stat-rel-def)
  subgoal by (simp add: mset-set-empty-iff extract-atms-clss-alt-def)
  subgoal by simp
  apply (rule finalise-init-code; assumption)
  subgoal by fast
  subgoal by fast
  subgoal premises p for - ba S T Ta Tb Tc u v
   using p(26)
   by (auto simp: twl-st-heur-loop-def get-conflict-wl-is-None-heur-def
```

extract-stats-def extract-state-stat-def
option-lookup-clause-rel-def trail-pol-def
extract-model-of-state-def rev-map
extract-model-of-state-stat-def model-stat-rel-def
dest!: *ann-lits-split-reasons-map-lit-of*)

apply (*rule init-dt-wl-heur-b*[*of* ‹*mset-set* (*extract-atms-clss CS* {})›])
subgoal by (*auto simp*: *lits-C*)
subgoal
by (*simp add*: *twl-st-heur-parsing-no-WL-wl-twl-st-heur-parsing-no-WL-init*)
subgoal by *fast*
apply (*rule init-dt-wl-heur*[*of* ‹*mset-set* (*extract-atms-clss CS* {})›])
subgoal by (*auto simp*: *lits-C*)
subgoal
by (*simp add*: *twl-st-heur-parsing-no-WL-wl-twl-st-heur-parsing-no-WL-init*)
apply (*rule rewatch-heur-st-rewatch-st*; *assumption*)
subgoal unfolding *convert-state-def* **by** (*rule get-conflict-wl-is-None-heur-init*)
subgoal by (*auto simp add*: *empty-init-code-def model-stat-rel-def*)
subgoal by *simp*
subgoal by (*simp add*: *empty-conflict-code-def model-stat-rel-def*)
subgoal by (*simp add*: *mset-set-empty-iff extract-atms-clss-alt-def*)
subgoal by *simp*
apply (*rule finalise-init-code*; *assumption*)
subgoal by *fast*
subgoal by *fast*
subgoal premises *p* **for** - *ba S T Ta Tb Tc u v*
using *p*(*34*)
by (*auto simp*: *twl-st-heur-loop-def get-conflict-wl-is-None-heur-def*
extract-stats-def extract-state-stat-def
option-lookup-clause-rel-def trail-pol-def
extract-model-of-state-def rev-map
extract-model-of-state-stat-def model-stat-rel-def
dest!: *ann-lits-split-reasons-map-lit-of*)
subgoal unfolding *from-init-state-def convert-state-def* **by** (*rule get-conflict-wl-is-None-heur-init3*)
subgoal by (*simp add*: *empty-init-code-def model-stat-rel-def*)
subgoal by *simp*
subgoal by (*simp add*: *empty-conflict-code-def model-stat-rel-def*)
subgoal by (*simp add*: *mset-set-empty-iff extract-atms-clss-alt-def*)
subgoal by (*simp add*: *mset-set-empty-iff extract-atms-clss-alt-def*)
subgoal for *uu ba S T Ta baa uua*
by (*rule rewatch-heur-st-fast-pre2*; *assumption?*) (*simp-all add*: *convert-state-def*)
apply (*rule rewatch-heur-st-rewatch-st3*; *assumption?*)
subgoal by *auto*
subgoal by *auto*
subgoal by (*clarsimp simp add*: *isasat-fast-init-def convert-state-def*
learned-clss-count-init-def)
apply (*rule finalise-init-code2*; *assumption?*)
subgoal by *clarsimp*
subgoal by (*clarsimp simp add*: *isasat-fast-def isasat-fast-init-def convert-state-def ac-simps*
learned-clss-count-init-def learned-clss-count-def)
apply (*rule-tac r1* = ‹*length* (*get-clauses-wl-heur Tc*)› **in** *cdcl-twl-stgy-restart-prog-early-wl-heur-cdcl-twl-stgy-restart-p*
THEN fref-to-Down])
subgoal by (*auto simp*: *isasat-fast-def snat64-max-def unat64-max-def unat32-max-def*)
subgoal by *fast*
subgoal by *fast*

subgoal premises *p* **for** - *ba S T Ta Tb Tc u v*
using *p*(*32*)
by (*auto simp*: *twl-st-heur-loop-def get-conflict-wl-is-None-heur-def*
extract-stats-def extract-state-stat-def
option-lookup-clause-rel-def trail-pol-def
extract-model-of-state-def rev-map
extract-model-of-state-stat-def model-stat-rel-def
dest!: *ann-lits-split-reasons-map-lit-of*)
done
qed

definition *length-get-clauses-wl-heur-init* **where**
‹*length-get-clauses-wl-heur-init S* = *length* (*get-clauses-wl-heur-init S*)›

definition *model-if-satisfiable* :: ‹*nat clauses* ⇒ *nat literal list option nres*› **where**
‹*model-if-satisfiable CS* = *SPEC* (λ*M*.
if satisfiable (*set-mset CS*) *then M* ≠ *None* ∧ *consistent-interp* (*set* (*the M*)) ∧ *set* (*the M*)
⊨*sm CS else M* = *None*)›

definition *SAT'* :: ‹*nat clauses* ⇒ *nat literal list option nres*› **where**
‹*SAT' CS* = *do* {
T ← *SAT CS*;
RETURN (*if pget-conflict T* = *None then Some* (*map lit-of* (*pget-trail T*)) *else None*)
}
›

lemma *SAT-model-if-satisfiable*:
‹(*SAT'*, *model-if-satisfiable*) ∈ [λ*CS*. *True*]$_f$ *Id*→ ⟨*Id*⟩*nres-rel*›
(**is** ‹- ∈[λ*CS*. *?P CS*]$_f$ *Id* → -›)
proof −
have *H*: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-stgy-invariant* (*init-state CS*)›
‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*init-state CS*)›
if ‹*?P CS*› ‹∀ *C*∈#*CS*. *distinct-mset C*› **for** *CS*
using *that* **by** (*auto simp*:
twl-struct-invs-def twl-st-inv.simps cdcl$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*
cdcl$_W$*-restart-mset.no-strange-atm-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-M-level-inv-def*
cdcl$_W$*-restart-mset.distinct-cdcl*$_W$*-state-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-conflicting-def*
cdcl$_W$*-restart-mset.cdcl*$_W$*-learned-clause-alt-def cdcl*$_W$*-restart-mset.no-smaller-propa-def*
past-invs.simps clauses-def twl-list-invs-def twl-stgy-invs-def clause-to-update-def
cdcl$_W$*-restart-mset.cdcl*$_W$*-stgy-invariant-def*
cdcl$_W$*-restart-mset.no-smaller-confl-def*
distinct-mset-set-def)
have *H*: ‹*s* ∈ {*M*. *if satisfiable* (*set-mset CS*) *then M* ≠ *None* ∧ *consistent-interp* (*set* (*the M*)) ∧ *set*
(*the M*) ⊨*sm CS else M* = *None*}›
if
[*simp*]: ‹*CS'* = *CS*› **and**
s: ‹*s* ∈ (λ*T*. *if pget-conflict T* = *None then Some* (*map lit-of* (*pget-trail T*)) *else None*) ‘
Collect (*conclusive-CDCL-state CS'* (*pinit-state CS'*))›
for *s* :: ‹*nat literal list option*› **and** *CS CS'*
proof −
obtain *T* **where**
s: ‹(*s* = *Some* (*map lit-of* (*pget-trail T*)) ∧ *pget-conflict T* = *None*) ∨
(*s* = *None* ∧ *pget-conflict T* ≠ *None*)› **and**
conc: ‹*conclusive-CDCL-state CS'* ([], *CS'*, {#}, *None*, {#}, {#}, {#}, {#}, {#}, {#}) *T*›
using *s* **by** *auto force*

```
    then show ?thesis
      using conc
      by (auto simp: conclusive-CDCL-state-def true-annots-true-cls lits-of-def)
  qed
  show ?thesis
    unfolding SAT'-def model-if-satisfiable-def SAT-def Let-def
    apply (intro frefI nres-relI)
    subgoal for CS' CS
      unfolding RES-RETURN-RES
      apply (rule RES-refine)
      unfolding pair-in-Id-conv bex-triv-one-point1 bex-triv-one-point2
      by (rule H)
    done
qed

lemma SAT-model-if-satisfiable':
  ‹(uncurry (λ-. SAT'), uncurry (λ-. model-if-satisfiable)) ∈
    [λ(-, CS). True]_f Id ×_r Id→ ⟨Id⟩nres-rel›
  using SAT-model-if-satisfiable by (auto simp: fref-def)

definition SAT-l' where
  ‹SAT-l' CS = do{
    S ← SAT-l CS;
    RETURN (if get-conflict-l S = None then Some (map lit-of (get-trail-l S)) else None)
  }›


definition SAT0' where
  ‹SAT0' CS = do{
    S ← SAT0 CS;
    RETURN (if get-conflict S = None then Some (map lit-of (get-trail S)) else None)
  }›


lemma twl-st-l-map-lit-of[twl-st-l, simp]:
  ‹(S, T) ∈ twl-st-l b ⟹ map lit-of (get-trail-l S) = map lit-of (get-trail T)›
  by (auto simp: twl-st-l-def convert-lits-l-map-lit-of)


lemma ISASAT-SAT-l':
  assumes
    ‹isasat-input-bounded (mset-set (⋃C∈set CS. atm-of ‘ set C))›
  shows ‹IsaSAT CS ≤ ⇓ Id (SAT-l' CS)›
  unfolding IsaSAT-def SAT-l'-def
  apply refine-vcg
  apply (rule SAT-wl-SAT-l)
  subgoal using assms by auto
  subgoal by (auto simp: extract-model-of-state-def)
  done

lemma SAT-l'-SAT0':
  shows ‹SAT-l' CS ≤ ⇓ Id (SAT0' CS)›
  unfolding SAT-l'-def SAT0'-def
  apply refine-vcg
  apply (rule SAT-l-SAT0)
  subgoal by (auto simp: extract-model-of-state-def)
```

done

lemma *SAT0′-SAT′*:
shows ‹*SAT0′ CS* ≤ ⇓ *Id* (*SAT′* (*mset* ‘# *mset CS*))›
unfolding *SAT′-def SAT0′-def*
apply *refine-vcg*
apply (*rule SAT0-SAT*)
subgoal by (*auto simp*: *extract-model-of-state-def twl-st-l twl-st*)
done

lemma *IsaSAT-heur-model-if-sat*:
assumes
‹*isasat-input-bounded* (*mset-set* (⋃ *C*∈*set CS*. *atm-of* ‘ *set C*))›
shows ‹*IsaSAT-heur opts CS* ≤ ⇓ *model-stat-rel* (*model-if-satisfiable* (*mset* ‘# *mset CS*))›
apply (*rule IsaSAT-heur-IsaSAT*[*THEN order-trans*])
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule ISASAT-SAT-l′*)
subgoal using *assms* **by** *auto*

unfolding *conc-fun-chain*
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule SAT-l′-SAT0′*)

unfolding *conc-fun-chain*
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule SAT0′-SAT′*)

unfolding *conc-fun-chain*
apply (*rule order-trans*)
apply (*rule ref-two-step′*)
apply (*rule SAT-model-if-satisfiable*[*THEN fref-to-Down*, *of* ‹*mset* ‘# *mset CS*›])
subgoal using *assms* **by** *auto*

unfolding *conc-fun-chain*
apply (*auto simp*: *model-stat-rel-def*)
done

lemma *IsaSAT-heur-model-if-sat′*: ‹(*uncurry IsaSAT-heur*, *uncurry* (λ-. *model-if-satisfiable*)) ∈
[λ(-, *CS*). (∀ *C*∈#*CS*. ∀ *L*∈#*C*. *nat-of-lit L* ≤ *unat32-max*)]$_f$
Id ×$_r$ *list-mset-rel O* ⟨*list-mset-rel*⟩*mset-rel* → ⟨*model-stat-rel*⟩*nres-rel*›
proof −
have *H*: ‹*isasat-input-bounded* (*mset-set* (⋃ *C*∈*set CS*. *atm-of* ‘ *set C*))›
if *CS-p*: ‹∀ *C*∈#*CS′*. ∀ *L*∈#*C*. *nat-of-lit L* ≤ *unat32-max*› **and**
‹(*CS*, *CS′*) ∈ *list-mset-rel O* ⟨*list-mset-rel*⟩*mset-rel*›
for *CS CS′*
unfolding *isasat-input-bounded-def*
proof
fix *L*
assume *L*: ‹*L* ∈# $\mathcal{L}_{all}$ (*mset-set* (⋃ *C*∈*set CS*. *atm-of* ‘ *set C*))›
then obtain *C* **where**
L: ‹*C*∈*set CS* ∧ (*L* ∈*set C* ∨ − *L* ∈ *set C*)›
apply (*cases L*)

```
    apply (auto simp: extract-atms-clss-alt-def unat32-max-def
        \mathcal{L}_{all}-def)+
    apply (metis literal.exhaust-sel)+
    done
  have ‹nat-of-lit L ≤ unat32-max ∨ nat-of-lit (−L) ≤ unat32-max›
    using L CS-p that by (auto simp: list-mset-rel-def mset-rel-def br-def
    br-def mset-rel-def p2rel-def rel-mset-def
      rel2p-def[abs-def] list-all2-op-eq-map-right-iff')
  then show ‹nat-of-lit L ≤ unat32-max›
    using L
    by (cases L) (auto simp: extract-atms-clss-alt-def unat32-max-def)
 qed
 show ?thesis
  apply (intro frefI nres-relI)
  unfolding uncurry-def
  apply clarify
  subgoal for opt1 CS opt2 CS'
  apply (rule IsaSAT-heur-model-if-sat[THEN order-trans, of CS - ‹opt1›])
  subgoal by (rule H) auto
  apply (auto simp: list-mset-rel-def mset-rel-def br-def
    br-def mset-rel-def p2rel-def rel-mset-def
      rel2p-def[abs-def] list-all2-op-eq-map-right-iff')
  done
  done
qed
```

24.3 Refinements of the Whole Bounded SAT Solver

This is the specification of the SAT solver:

```
definition SAT-bounded :: ‹nat clauses ⇒ (bool × nat prag-st) nres› where
 ‹SAT-bounded CS = do{
   T ← SPEC(λT. T = pinit-state (remdups-mset '# CS));
   finished ← SPEC(λ-. True);
   if ¬finished then
     RETURN (True, T)
   else
     SPEC (λ(b, U). ¬b ⟶ conclusive-CDCL-state CS T U)
 }›

definition SAT0-bounded :: ‹nat clause-l list ⇒ (bool × nat twl-st) nres› where
 ‹SAT0-bounded CS = do{
   let (S :: nat twl-st-init) = init-state0;
   T ← SPEC (λT. init-dt-spec0 CS (to-init-state0 S) T);
   finished ← SPEC(λ-. True);
   if ¬finished then do {
     RETURN (True, fst init-state0)
   } else do {
     let T = fst T;
     if get-conflict T ≠ None
     then RETURN (False, T)
     else if CS = [] then RETURN (False, fst init-state0)
     else do {
       ASSERT (extract-atms-clss CS {} ≠ {});
       ASSERT (clauses-to-update T = {#});
```

```
      ASSERT(clause `# (get-clauses T) + unit-clss T + subsumed-clauses T + get-all-clauses0 T =
remdups-mset `# mset  `# mset CS);
      ASSERT(get-learned-clss T = {#});
      cdcl-twl-stgy-restart-prog-bounded T
    }
   }
 }›

lemma SAT0-bounded-SAT-bounded:
 shows ‹SAT0-bounded CS ≤ ⇓ ({((b, S), (b', T)). b = b' ∧ (¬b ⟶ T = pstate_W-of S)}) (SAT-bounded
(mset `# mset CS))›
   (is ‹- ≤ ⇓?A -›)
proof −
 let ?CS = ‹remdups-mset `# mset `# mset CS›
 have conflict-during-init:
   ‹RETURN (False, fst T)
      ≤ ⇓ {((b, S), b', T). b = b' ∧ (¬b ⟶ T = pstate_W-of S)}
        (SPEC (λ(b, U). ¬b ⟶ conclusive-CDCL-state (mset `# mset CS) S U))›
   if
     TS: ‹(T, S)
     ∈ {(S, T).
        (init-dt-spec0 CS (to-init-state0 init-state0) S) ∧
        (T = pinit-state ?CS)}› and
     ‹¬ ¬ failed'› and
     ‹¬ ¬ failed› and
     confl: ‹get-conflict (fst T) ≠ None›
    for bS bT failed T failed' S
 proof −
   have failed[simp]: ‹failed› ‹failed'› ‹failed = True› ‹failed' = True›
     using that
     by auto
   have
     struct-invs: ‹twl-struct-invs-init T› and
     ‹clauses-to-update-init T = {#}› and
     count-dec: ‹∀ s∈set (get-trail-init T). ¬ is-decided s› and
     ‹get-conflict-init T = None ⟶
      literals-to-update-init T =
      uminus `# lit-of `# mset (get-trail-init T)› and
     clss: ‹?CS + clauses (get-init-clauses-init (to-init-state0 init-state0)) +
    other-clauses-init (to-init-state0 init-state0) +
    get-unit-init-clauses-init (to-init-state0 init-state0) +
    get-init-clauses0-init (to-init-state0 init-state0) +
    get-subsumed-init-clauses-init (to-init-state0 init-state0) +
    get-init-clauses0-init (to-init-state0 init-state0) =
    clauses (get-init-clauses-init T) + other-clauses-init T + get-unit-init-clauses-init T +
    get-subsumed-init-clauses-init T +
    get-init-clauses0-init T› and
     learned: ‹get-learned-clauses-init (to-init-state0 init-state0) =
         get-learned-clauses-init T›
       ‹get-unit-learned-clauses-init T =
         get-unit-learned-clauses-init (to-init-state0 init-state0)›
       ‹get-subsumed-learned-clauses-init T =
         get-subsumed-learned-clauses-init (to-init-state0 init-state0)›
       ‹get-learned-clauses0-init T =
         get-learned-clauses0-init (to-init-state0 init-state0)› and
     ‹twl-stgy-invs (fst T)› and
```

```
    ‹other-clauses-init T ≠ {#} ⟶ get-conflict-init T ≠ None› and
    ‹{#} ∈# mset ‘# mset CS ⟶ get-conflict-init T ≠ None› and
    ‹get-conflict-init (to-init-state0 init-state0) ≠ None ⟶
     get-conflict-init (to-init-state0 init-state0) = get-conflict-init T›
    using TS unfolding init-dt-wl-spec-def init-dt-spec0-def
      Set.mem-Collect-eq prod.case failed simp-thms apply −
    apply normalize-goal+
    by metis+

  have count-dec: ‹count-decided (get-trail (fst T)) = 0›
    using count-dec unfolding count-decided-0-iff by (auto simp: twl-st-init
      twl-st-wl-init)

  have le: ‹cdcl_W-restart-mset.cdcl_W-learned-clause (state_W-of-init T)› and
    all-struct-invs:
      ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (state_W-of-init T)›
    using struct-invs unfolding twl-struct-invs-init-def
      cdcl_W-restart-mset.cdcl_W-all-struct-inv-def
      pcdcl-all-struct-invs-def state_W-of-init.simps[symmetric]
    by fast+
  have ‹cdcl_W-restart-mset.cdcl_W-conflicting (state_W-of-init T)›
    using struct-invs unfolding twl-struct-invs-init-def
      cdcl_W-restart-mset.cdcl_W-all-struct-inv-def
      pcdcl-all-struct-invs-def state_W-of-init.simps[symmetric]
    by fast
  have ‹unsatisfiable (set-mset (remdups-mset ‘# mset ‘# mset (rev CS)))›
    using conflict-of-level-unsatisfiable[OF all-struct-invs] count-dec confl
      learned le clss
    by (auto simp: clauses-def mset-take-mset-drop-mset′ twl-st-init twl-st-wl-init
        image-image to-init-state0-def init-state0-def
        cdcl_W-restart-mset.cdcl_W-learned-clauses-entailed-by-init-def ac-simps
twl-st-l-init pget-all-init-clss-pstate_W-of-init)
  then have unsat[simp]: ‹unsatisfiable (mset ‘ set CS)›
    using satisfiable-remdups-iff[of ‹set CS›]
    by auto
  then have [simp]: ‹CS ≠ []›
    by (auto simp del: unsat)
  show ?thesis
    unfolding conclusive-CDCL-state-def
    apply (rule RETURN-SPEC-refine)
    apply (rule exI[of - ‹(False, pstate_W-of (fst T))›])
    apply (intro conjI)
    subgoal
      by auto
    subgoal
      using struct-invs learned count-dec clss confl
      by (clarsimp simp: twl-st-init twl-st-wl-init twl-st-l-init)
    done
qed

have empty-clauses: ‹RETURN (False, fst init-state0)
≤ ⇓ ?A
  (SPEC
    (λ(b, U). ¬b ⟶ conclusive-CDCL-state (mset ‘# mset CS) S U))›
  if
    TS: ‹(T, S)
```

```
    ∈ {(S, T).
      (init-dt-spec0 CS (to-init-state0 init-state0) S) ∧
      (T = pinit-state (?CS))}› and
   [simp]: ‹CS = []›
  for bS bT failed T failed' S
proof −
  let ?CS = ‹mset '# mset CS›
  have [dest]: ‹cdcl_W-restart-mset.cdcl_W ([], {#}, {#}, None) (a, aa, ab, b) ⟹ False›
    for a aa ab b
    by (metis cdcl_W-restart-mset.cdcl_W.cases cdcl_W-restart-mset.cdcl_W-stgy.conflict'
      cdcl_W-restart-mset.cdcl_W-stgy.propagate' cdcl_W-restart-mset.other'
cdcl_W-stgy-cdcl_W-init-state-empty-no-step init-state.simps)
  show ?thesis
    by (rule RETURN-RES-refine, rule exI[of - ‹(False, pinit-state {#})›])
      (use that in ‹auto simp: conclusive-CDCL-state-def init-state0-def›)
qed

have extract-atms-clss-nempty: ‹extract-atms-clss CS {} ≠ {}›
 if
    TS: ‹(T, S)
    ∈ {(S, T).
      (init-dt-spec0 CS (to-init-state0 init-state0) S) ∧
      (T = pinit-state (?CS))}› and
    ‹CS ≠ []› and
    ‹¬get-conflict (fst T) ≠ None›
  for bS bT failed T failed' S
proof −
  show ?thesis
    using that
    by (cases T; cases CS)
      (auto simp: init-state0-def to-init-state0-def init-dt-spec0-def
        extract-atms-clss-alt-def)
qed


have cdcl-twl-stgy-restart-prog: ‹cdcl-twl-stgy-restart-prog-bounded (fst T)
  ≤ ⇓ {((b, S), b', T). b = b' ∧ (¬b ⟶ T = pstate_W-of S)}
    (SPEC (λ(b, U). ¬b ⟶ conclusive-CDCL-state (mset '# mset CS) S U))› (is ?G1)
  if
   bT-bS: ‹(T, S)
    ∈ {(S, T).
      (init-dt-spec0 CS (to-init-state0 init-state0) S) ∧
      (T = pinit-state ?CS)}› and
    ‹CS ≠ []› and
    confl: ‹¬get-conflict (fst T) ≠ None› and
    CS-nempty[simp]: ‹CS ≠ []› and
    ‹extract-atms-clss CS {} ≠ {}› and
    ‹clause '# get-clauses (fst T) + unit-clss (fst T) + subsumed-clauses (fst T) +
      get-all-clauses0 (fst T) = ?CS› and
    ‹get-learned-clss (fst T) = {#}›
  for bS bT failed T failed' S
proof −

  have
    struct-invs: ‹twl-struct-invs-init T› and
    clss-to-upd: ‹clauses-to-update-init T = {#}› and
```

count-dec: ‹∀ *s*∈*set* (*get-trail-init T*). ¬ *is-decided s*› **and**
‹*get-conflict-init T* = *None* ⟶
literals-to-update-init T =
uminus ‘# *lit-of* ‘# *mset* (*get-trail-init T*)› **and**
clss: ‹ *?CS* + *clauses* (*get-init-clauses-init* (*to-init-state0 init-state0*)) +
other-clauses-init (*to-init-state0 init-state0*) +
get-unit-init-clauses-init (*to-init-state0 init-state0*) +
get-init-clauses0-init (*to-init-state0 init-state0*) +
get-subsumed-init-clauses-init (*to-init-state0 init-state0*) +
get-init-clauses0-init (*to-init-state0 init-state0*) =
clauses (*get-init-clauses-init T*) + *other-clauses-init T* + *get-unit-init-clauses-init T* +
get-subsumed-init-clauses-init T +
get-init-clauses0-init T› **and**
learned: ‹*get-learned-clauses-init* (*to-init-state0 init-state0*) =
get-learned-clauses-init T›
‹*get-unit-learned-clauses-init T* =
get-unit-learned-clauses-init (*to-init-state0 init-state0*)›
‹*get-subsumed-learned-clauses-init T* =
get-subsumed-learned-clauses-init (*to-init-state0 init-state0*)›
‹*get-learned-clauses0-init T* =
get-learned-clauses0-init (*to-init-state0 init-state0*)› **and**
stgy-invs: ‹*twl-stgy-invs* (*fst T*)› **and**
oth: ‹*other-clauses-init T* ≠ {#} ⟶ *get-conflict-init T* ≠ *None*› **and**
‹{#} ∈# *mset* ‘# *mset CS* ⟶ *get-conflict-init T* ≠ *None*› **and**
‹*get-conflict-init* (*to-init-state0 init-state0*) ≠ *None* ⟶
get-conflict-init (*to-init-state0 init-state0*) = *get-conflict-init T*›
using *bT-bS* **unfolding** *init-dt-wl-spec-def init-dt-spec0-def*
Set.mem-Collect-eq simp-thms prod.case **apply** −
apply *normalize-goal+*
by *metis+*
have *struct-invs*: ‹*twl-struct-invs* (*fst T*)›
by (*rule twl-struct-invs-init-twl-struct-invs*)
(*use struct-invs oth confl* **in** ‹*auto simp*: *twl-st-init*›)
have *clss-to-upd*: ‹*clauses-to-update* (*fst T*) = {#}›
using *clss-to-upd* **by** (*auto simp*: *twl-st-init*)
have *init*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init*
(*state-of* (*pstate*$_W$*-of* (*fst T*)))›
using *learned*
by (*cases T*)
(*auto simp*: $cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init-def*
to-init-state0-def init-state0-def get-all-init-clss-alt-init-def)

have *If-RES-RES*: ‹*If b* (*RES* (*A*)) (*RES* (*B*)) = *RES* ({*x*. (*b* ⟶ *x* ∈ *A*) ∧ (¬*b* ⟶ *x* ∈ *B*)})›
for *A B b*
by *auto*
have *init*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init*
(*state-of* (*pstate*$_W$*-of* (*fst T*)))›
using *learned*
by (*cases T*)
(*auto simp*:
to-init-state0-def init-state0-def get-all-init-clss-alt-init-def
conclusive-CDCL-state-def
$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init-def*)

have *conclusive-le*: ‹*conclusive-TWL-run-bounded* (*fst T*)

```
  ≤ ⇓ {((b, S), b', T). b = b' ∧ (¬b ⟶ T = pstate_W-of S)}
    (SPEC (λ(b, U). ¬b ⟶ conclusive-CDCL-state (mset `# mset CS) S U))›
    apply (rule order-trans[of -
      ‹do {
      b ← SPEC (λ- :: bool. True);
    if ¬b then do {T ← conclusive-TWL-run (fst T); RETURN (False, T)}
      else  do {
      T ← SPEC(λTa. ∃ R1 R2 m m_0 n_0.
      cdcl-twl-stgy-restart** (fst T, fst T, fst T, m_0, n_0, True) (R1, R2, Ta, m));
      RETURN (True, T)
      }
      }›])
    subgoal
      unfolding partial-conclusive-TWL-run-def conclusive-TWL-run-def RES-RETURN-RES
        If-RES-RES RES-RES-RETURN-RES less-eq-nres.simps
      apply (rule)
      apply (drule Collect-case-prodD)
      apply normalize-goal+
      apply (rule-tac a = ‹fst x› in UN-I)
      apply simp
      apply (fastforce simp: pair-in-image-Pair)
      done
    subgoal
      apply (refine-vcg
        lhs-step-If)
      subgoal
        using satisfiable-remdups-iff[of ‹set CS›]
        apply −
        unfolding conc-fun-RES
        apply (rule RETURN-le-RES-no-return)
        apply (rule conclusive-TWL-run-conclusive-CDCL-state[THEN order-trans])
        using clss arg-cong[OF clss, of set-mset, simplified] bT-bS confl oth init
        by (auto simp: br-def conc-fun-RES struct-invs stgy-invs
            to-init-state0-def init-state0-def get-all-init-clss-alt-init-def
            conclusive-CDCL-state-def image-image
            true-annots-remdups-mset[symmetric, of - ‹mset ` set CS›])
      subgoal
        by (intro RETURN-RES-refine)
          (auto simp: conclusive-CDCL-state-def)
        done
      done
    show ?G1
      apply (rule cdcl-twl-stgy-restart-prog-bounded-spec[THEN order-trans])
          apply (rule struct-invs; fail)
         apply (rule stgy-invs; fail)
        apply (rule clss-to-upd; fail)
       apply (use confl in ‹simp add: twl-st-init›; fail)
      apply (rule init[unfolded state_W-of-def[symmetric]]; fail)
     apply (rule conclusive-le)
     done
  qed
```

The following does not relate anything, because the initialisation is already doing some steps.

```
  have [refine0]:
    ‹SPEC
     (λT.  init-dt-spec0 CS (to-init-state0 init-state0) T)
```

```
    ≤ ⇓ {(S, T).
          (init-dt-spec0 CS (to-init-state0 init-state0) S) ∧
          (T = pinit-state ?CS)}
       (SPEC (λT. T = pinit-state ?CS))›
    by (rule RES-refine)
     (auto simp: init-state0-def to-init-state0-def
        extract-atms-clss-alt-def intro!: )[]
  show ?thesis
    unfolding SAT0-bounded-def SAT-bounded-def
    apply (subst Let-def)
    apply (refine-vcg)
    subgoal by (auto simp: RETURN-def intro!: RES-refine)
    subgoal by (auto simp: RETURN-def intro!: RES-refine)
    apply (rule lhs-step-If)
    subgoal
      by (rule conflict-during-init)
    apply (rule lhs-step-If)
    subgoal
      by (rule empty-clauses) assumption+
    apply (intro ASSERT-leI)
    subgoal for b T
      by (rule extract-atms-clss-nempty)
    subgoal for S T
      by (cases S)
       (auto simp: init-state0-def to-init-state0-def init-dt-spec0-def
         extract-atms-clss-alt-def)
    subgoal for S T
      by (cases S)
       (auto simp: init-state0-def to-init-state0-def init-dt-spec0-def
         extract-atms-clss-alt-def)
    subgoal for S T
      by (cases S)
       (auto simp: init-state0-def to-init-state0-def init-dt-spec0-def
         extract-atms-clss-alt-def)
    subgoal for S T
      by (rule cdcl-twl-stgy-restart-prog)
    done
qed

definition SAT-l-bounded :: ‹nat clause-l list ⇒ (bool × nat twl-st-l) nres› where
  ‹SAT-l-bounded CS = do{
     let S = init-state-l;
     T ← init-dt CS (to-init-state-l S);
     finished ← SPEC (λ- :: bool. True);
     if ¬finished then do {
       RETURN (True, fst init-state-l)
     } else do {
       let T = fst T;
       if get-conflict-l T ≠ None
       then RETURN (False, T)
       else if CS = [] then RETURN (False, fst init-state-l)
       else do {
         ASSERT (extract-atms-clss CS {} ≠ {});
         ASSERT (clauses-to-update-l T = {#});
         ASSERT(mset `# ran-mf (get-clauses-l T) + get-unit-clauses-l T +
          get-subsumed-clauses-l T + get-clauses0-l T = remdups-mset `# mset `# mset CS);
```

```
        ASSERT(learned-clss-l (get-clauses-l T) = {#});
        cdcl-twl-stgy-restart-prog-bounded-l T
      }
    }
  }›

lemma SAT-l-bounded-SAT0-bounded:
 shows ‹SAT-l-bounded CS ≤ ⇓ {((b, T),(b', T')). b=b' ∧ (¬b ⟶ (T, T') ∈ twl-st-l None)} (SAT0-bounded
CS)›
    (is ‹- ≤ ⇓ ?A -›)
proof −
  have inj: ‹inj (uminus :: - literal ⇒ -)›
    by (auto simp: inj-on-def)
  have [simp]: ‹{#− lit-of x. x ∈# A#} = {#− lit-of x. x ∈# B#} ⟷
    {#lit-of x. x ∈# A#} = {#lit-of x. x ∈# B#}› for A B :: ‹(nat literal, nat literal,
          nat) annotated-lit multiset›
    unfolding multiset.map-comp[unfolded comp-def, symmetric]
    apply (subst inj-image-mset-eq-iff[of uminus])
    apply (rule inj)
    by (auto simp: inj-on-def)[]
  have get-unit-twl-st-l: ‹(s, x) ∈ twl-st-l-init ⟹ get-learned-unit-clauses-l-init s = {#} ⟹
     learned-clss-l (get-clauses-l-init s) = {#} ⟹
    {#mset (fst x). x ∈# ran-m (get-clauses-l-init s)#} +
    (get-unit-clauses-l-init s + get-subsumed-init-clauses-l-init s) =
    clause `# get-init-clauses-init x + (get-unit-init-clauses-init x +
      get-subsumed-init-clauses-init x)› for s x
    apply (cases s; cases x)
    apply (auto simp: twl-st-l-init-def mset-take-mset-drop-mset')
    by (metis (mono-tags, lifting) add.right-neutral all-clss-l-ran-m)

  have init-dt-pre: ‹init-dt-pre CS (to-init-state-l init-state-l)›
    by (rule init-dt-pre-init)

  have init-dt-spec0: ‹init-dt CS (to-init-state-l init-state-l)
      ≤ ⇓{((T),T'). (T, T') ∈ twl-st-l-init ∧ twl-list-invs (fst T) ∧
           clauses-to-update-l (fst T) = {#}}
         (SPEC (init-dt-spec0 CS (to-init-state0 init-state0)))›
    apply (rule init-dt-full[THEN order-trans])
    subgoal by (rule init-dt-pre)
    subgoal
      apply (rule RES-refine)
      unfolding init-dt-spec-def Set.mem-Collect-eq init-dt-spec0-def
        to-init-state-l-def init-state-l-def
        to-init-state0-def init-state0-def
      apply normalize-goal+
      apply (rule-tac x=x in bexI)
      subgoal for s x by (auto simp: twl-st-l-init)
      subgoal for s x
        unfolding Set.mem-Collect-eq
        by (simp-all add: twl-st-init twl-st-l-init twl-st-l-init-no-decision-iff get-unit-twl-st-l)
      done
    done

  have init-state0: ‹ ((False, fst init-state-l), False, fst init-state0)
    ∈ ?A›
    by (auto simp: twl-st-l-def init-state0-def init-state-l-def)
```

```
  show ?thesis
    unfolding SAT-l-bounded-def SAT0-bounded-def
    apply (refine-vcg init-dt-spec0)
    subgoal by auto
    subgoal by (auto simp: twl-st-l-init twl-st-init)
    subgoal by (auto simp: twl-st-l-init-alt-def simp del: twl-st-init(42)
      simp flip: twl-st-init(42))
    subgoal by (auto simp: twl-st-l-init-alt-def)
    subgoal by auto
    subgoal by (rule init-state0)
    subgoal for b ba T Ta
      unfolding all-clss-lf-ran-m[symmetric] image-mset-union to-init-state0-def init-state0-def
      by (cases T; cases Ta)
        (auto simp: twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset'
          init-dt-spec0-def)
    subgoal for b ba T Ta
      unfolding all-clss-lf-ran-m[symmetric] image-mset-union
     by (cases T; cases Ta) (auto simp: twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset',
        auto simp: ac-simps)
    subgoal for T Ta finished finisheda
     by (cases T; cases Ta) (auto simp: twl-st-l-init twl-st-init twl-st-l-init-def mset-take-mset-drop-mset')
    subgoal for T Ta finished finisheda
      by (rule cdcl-twl-stgy-restart-prog-bounded-l-cdcl-twl-stgy-restart-prog-bounded[THEN fref-to-Down,
of - ‹fst Ta›,
          THEN order-trans])
        (auto simp: twl-st-l-init-alt-def mset-take-mset-drop-mset' intro!: conc-fun-R-mono)
    done
qed

definition SAT-wl-bounded :: ‹nat clause-l list ⇒ (bool × nat twl-st-wl) nres› where
  ‹SAT-wl-bounded CS = do{
    ASSERT(isasat-input-bounded (mset-set (extract-atms-clss CS {})));
    let 𝒜_in' = extract-atms-clss CS {};
    let S = init-state-wl;
    T ← init-dt-wl' CS (to-init-state S);
    let T = from-init-state T;
    finished ← SPEC (λ- :: bool. True);
    if ¬finished then do {
        RETURN(¬finished, T)
    } else do {
      if get-conflict-wl T ≠ None
      then RETURN (False, T)
       else if CS = [] then RETURN (False, ([], fmempty, None, {#}, {#},{#}, {#}, {#}, {#}, {#},
{#}, {#}, λ-. undefined))
      else do {
        ASSERT (extract-atms-clss CS {} ≠ {});
        ASSERT(isasat-input-bounded-nempty (mset-set 𝒜_in'));
        ASSERT(mset `# ran-mf (get-clauses-wl T) + get-unit-clauses-wl T +
          get-subsumed-clauses-wl T + get-clauses0-wl T = remdups-mset `# mset `# mset CS);
        ASSERT(learned-clss-l (get-clauses-wl T) = {#});
        T ← rewatch-st (finalise-init T);
        cdcl-twl-stgy-restart-prog-bounded-wl T
      }
    }
  }›
```

lemma *SAT-l-bounded-alt-def*:
  ‹*SAT-l-bounded CS* = *do*{
    $\mathcal{A} \leftarrow$ *RETURN* (); ~~////////~~
    let S = *init-state-l*;
    $\mathcal{A} \leftarrow$ *RETURN* (); ~~//////////~~
    T ← *init-dt CS* (*to-init-state-l S*);
    failed ← *SPEC* (λ- :: *bool*. *True*);
    if ¬*failed then do* {
      RETURN(¬*failed*, *fst init-state-l*)
    } *else do* {
      let T = *T*;
      if get-conflict-l-init T ≠ *None*
      then RETURN (*False*, *fst T*)
      else if CS = [] *then RETURN* (*False*, *fst init-state-l*)
      else do {
        ASSERT (*extract-atms-clss CS* {} ≠ {});
        ASSERT (*clauses-to-update-l* (*fst T*) = {#});
        ASSERT(*mset* ‘# *ran-mf* (*get-clauses-l* (*fst T*)) + *get-unit-clauses-l* (*fst T*) +
          get-subsumed-clauses-l (*fst T*) + *get-clauses0-l* (*fst T*)= *remdups-mset* ‘# *mset* ‘# *mset CS*);
        ASSERT(*learned-clss-l* (*get-clauses-l* (*fst T*)) = {#});
        let T = *fst T*;
        cdcl-twl-stgy-restart-prog-bounded-l T
      }
    }
  }›
  unfolding *SAT-l-bounded-def* **by** (*auto cong*: *if-cong Let-def twl-st-l-init*)

lemma *SAT-wl-bounded-SAT-l-bounded*:
  assumes
    bounded: ‹*isasat-input-bounded* (*mset-set* ($\bigcup C \in$*set CS*. *atm-of* ‘ *set C*))›
  shows ‹*SAT-wl-bounded CS* ≤ ⇓ {((*b*, *T*),(*b*′, *T*′)). *b* =*b*′ ∧ (¬*b* ⟶ (*T*, *T*′) ∈ *state-wl-l None*)}
(*SAT-l-bounded CS*)›
proof −
  have *extract-atms-clss*: ‹(*extract-atms-clss CS* {}, ()) ∈ {(*x*, -). *x* = *extract-atms-clss CS* {}}›
    by *auto*
  have *init-dt-wl-pre*: ‹*init-dt-wl-pre CS* (*to-init-state init-state-wl*)›
    by (*rule init-dt-wl-pre*)

  have *init-rel*: ‹(*to-init-state init-state-wl*, *to-init-state-l init-state-l*)
    ∈ *state-wl-l-init*›
    by (*auto simp*: *init-dt-pre-def state-wl-l-init-def state-wl-l-init′-def*
      twl-st-l-init-def twl-init-invs to-init-state-def init-state-wl-def
      init-state-l-def to-init-state-l-def)[]

  — The following stlightly strange theorem allows to reuse the definition and the correctness of *init-dt-wl-heur-full*, which was split in the definition for purely refinement-related reasons.
  define *init-dt-wl-rel* **where**
    ‹*init-dt-wl-rel S* ≡ ({(*T*, *T*′). *RETURN T* ≤ *init-dt-wl*′ *CS S* ∧ *T*′ = ()})› **for** *S*
  have *init-dt-wl*′:
    ‹*init-dt-wl*′ *CS S* ≤ *SPEC* (λ*c*. (*c*, ()) ∈ (*init-dt-wl-rel S*))›
    if
      ‹*init-dt-wl-pre CS S*› **and**
      ‹(*S*, *S*′) ∈ *state-wl-l-init*›
      for *S S*′

proof −
have [*simp*]: ‹(*U*, *U'*) ∈ ({(*T*, *T'*). *RETURN T* ≤ *init-dt-wl' CS S* ∧ *remove-watched T* = *T'*} *O*
state-wl-l-init) ⟷ ((*U*, *U'*) ∈ {(*T*, *T'*). *remove-watched T* = *T'*} *O*
state-wl-l-init ∧ *RETURN U* ≤ *init-dt-wl' CS S*)› **for** *S S' U U'*
by *auto*
have *H*: ‹*A* ≤ ⇓ ({(*S*, *S'*). *P S S'*}) *B* ⟷ *A* ≤ ⇓ ({(*S*, *S'*). *RETURN S* ≤ *A* ∧ *P S S'*}) *B*›
for *A B P R*
by (*simp add*: *pw-conc-inres pw-conc-nofail pw-le-iff p2rel-def*)
have *nofail*: ‹*nofail* (*init-dt-wl' CS S*)›
apply (*rule SPEC-nofail*)
apply (*rule order-trans*)
apply (*rule init-dt-wl'-spec*[*unfolded conc-fun-RES*])
using *that* **by** *auto*
have *H*: ‹*A* ≤ ⇓ ({(*S*, *S'*). *P S S'*} *O R*) *B* ⟷ *A* ≤ ⇓ ({(*S*, *S'*). *RETURN S* ≤ *A* ∧ *P S S'*} *O*
R) *B*›
for *A B P R*
by (*smt Collect-cong H case-prod-cong conc-fun-chain*)
show *?thesis*
unfolding *init-dt-wl-rel-def*
using *that*
by (*auto simp*: *nofail no-fail-spec-le-RETURN-itself*)
qed

have *conflict-during-init*:
‹((*False*, ([]), *fmempty*, *None*, {#}, {#}, {#}, {#}, {#}, {#}, {#}, {#}, λ-. *undefined*)),
(*False*, *fst init-state-l*))
∈ {((*b*, *T*), *b'*, *T'*). *b* = *b'* ∧ (¬*b* ⟶ (*T*, *T'*) ∈ *state-wl-l None*)}›
by (*auto simp*: *init-state-l-def state-wl-l-def*)

have *init-init-dt*: ‹*RETURN* (*from-init-state T*)
≤ ⇓ ({(*S*, *S'*). *S* = *fst S'*} *O* {(*S* :: *nat twl-st-wl-init-full*, *S'* :: *nat twl-st-wl-init*).
remove-watched S = *S'*} *O state-wl-l-init*)
(*init-dt CS* (*to-init-state-l init-state-l*))›
(**is** ‹- ≤ ⇓ *?init-dt* - ›)
if
‹(*extract-atms-clss CS* {}, (*A*::*unit*)) ∈ {(*x*, -). *x* = *extract-atms-clss CS* {}}› **and**
‹(*T*, *Ta*) ∈ *init-dt-wl-rel* (*to-init-state init-state-wl*)›
for *A T Ta*
proof −
have *1*: ‹*RETURN T* ≤ *init-dt-wl' CS* (*to-init-state init-state-wl*)›
using *that* **by** (*auto simp*: *init-dt-wl-rel-def from-init-state-def*)
have *2*: ‹*RETURN* (*from-init-state T*) ≤ ⇓ {(*S*, *S'*). *S* = *fst S'*} (*RETURN T*)›
by (*auto simp*: *RETURN-refine from-init-state-def*)
have *2*: ‹*RETURN* (*from-init-state T*) ≤ ⇓ {(*S*, *S'*). *S* = *fst S'*} (*init-dt-wl' CS* (*to-init-state*
init-state-wl))›
apply (*rule 2*[*THEN order-trans*])
apply (*rule ref-two-step'*)
apply (*rule 1*)
done
show *?thesis*
apply (*rule order-trans*)
apply (*rule 2*)
unfolding *conc-fun-chain*[*symmetric*]
apply (*rule ref-two-step'*)
unfolding *conc-fun-chain*
apply (*rule init-dt-wl'-init-dt*)

apply (*rule init-dt-wl-pre*)
subgoal by (*auto simp*: *to-init-state-def init-state-wl-def to-init-state-l-def*
init-state-l-def state-wl-l-init-def state-wl-l-init'-def)
done
qed

have *cdcl-twl-stgy-restart-prog-wl-D2*: ‹*cdcl-twl-stgy-restart-prog-bounded-wl U'*
≤ ⇓ {((*b*, *T*), (*b'*, *T'*)). *b* = *b'* ∧ (¬*b* ⟶ (*T*, *T'*) ∈ *state-wl-l None*)}
(*cdcl-twl-stgy-restart-prog-bounded-l* (*fst T'*))› (**is** *?A*)
if
U': ‹(*U'*, *fst T'*) ∈ {(*S*, *T*). (*S*, *T*) ∈ *state-wl-l None* ∧ *correct-watching S* ∧ *blits-in-*$\mathcal{L}_{in}$ *S*}›
for $\mathcal{A}$ *b b' T* $\mathcal{A}'$ *T' c c' U'*
proof −
have *1*: ‹ {(*T*, *T'*). (*T*, *T'*) ∈ *state-wl-l None*} = *state-wl-l None*›
by *auto*
have *lits*: ‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st* (*U'*)) (*U'*)›
using *U'* **by** (*auto simp*: *finalise-init-def correct-watching.simps*)
show *?A*
apply (*rule cdcl-twl-stgy-restart-prog-bounded-wl-spec*[*unfolded fref-param1*, *THEN fref-to-Down*,
THEN order-trans])
apply *fast*
using *U'* **by** (*auto simp*: *finalise-init-def intro*!: *conc-fun-R-mono*)

qed

have *rewatch-st-fst*: ‹*rewatch-st* (*finalise-init* (*from-init-state T*))
≤ *SPEC* (λ*c*. (*c*, *fst Ta*) ∈ {(*S*, *T*). (*S*, *T*) ∈ *state-wl-l None* ∧ *correct-watching S* ∧ *blits-in-*$\mathcal{L}_{in}$ *S*})›
(**is** ‹- ≤ *SPEC ?rewatch*›)
if

‹(*extract-atms-clss CS* {}, $\mathcal{A}$) ∈ {(*x*, -). *x* = *extract-atms-clss CS* {}}› **and**
T: ‹(*T*, $\mathcal{A}'$) ∈ *init-dt-wl-rel* (*to-init-state init-state-wl*)› **and**
T-Ta: ‹(*from-init-state T*, *Ta*)
∈ {(*S*, *S'*). *S* = *fst S'*} *O*
{(*S*, *S'*). *remove-watched S* = *S'*} *O state-wl-l-init*› **and**
‹¬ *get-conflict-wl* (*from-init-state T*) ≠ *None*› **and**
‹¬ *get-conflict-l-init Ta* ≠ *None*›
for $\mathcal{A}$ *b ba T* $\mathcal{A}'$ *Ta bb bc*
proof −
have *1*: ‹*RETURN T* ≤ *init-dt-wl' CS* (*to-init-state init-state-wl*)›
using *T* **unfolding** *init-dt-wl-rel-def* **by** *auto*
have *2*: ‹*RETURN T* ≤ ⇓ {(*S*, *S'*). *remove-watched S* = *S'*}
(*SPEC* (*init-dt-wl-spec CS* (*to-init-state init-state-wl*)))›
using *order-trans*[*OF 1 init-dt-wl'-spec*[*OF init-dt-wl-pre*]] **.**

have *empty-watched*: ‹*get-watched-wl* (*finalise-init* (*from-init-state T*)) = (λ-. [])›
using *1 2 init-dt-wl'-spec*[*OF init-dt-wl-pre*]
by (*cases T*; *cases* ‹*init-dt-wl CS* (*init-state-wl*, {#})›)
(*auto simp*: *init-dt-wl-spec-def RETURN-RES-refine-iff*
finalise-init-def from-init-state-def state-wl-l-init-def
state-wl-l-init'-def to-init-state-def to-init-state-l-def
init-state-l-def init-dt-wl'-def RES-RETURN-RES)

have *1*: ‹*length* (*aa* ∝ *x*) ≥ *2*› ‹*distinct* (*aa* ∝ *x*)›
if

```
      struct: ‹twl-struct-invs-init
        ((af,
        {#TWL-Clause (mset (watched-l (fst x))) (mset (unwatched-l (fst x)))
        . x ∈# init-clss-l aa#},
        {#}, y, ac, {#}, NS, US, N0, U0, {#}, ae),
        OC)› and
x: ‹x ∈# dom-m aa› and
learned: ‹learned-clss-l aa = {#}›
for af aa y ac ae x OC NS US N0 U0
  proof −
    have irred: ‹irred aa x›
      using that by (cases ‹fmlookup aa x›) (auto simp: ran-m-def dest!: multi-member-split
 split: if-splits)
    have ‹Multiset.Ball
({#TWL-Clause (mset (watched-l (fst x))) (mset (unwatched-l (fst x)))
 . x ∈# init-clss-l aa#} +
 {#})
struct-wf-twl-cls›
using struct unfolding twl-struct-invs-init-def fst-conv twl-st-inv.simps
by fast
    then show ‹length (aa ∝ x) ≥ 2› ‹distinct (aa ∝ x)›
      using x learned in-ran-mf-clause-inI[OF x, of True] irred
by (auto simp: mset-take-mset-drop-mset′ dest!: multi-member-split[of x]
 split: if-splits)
  qed
  have min-len: ‹x ∈# dom-m (get-clauses-wl (finalise-init (from-init-state T))) ⟹
    distinct (get-clauses-wl (finalise-init (from-init-state T)) ∝ x) ∧
    2 ≤ length (get-clauses-wl (finalise-init (from-init-state T)) ∝ x)›
    for x
    using 2
    by (cases T)
     (auto simp: init-dt-wl-spec-def RETURN-RES-refine-iff
      finalise-init-def from-init-state-def state-wl-l-init-def
state-wl-l-init′-def to-init-state-def to-init-state-l-def
      init-state-l-def init-dt-wl′-def RES-RETURN-RES
      init-dt-spec-def init-state-wl-def twl-st-l-init-def
      intro: 1)

  show ?thesis
    apply (rule rewatch-st-correctness[THEN order-trans])
    subgoal by (rule empty-watched)
    subgoal by (rule min-len)
    subgoal using T-Ta by (auto simp: finalise-init-def
       state-wl-l-init-def state-wl-l-init′-def state-wl-l-def
 correct-watching-init-correct-watching
 correct-watching-init-blits-in-\mathcal{L}_{in})
    done
qed

have all-le: ‹∀ C∈set CS. ∀ L∈set C. nat-of-lit L ≤ unat32-max›
proof (intro ballI)
  fix C L
  assume ‹C ∈ set CS› and ‹L ∈ set C›
  then have ‹L ∈# \mathcal{L}_{all} (mset-set (⋃ C∈set CS. atm-of ‘ set C))›
    by (auto simp: in-\mathcal{L}_{all}-atm-of-\mathcal{A}_{in})
  then show ‹nat-of-lit L ≤ unat32-max›
```

```
    using assms by auto
  qed
  have [simp]: ‹(Tc, fst Td) ∈ state-wl-l None ⟹
      get-conflict-l-init Td = get-conflict-wl Tc› for Tc Td
   by (cases Tc; cases Td; auto simp: state-wl-l-def)
  show ?thesis
    unfolding SAT-wl-bounded-def SAT-l-bounded-alt-def
    apply (refine-vcg extract-atms-clss init-dt-wl' init-rel)
    subgoal using assms unfolding extract-atms-clss-alt-def by auto
    subgoal by (rule init-dt-wl-pre)
    apply (rule init-init-dt; assumption)
    subgoal by auto
    subgoal by auto
    subgoal by (auto simp: from-init-state-def state-wl-l-init-def state-wl-l-init'-def)
    subgoal by (auto simp: from-init-state-def state-wl-l-init-def state-wl-l-init'-def
      state-wl-l-def)
    subgoal by auto
    subgoal by (rule conflict-during-init)
    subgoal using bounded by (auto simp: isasat-input-bounded-nempty-def extract-atms-clss-alt-def
      simp del: isasat-input-bounded-def)
    subgoal by (auto simp: isasat-input-bounded-nempty-def extract-atms-clss-alt-def state-wl-l-init'-def
      state-wl-l-init-def
      simp del: isasat-input-bounded-def)
    subgoal by (auto simp: isasat-input-bounded-nempty-def extract-atms-clss-alt-def state-wl-l-init'-def
      state-wl-l-init-def
      simp del: isasat-input-bounded-def)
    apply (rule rewatch-st-fst; assumption)
    subgoal for A T A' Ta finished finished'
      unfolding twl-st-l-init[symmetric]
      by (rule cdcl-twl-stgy-restart-prog-wl-D2)
    done
qed


definition SAT-bounded' :: ‹nat clauses ⇒ (bool × nat literal list option) nres› where
  ‹SAT-bounded' CS = do {
    (b, T) ← SAT-bounded CS;
    RETURN(b, if pget-conflict T = None then Some (map lit-of (pget-trail T)) else None)
  }
›

definition model-if-satisfiable-bounded :: ‹nat clauses ⇒ (bool × nat literal list option) nres› where
  ‹model-if-satisfiable-bounded CS = SPEC (λ(b, M). ¬b ⟶
        (if satisfiable (set-mset CS) then M ≠ None ∧ set (the M) ⊨sm CS else M = None))›


lemma SAT-bounded-model-if-satisfiable:
  ‹(SAT-bounded', model-if-satisfiable-bounded) ∈ [λCS. True]_f Id→
     ⟨{((b, S), (b', T)). b = b' ∧ (¬b ⟶ S = T)}⟩nres-rel›
   (is ‹- ∈[λCS. ?P CS]_f Id → -›)
proof −
  have H: ‹cdcl_W-restart-mset.cdcl_W-stgy-invariant (init-state CS)›
    ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (init-state CS)›
    if ‹?P CS› ‹∀ C∈#CS. distinct-mset C› for CS
    using that by (auto simp:
      twl-struct-invs-def twl-st-inv.simps cdcl_W-restart-mset.cdcl_W-all-struct-inv-def
```

cdcl$_W$-restart-mset.no-strange-atm-def cdcl$_W$-restart-mset.cdcl$_W$-M-level-inv-def
cdcl$_W$-restart-mset.distinct-cdcl$_W$-state-def cdcl$_W$-restart-mset.cdcl$_W$-conflicting-def
cdcl$_W$-restart-mset.cdcl$_W$-learned-clause-alt-def cdcl$_W$-restart-mset.no-smaller-propa-def
past-invs.simps clauses-def twl-list-invs-def twl-stgy-invs-def clause-to-update-def
cdcl$_W$-restart-mset.cdcl$_W$-stgy-invariant-def
cdcl$_W$-restart-mset.no-smaller-confl-def
distinct-mset-set-def)
have *H*: ‹*s* ∈ {*M*. *if satisfiable* (*set-mset CS*) *then M* ≠ *None* ∧ *set* (*the M*) ⊨*sm CS else M* = *None*}›
if
[*simp*]: ‹*CS′* = *CS*› **and**
s: ‹*s* ∈ (λ*T*. *if pget-conflict T* = *None then Some* (*map lit-of* (*pget-trail T*)) *else None*) ‘
Collect (*conclusive-CDCL-state CS′* (*pinit-state* (*remdups-mset* ‘# *CS′*)))›
for *s* :: ‹*nat literal list option*› **and** *CS CS′*
proof −
obtain *T* **where**
s: ‹(*s* = *Some* (*map lit-of* (*pget-trail T*)) ∧ *pget-conflict T* = *None*) ∨
(*s* = *None* ∧ *pget-conflict T* ≠ *None*)› **and**
conc: ‹*conclusive-CDCL-state CS′* ([], *remdups-mset* ‘# *CS′*, {#}, *None*, {#}, {#}, {#}, {#}, {#}) *T*›
using *s* **by** *auto force*
show *?thesis*
using *s conc*
by (*auto simp*: *conclusive-CDCL-state-def lits-of-def true-annots-true-cls*)
qed

have *H*: ‹∃ *s′*∈{(*b*, *M*).
¬ *b* ⟶
(*if satisfiable* (*set-mset CS*) *then M* ≠ *None* ∧ *set* (*the M*) ⊨*sm CS*
else M = *None*)}.
(*s*, *s′*) ∈ {((*b*, *S*), *b′*, *T*). *b* = *b′* ∧ (¬*b* ⟶ *S* = *T*)}›
if
‹*CS′* = *CS*› **and**
‹*xa* ∈ {*T*. *T* = *pinit-state* (*remdups-mset* ‘# *CS′*)}› **and**
‹*xb* ∈ {-. *True*}› **and**
‹*s* ∈ *uncurry*
(λ*b T*. (*b*, *if pget-conflict T* = *None*
then Some (*map lit-of* (*pget-trail T*)) *else None*)) ‘
(*if* ¬ *xb then* {(*True*, *xa*)}
else {(*b*, *U*). ¬ *b* ⟶ *conclusive-CDCL-state CS′ xa U*})›
for *s* :: ‹*bool* × *nat literal list option*› **and**
CS CS′ :: ‹*nat literal multiset multiset*› **and** *xa* **and** *xb* :: *bool*
proof −
obtain *b T* **where**
s: ‹*s* = (*b*, *T*)› **by** (*cases s*)
have
b: ‹¬*b* ⟶ *T* ∈ (λ*T*. *if pget-conflict T* = *None then Some* (*map lit-of* (*pget-trail T*)) *else None*) ‘
Collect (*conclusive-CDCL-state CS* (*pinit-state* (*remdups-mset* ‘# *CS*)))›
using *that*(*2*−*4*)
by (*force simp add*: *image-iff s that split*: *if-splits*)
show *?thesis*
proof (*cases b*)
case *False*
then have *T*: ‹*T* ∈ (λ*T*. *if pget-conflict T* = *None then Some* (*map lit-of* (*pget-trail T*)) *else None*) ‘
Collect (*conclusive-CDCL-state CS′* (*pinit-state* (*remdups-mset* ‘# *CS′*)))›

```
      using b unfolding that(1) by fast
    show ?thesis
      using H[OF that(1) T]
      by (rule-tac x = ‹s› in bexI)
        (auto simp add: s False that)
  qed (auto simp: s)
 qed
 have if-RES: ‹(if xb then RETURN x
       else RES P) = (RES (if xb then {x} else P))› for x xb P
  by (auto simp: RETURN-def)
 show ?thesis
  unfolding SAT-bounded'-def model-if-satisfiable-bounded-def SAT-bounded-def Let-def
   nres-monad3
  apply (intro frefI nres-relI)
  apply refine-vcg
  subgoal for CS' CS
   unfolding RES-RETURN-RES RES-RES-RETURN-RES2 if-RES
   apply (rule RES-refine)
   unfolding pair-in-Id-conv bex-triv-one-point1 bex-triv-one-point2
   using H by presburger
  done
qed

lemma SAT-bounded-model-if-satisfiable':
 ‹(uncurry (λ-. SAT-bounded'), uncurry (λ-. model-if-satisfiable-bounded)) ∈
  [λ(-, CS). True]_f Id ×_r Id→ ⟨{((b, S), (b', T)). b = b' ∧ (¬b ⟶ S = T)}⟩nres-rel›
 using SAT-bounded-model-if-satisfiable unfolding fref-def
 by auto

definition SAT-l-bounded' where
 ‹SAT-l-bounded' CS = do{
  (b, S) ← SAT-l-bounded CS;
  RETURN (b, if ¬b ∧ get-conflict-l S = None then Some (map lit-of (get-trail-l S)) else None)
 }›


definition SAT0-bounded' where
 ‹SAT0-bounded' CS = do{
  (b, S) ← SAT0-bounded CS;
  RETURN (b, if ¬b ∧ get-conflict S = None then Some (map lit-of (get-trail S)) else None)
 }›

lemma SAT-l-bounded'-SAT0-bounded':
 shows ‹SAT-l-bounded' CS ≤ ⇓ {((b, S), (b', T)). b = b' ∧ (¬b ⟶ S = T)} (SAT0-bounded' CS)›
 unfolding SAT-l-bounded'-def SAT0-bounded'-def
 apply refine-vcg
 apply (rule SAT-l-bounded-SAT0-bounded)
 subgoal by (auto simp: extract-model-of-state-def)
 done

lemma SAT0-bounded'-SAT-bounded':
 shows ‹SAT0-bounded' CS ≤ ⇓ {((b, S), (b', T)). b = b' ∧ (¬b ⟶ S = T)} (SAT-bounded' (mset
‘# mset CS))›
 unfolding SAT-bounded'-def SAT0-bounded'-def
 apply refine-vcg
 apply (rule SAT0-bounded-SAT-bounded)
```

```
  subgoal by (auto simp: extract-model-of-state-def twl-st-l twl-st)
  done


definition IsaSAT-bounded :: ‹nat clause-l list ⇒ (bool × nat literal list option) nres› where
  ‹IsaSAT-bounded CS = do{
    (b, S) ← SAT-wl-bounded CS;
    RETURN (b, if ¬b ∧ get-conflict-wl S = None then extract-model-of-state S else extract-stats S)
  }›

lemma IsaSAT-bounded-alt-def:
  ‹IsaSAT-bounded CS = do{
    ASSERT(isasat-input-bounded (mset-set (extract-atms-clss CS {})));
    let 𝒜_in' = extract-atms-clss CS {};
    S ← RETURN init-state-wl;
    T ← init-dt-wl' CS (to-init-state S);
    failed ← SPEC (λ- :: bool. True);
    if ¬failed then do {
        RETURN (True, extract-stats init-state-wl)
    } else do {
      let T = from-init-state T;
      if get-conflict-wl T ≠ None
      then RETURN (False, extract-stats T)
      else if CS = [] then RETURN (False, Some [])
      else do {
        ASSERT (extract-atms-clss CS {} ≠ {});
        ASSERT(isasat-input-bounded-nempty (mset-set 𝒜_in'));
        ASSERT(mset ‘# ran-mf (get-clauses-wl T) + get-unit-clauses-wl T +
          get-subsumed-clauses-wl T + get-clauses0-wl T = remdups-mset ‘# mset ‘# mset CS);
        ASSERT(learned-clss-l (get-clauses-wl T) = {#});
        T ← rewatch-st T;
        T ← RETURN (finalise-init T);
        (b, S) ← cdcl-twl-stgy-restart-prog-bounded-wl T;
        RETURN (b, if ¬b ∧ get-conflict-wl S = None then extract-model-of-state S else extract-stats S)
      }
    }
  }› (is ‹?A = ?B›) for CS opts
proof −
  have H: ‹A = B ⟹ A ≤ ⇓ Id B› for A B
    by auto
  have 1: ‹?A ≤ ⇓ Id ?B›
    unfolding IsaSAT-bounded-def SAT-wl-bounded-def nres-bind-let-law If-bind-distrib nres-monad-laws
      Let-def finalise-init-def
    apply (refine-vcg)
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by (auto simp: extract-model-of-state-def)
    subgoal by (auto simp: extract-model-of-state-def)
    subgoal by auto
    subgoal by auto
    apply (rule H; solves auto)
    apply (rule H; solves auto)
    subgoal by (auto simp: extract-model-of-state-def)
    done
```

```
  have 2: ‹?B ≤ ⇓ Id ?A›
   unfolding IsaSAT-bounded-def SAT-wl-bounded-def nres-bind-let-law If-bind-distrib nres-monad-laws
     Let-def finalise-init-def
    apply (refine-vcg)
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by auto
    subgoal by (auto simp: extract-model-of-state-def)
    subgoal by auto
    subgoal by auto
    apply (rule H; solves auto)
    apply (rule H; solves auto)
    subgoal by auto
    done

  show ?thesis
    using 1 2 by simp
qed



definition empty-conflict-code' :: ‹(bool × - list × isasat-stats) nres› where
  ‹empty-conflict-code' = do{
     let M0 = [];
     RETURN (False, M0, empty-stats)}›


lemma IsaSAT-bounded-heur-alt-def:
  ‹IsaSAT-bounded-heur opts CS = do{
    ASSERT(isasat-input-bounded (mset-set (extract-atms-clss CS {})));
    ASSERT(∀ C∈set CS. ∀ L∈set C. nat-of-lit L ≤ unat32-max);
    let 𝒜in' = mset-set (extract-atms-clss CS {});
    ASSERT(isasat-input-bounded 𝒜in');
    ASSERT(distinct-mset 𝒜in');
    S ← init-state-wl-heur 𝒜in';
    (T::twl-st-wl-heur-init) ← init-dt-wl-heur-b CS S;
    failed ← RETURN ((isasat-fast-init T ∧ ¬is-failed-heur-init T));
    if ¬failed
    then  do {
      RETURN (True, empty-init-code)
    } else do {
      let T = convert-state 𝒜in' T;
      if ¬get-conflict-wl-is-None-heur-init T
      then RETURN (False, empty-init-code)
      else if CS = [] then do {stat ← empty-conflict-code; RETURN (False, stat)}
      else do {
        ASSERT(𝒜in' ≠ {#});
        ASSERT(isasat-input-bounded-nempty 𝒜in');
        ASSERT(rewatch-heur-st-fast-pre T);
        T ← rewatch-heur-st-init T;
        ASSERT(isasat-fast-init T);
        T ← finalise-init-code opts (T::twl-st-wl-heur-init);
        ASSERT(isasat-fast T);
```

```
      (b, U) ← cdcl-twl-stgy-restart-prog-bounded-wl-heur T;
      RETURN (b, if ¬b ∧ get-conflict-wl-is-None-heur U then extract-model-of-state-stat U
        else extract-state-stat U)
    }
  }
 }›
  unfolding Let-def IsaSAT-bounded-heur-def init-state-wl-heur-fast-def
   bind-to-let-conv isasat-information-banner-def virtual-copy-def
   id-apply IsaSAT-Profile.start-def IsaSAT-Profile.stop-def nres-monad1
  unfolding
   convert-state-def de-Morgan-disj not-not if-not-swap
  by (intro bind-cong[OF refl] if-cong[OF refl] refl)

lemma IsaSAT-heur-bounded-IsaSAT-bounded:
 ‹IsaSAT-bounded-heur b CS ≤ ⇓(bool-rel ×f model-stat-rel) (IsaSAT-bounded CS)›
proof −
  have init-dt-wl-heur: ‹init-dt-wl-heur-unb CS S ≤
      ⇓(twl-st-heur-parsing-no-WL A True O {(S, T). S = remove-watched T ∧
          get-watched-wl (fst T) = (λ-. [])})
       (init-dt-wl' CS T)›
    if
     ‹case (CS, T) of
      (CS, S) ⇒
  (∀ C∈set CS. literals-are-in-ℒin A (mset C))› and
     ‹((CS, S), CS, T) ∈ ⟨Id⟩list-rel ×f twl-st-heur-parsing-no-WL A True›
  for A CS T S
  proof −
   have H: ‹init-dt-wl' CS T = do {T ← init-dt-wl' CS T; RETURN T}›
    by auto
   have I: ‹⇓ {((S, C), T). C = [] ∧ (S, T) ∈ twl-st-heur-parsing-no-WL A True} (init-dt-wl CS T)
    ≤ ⇓{((S, C), T). (S,T) ∈ twl-st-heur-parsing-no-WL A True O {(S, T). S = remove-watched T ∧
     get-watched-wl (fst T) = (λ-. [])}} (init-dt-wl' CS T)›
    by (cases ‹init-dt-wl CS T›)
     (force simp: init-dt-wl'-def RES-RETURN-RES conc-fun-RES
      Image-iff)+
   show ?thesis
    unfolding init-dt-wl-heur-unb-def
    apply (subst H)
    apply (refine-rcg)
    apply (rule init-dt-wl-heur-init-dt-wl[THEN fref-to-Down-curry, of A CS T CS ‹(S,[])›,
      THEN order-trans])
    apply (rule that(1))
    apply (use that(2) in auto)[]
    apply (rule I)
    apply auto
    done
  qed
  have init-dt-wl-heur-b: ‹init-dt-wl-heur-b CS S ≤
      ⇓(twl-st-heur-parsing-no-WL A False O {(S, T). S = remove-watched T ∧
          get-watched-wl (fst T) = (λ-. [])})
       (init-dt-wl' CS T)›
    if
     ‹case (CS, T) of
      (CS, S) ⇒
  (∀ C∈set CS. literals-are-in-ℒin A (mset C))› and
     ‹((CS, S), CS, T) ∈ ⟨Id⟩list-rel ×f twl-st-heur-parsing-no-WL A True›
```

```
  for 𝒜 CS T S
proof −
  have H: ‹init-dt-wl′ CS T = do {T ← init-dt-wl′ CS T; RETURN T}›
    by auto
  have I: ‹⇓ {((S, C), T). C = [] ∧ (S, T) ∈ twl-st-heur-parsing-no-WL 𝒜 False} (init-dt-wl CS T)
    ≤ ⇓{((S, C), T). (S,T) ∈ twl-st-heur-parsing-no-WL 𝒜 False O {(S, T). S = remove-watched T ∧
    get-watched-wl (fst T) = (λ-. [])}} (init-dt-wl′ CS T)›
    by (cases ‹init-dt-wl CS T›)
      (force simp: init-dt-wl′-def RES-RETURN-RES conc-fun-RES
      Image-iff)+
  show ?thesis
    unfolding init-dt-wl-heur-b-def
    apply (subst H)
    apply (refine-rcg)
      apply (rule init-dt-wl-heur-init-dt-wl[THEN fref-to-Down-curry, of 𝒜 CS T CS ‹(S, [])›,
        THEN order-trans])
      apply (rule that(1))
      using that(2) apply (force simp: twl-st-heur-parsing-no-WL-def)
      apply (rule I)
      apply auto
      done
  qed
  have virtual-copy: ‹(virtual-copy 𝒜, ()) ∈ {(ℬ, c). c = () ∧ ℬ = 𝒜}› for ℬ 𝒜
    by (auto simp: virtual-copy-def)
  have input-le: ‹∀ C∈set CS. ∀ L∈set C. nat-of-lit L ≤ unat32-max›
    if ‹isasat-input-bounded (mset-set (extract-atms-clss CS {}))›
  proof (intro ballI)
    fix C L
    assume ‹C ∈ set CS› and ‹L ∈ set C›
    then have ‹L ∈# 𝓛_all (mset-set (extract-atms-clss CS {}))›
      by (auto simp: extract-atms-clss-alt-def in-𝓛_all-atm-of-𝒜_in)
    then show ‹nat-of-lit L ≤ unat32-max›
      using that by auto
  qed
  have lits-C: ‹literals-are-in-𝓛_in (mset-set (extract-atms-clss CS {})) (mset C)›
    if ‹C ∈ set CS› for C CS
    using that
    by (force simp: literals-are-in-𝓛_in-def in-𝓛_all-atm-of-𝒜_in
     in-all-lits-of-m-ain-atms-of-iff extract-atms-clss-alt-def
     atm-of-eq-atm-of)
  have init-state-wl-heur: ‹isasat-input-bounded 𝒜 ⟹
     init-state-wl-heur 𝒜 ≤ SPEC (λc. (c, init-state-wl) ∈
       {(S, S′). (S, S′) ∈ twl-st-heur-parsing-no-WL-wl 𝒜 True ∧
        inres (init-state-wl-heur 𝒜) S})› for 𝒜
    by (rule init-state-wl-heur-init-state-wl[THEN fref-to-Down-unRET-uncurry0-SPEC,
      of 𝒜, THEN strengthen-SPEC, THEN order-trans])
      auto

  have get-conflict-wl-is-None-heur-init: ‹ (Tb, Tc)
    ∈ ({(S,T). (S, T) ∈ twl-st-heur-parsing (mset-set (extract-atms-clss CS {})) True ∧
         get-clauses-wl-heur-init S = get-clauses-wl-heur-init U ∧
  get-conflict-wl-heur-init S = get-conflict-wl-heur-init U ∧
         get-clauses-wl (fst T) = get-clauses-wl (fst V) ∧
  get-conflict-wl (fst T) = get-conflict-wl (fst V) ∧
  get-unit-clauses-wl (fst T) = get-unit-clauses-wl (fst V)} O {(S, T). S = (T, {#})}) ⟹
    (¬ get-conflict-wl-is-None-heur-init Tb) = (get-conflict-wl Tc ≠ None)› for Tb Tc U V
```

by (*cases V*; *cases ‹get-conflict-wl-heur-init U›*) (*auto simp*: *twl-st-heur-parsing-def Collect-eq-comp′*
get-conflict-wl-is-None-heur-init-def
option-lookup-clause-rel-def)
have *get-conflict-wl-is-None-heur-init3*: ‹(*T*, *Ta*)
∈ *twl-st-heur-parsing-no-WL* (*mset-set* (*extract-atms-clss CS* {})) *False O*
{(*S*, *T*). *S* = *remove-watched T* ∧ *get-watched-wl* (*fst T*) = (λ-. [])} ⟹
(¬ *get-conflict-wl-is-None-heur-init T*) = (*get-conflict-wl* (*fst Ta*) ≠ *None*)› **for** *T Ta failed faileda*
by (*cases T*; *cases Ta*) (*auto simp*: *twl-st-heur-parsing-no-WL-def*
get-conflict-wl-is-None-heur-init-def
option-lookup-clause-rel-def)
have *banner*: ‹*isasat-information-banner*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *Tb*)
≤ *SPEC* (λ*c*. (*c*, ()) ∈ {(-, -). *True*})› **for** *Tb*
by (*auto simp*: *isasat-information-banner-def*)

let *?TT* = ‹*rewatch-heur-st-rewatch-st-rel CS*›
have *finalise-init-code*: ‹*finalise-init-code b*
(*convert-state* (*virtual-copy* (*mset-set* (*extract-atms-clss CS* {}))) *Tb*)
≤ *SPEC* (λ*c*. (*c*, *finalise-init Tc*) ∈ *twl-st-heur*)› (**is** *?A*) **and**
finalise-init-code3: ‹*finalise-init-code b Tb*
≤ *SPEC* (λ*c*. (*c*, *finalise-init Tc*) ∈ *twl-st-heur*)› (**is** *?B*)
if
T: ‹(*Tb*, *Tc*) ∈ *?TT U V*› **and**
confl: ‹¬ *get-conflict-wl Tc* ≠ *None*› **and**
nempty: ‹*extract-atms-clss CS* {} ≠ {}› **and**
clss-CS: ‹*mset* ‘# *ran-mf* (*get-clauses-wl Tc*) + *get-unit-clauses-wl Tc* +
get-subsumed-clauses-wl Tc + *get-clauses0-wl Tc* = *mset* ‘# *mset CS*› **and**
learned: ‹*learned-clss-l* (*get-clauses-wl Tc*) = {#}›
for *ba S T Ta Tb Tc u v U V*
proof −
have *1*: ‹*get-conflict-wl Tc* = *None*›
using *confl* **by** *auto*
have *2*: ‹*all-atms-st Tc* ≠ {#}›
using *nempty clss-CS* **unfolding** *all-atms-def all-lits-alt-def all-atms-st-def*
by (*simp add*: *ac-simps extract-atms-clss-alt-def*
all-lits-of-mm-empty-iff)
have *3*: ‹*set-mset* (*all-atms-st Tc*) = *set-mset* (*mset-set* (*extract-atms-clss CS* {}))›
using *nempty clss-CS* **unfolding** *all-atms-def all-atms-st-def all-lits-def*
apply (*auto simp*: *extract-atms-clss-alt-def ac-simps*
all-lits-of-mm-empty-iff in-all-lits-of-mm-ain-atms-of-iff atms-of-ms-def)
by (*metis* (*no-types*, *lifting*) *UN-iff atm-of-all-lits-of-mm*(*2*) *atm-of-lit-in-atms-of*
atms-of-mmltiset atms-of-ms-mset-unfold in-set-mset-eq-in set-image-mset)
have *H*: ‹*A* = *B* ⟹ *x* ∈ *A* ⟹ *x* ∈ *B*› **for** *A B x*
by *auto*
have *H′*: ‹*A* = *B* ⟹ *A x* ⟹ *B x*› **for** *A B x*
by *auto*

note *cong* = *trail-pol-cong heuristic-rel-cong*
option-lookup-clause-rel-cong
vdom-m-cong[*THEN H*] *isasat-input-nempty-cong*[*THEN iffD1*]
isasat-input-bounded-cong[*THEN iffD1*]
cach-refinement-empty-cong[*THEN H′*]
phase-saving-cong[*THEN H′*]
$\mathcal{L}_{all}$-*cong*[*THEN H*]
D_0-*cong*[*THEN H*] *lookup-clause-rel-cong*

```
  have 4: ‹(convert-state (mset-set (extract-atms-clss CS {})) Tb, Tc)
  ∈ twl-st-heur-post-parsing-wl True›
    using T nempty
    by (clarsimp simp: twl-st-heur-parsing-def twl-st-heur-post-parsing-wl-def
      convert-state-def eq-commute[of ‹mset -› ‹dom-m -›] all-atms-st-def
vdom-m-cong[OF 3[symmetric]] 𝓛all-cong[OF 3[symmetric]] bump-heur-init-cong[OF 3[symmetric]]
dest!: cong[OF 3[symmetric]])
     (simp-all add: add.assoc 𝓛all-all-atms-all-lits ac-simps
      flip: all-lits-def all-lits-alt-def2 all-lits-alt-def)
  show ?A
   by (rule finalise-init-finalise-init[THEN fref-to-Down-unRET-curry-SPEC, of b])
    (use 1 2 learned 4 in auto)
  then show ?B unfolding convert-state-def by auto
qed

have get-conflict-wl-is-None-heur-init2: ‹(U, V)
  ∈ twl-st-heur-parsing-no-WL (mset-set (extract-atms-clss CS {})) True O
    {(S, T). S = remove-watched T ∧ get-watched-wl (fst T) = (λ-. [])} ⟹
  (¬ get-conflict-wl-is-None-heur-init
      (convert-state (virtual-copy (mset-set (extract-atms-clss CS {}))) U)) =
  (get-conflict-wl (from-init-state V) ≠ None)› for U V
  by (auto simp: twl-st-heur-parsing-def Collect-eq-comp'
    get-conflict-wl-is-None-heur-init-def twl-st-heur-parsing-no-WL-def
    option-lookup-clause-rel-def convert-state-def from-init-state-def)

have rewatch-heur-st-fast-pre: ‹rewatch-heur-st-fast-pre
(convert-state (virtual-copy (mset-set (extract-atms-clss CS {}))) T)›
  if
    T: ‹(T, Ta)
     ∈ twl-st-heur-parsing-no-WL (mset-set (extract-atms-clss CS {})) True O
{(S, T). S = remove-watched T ∧ get-watched-wl (fst T) = (λ-. [])}› and
    length-le: ‹¬¬isasat-fast-init (convert-state (virtual-copy (mset-set (extract-atms-clss CS {}))) T)›
  for uu ba S T Ta baa uua uub
proof −
  have ‹valid-arena (get-clauses-wl-heur-init T) (get-clauses-wl (fst Ta))
    (set (get-vdom-heur-init T))›
    using T by (auto simp: twl-st-heur-parsing-no-WL-def)
  then show ?thesis
    using length-le unfolding rewatch-heur-st-fast-pre-def convert-state-def
      isasat-fast-init-def unat64-max-def unat32-max-def
    by (auto dest: valid-arena-in-vdom-le-arena)
qed
have rewatch-heur-st-fast-pre2: ‹rewatch-heur-st-fast-pre
(convert-state (mset-set (extract-atms-clss CS {})) T)›
  if
    T: ‹(T, Ta)
     ∈ twl-st-heur-parsing-no-WL (mset-set (extract-atms-clss CS {})) False O
{(S, T). S = remove-watched T ∧ get-watched-wl (fst T) = (λ-. [])}› and
    length-le: ‹¬¬isasat-fast-init (convert-state (virtual-copy (mset-set (extract-atms-clss CS {}))) T)›
and
    failed: ‹¬is-failed-heur-init T›
  for uu ba S T Ta baa uua uub
proof −
  have
    Ta: ‹(T, Ta)
   ∈ twl-st-heur-parsing-no-WL (mset-set (extract-atms-clss CS {})) True O
```

```
    {(S, T). S = remove-watched T ∧ get-watched-wl (fst T) = (λ-. [])}›
    using failed T by (cases T; cases Ta) (fastforce simp: twl-st-heur-parsing-no-WL-def)
  from rewatch-heur-st-fast-pre[OF this length-le]
  show ?thesis by simp
 qed
 have finalise-init-code2: ‹finalise-init-code b Tb
≤ SPEC (λc. (c, finalise-init Tc) ∈  {(S', T').
          (S', T') ∈ twl-st-heur ∧
          get-clauses-wl-heur-init Tb = get-clauses-wl-heur S' ∧
         get-learned-count-init Tb = get-learned-count S'})›
   (is ‹- ≤ SPEC (λc. - ∈ ?P)›)
 if
  Ta: ‹(T, Ta)
  ∈ twl-st-heur-parsing-no-WL (mset-set (extract-atms-clss CS {})) False O
    {(S, T). S = remove-watched T ∧ get-watched-wl (fst T) = (λ-. [])}› and
  confl: ‹¬ get-conflict-wl (from-init-state Ta) ≠ None› and
  ‹CS ≠ []› and
  nempty: ‹extract-atms-clss CS {} ≠ {}› and
  ‹isasat-input-bounded-nempty (mset-set (extract-atms-clss CS {}))› and
  clss-CS: ‹mset `# ran-mf (get-clauses-wl (from-init-state Ta)) +
   get-unit-clauses-wl (from-init-state Ta) + get-subsumed-clauses-wl (from-init-state Ta)
     + get-clauses0-wl (from-init-state Ta) =
   remdups-mset `# mset `# mset CS› and
  learned: ‹learned-clss-l (get-clauses-wl (from-init-state Ta)) = {#}› and
  ‹virtual-copy (mset-set (extract-atms-clss CS {})) ≠ {#}› and
  ‹isasat-input-bounded-nempty
    (virtual-copy (mset-set (extract-atms-clss CS {})))› and
  T: ‹(Tb, Tc) ∈ ?TT T Ta› and
  failed: ‹¬is-failed-heur-init T›
  for uu ba S T Ta baa uua uub V W b Tb Tc
 proof −
  have
  Ta: ‹(T, Ta)
  ∈ twl-st-heur-parsing-no-WL (mset-set (extract-atms-clss CS {})) True O
    {(S, T). S = remove-watched T ∧ get-watched-wl (fst T) = (λ-. [])}›
    using failed Ta by (cases T; cases Ta) (fastforce simp: twl-st-heur-parsing-no-WL-def)

  have 1: ‹get-conflict-wl Tc = None›
    using confl T by (auto simp: from-init-state-def)
  have Ta-Tc: ‹all-atms-st Tc = all-atms-st (from-init-state Ta)›
    using T Ta
    unfolding all-lits-alt-def  mem-Collect-eq prod.case relcomp.simps
      all-atms-def add.assoc apply −
    apply normalize-goal+
    by (auto simp flip: all-atms-def[symmetric] simp: all-lits-def
      twl-st-heur-parsing-no-WL-def twl-st-heur-parsing-def all-atms-st-def
      from-init-state-def)
  moreover have 3: ‹set-mset (all-atms-st (from-init-state Ta)) = set-mset (mset-set (extract-atms-clss
CS {}))›
    using clss-CS unfolding all-lits-alt-def  mem-Collect-eq prod.case relcomp.simps
      all-atms-def all-atms-st-def apply −
      by (auto simp: extract-atms-clss-alt-def ac-simps
        atm-of-all-lits-of-mm atms-of-ms-def)
  ultimately have 2: ‹all-atms-st Tc ≠ {#}›
    using nempty
    by auto
```

```
  have H: ‹A = B ⟹ x ∈ A ⟹ x ∈ B› for A B x
    by auto
  have H': ‹A = B ⟹ A x ⟹ B x› for A B x
    by auto

  note cong = trail-pol-cong heuristic-rel-cong
    option-lookup-clause-rel-cong
    vdom-m-cong[THEN H] isasat-input-nempty-cong[THEN iffD1]
    isasat-input-bounded-cong[THEN iffD1]
    cach-refinement-empty-cong[THEN H']
    phase-saving-cong[THEN H']
    ℒ_all-cong[THEN H]
    D₀-cong[THEN H] lookup-clause-rel-cong

  have 4: ‹(convert-state (mset-set (extract-atms-clss CS {})) Tb, Tc)
  ∈ twl-st-heur-post-parsing-wl True›
    using T nempty
    by (clarsimp simp: twl-st-heur-parsing-def twl-st-heur-post-parsing-wl-def all-atms-st-def
      convert-state-def eq-commute[of ‹mset -› ‹dom-m -›] from-init-state-def
vdom-m-cong[OF 3[symmetric]] ℒ_all-cong[OF 3[symmetric]] bump-heur-init-cong[OF 3[symmetric]]
dest!: cong[OF 3[symmetric]])
      (simp-all add: add.assoc ℒ_all-all-atms-all-lits ac-simps
       flip: all-lits-def all-lits-alt-def2 all-lits-alt-def)

  show ?thesis
    apply (rule finalise-init-finalise-init-full[unfolded conc-fun-RETURN,
      THEN order-trans])
    by (use 1 2 learned 4 T in ‹auto simp: from-init-state-def convert-state-def›)
 qed
 have isasat-fast: ‹isasat-fast Td›
  if
    fast: ‹¬ ¬ isasat-fast-init
   (convert-state (virtual-copy (mset-set (extract-atms-clss CS {})))
     T)› and
    Tb: ‹(Tb, Tc) ∈ ?TT T Ta› and
    Td: ‹(Td, Te) ∈ ?P Tb›
   for uu ba S T Ta baa uua uub Tb Tc Td Te
 proof −
   have ‹get-learned-count-init Tb = get-learned-count Td ⟹
     learned-clss-count-init Tb = learned-clss-count Td›
     by (cases Tb; cases Td; auto simp: learned-clss-count-init-def
       learned-clss-count-def)
    moreover have ‹get-learned-count Td = get-learned-count-init T ⟹
     learned-clss-count Td = learned-clss-count-init T›
     by (cases Td; cases T; auto simp: learned-clss-count-init-def
       learned-clss-count-def clss-size-lcountUS-def clss-size-lcountUE-def
       clss-size-lcount-def)
    ultimately show ?thesis
     using fast Td Tb unfolding mem-Collect-eq prod.case isasat-fast-init-def
     by (auto simp add: isasat-fast-def
       convert-state-def)
 qed
 define init-succesfull where ‹init-succesfull T = RETURN ((isasat-fast-init T ∧ ¬ is-failed-heur-init
T))› for T
 define init-succesfull2 where ‹init-succesfull2 = SPEC (λ- :: bool. True)›
```

have [*refine*]: ‹*init-succesfull T* ≤ ⇓ {(*b*, *b'*). (*b* = *b'*) ∧ (*b* ⟷ (*isasat-fast-init T* ∧ ¬ *is-failed-heur-init T*))}
init-succesfull2› **for** *T*
by (*auto simp*: *init-succesfull-def init-succesfull2-def intro*!: *RETURN-RES-refine*)
show *?thesis*
supply [[*goals-limit=1*]]
unfolding *IsaSAT-bounded-heur-alt-def IsaSAT-bounded-alt-def init-succesfull-def*[*symmetric*]
apply (*rewrite at* ‹*do* {- ← *init-dt-wl'* - -; - ← (¤ :: *bool nres*); *If* - - - }› *init-succesfull2-def*[*symmetric*])
apply (*refine-vcg virtual-copy init-state-wl-heur banner*)
subgoal by (*rule input-le*)
subgoal by (*rule distinct-mset-mset-set*)
apply (*rule init-dt-wl-heur-b*[*of* ‹*mset-set* (*extract-atms-clss CS* {})›])
subgoal by (*auto simp*: *lits-C*)
subgoal by (*clarsimp simp*: *twl-st-heur-parsing-no-WL-wl-def*
twl-st-heur-parsing-no-WL-def to-init-state-def
init-state-wl-def init-state-wl-heur-def
inres-def RES-RES-RETURN-RES
RES-RETURN-RES)
(*auto simp add*: *ac-simps*)
subgoal by *auto*
subgoal by (*simp add*: *empty-conflict-code-def model-stat-rel-def*
empty-init-code-def)
subgoal unfolding *from-init-state-def convert-state-def*
by (*rule get-conflict-wl-is-None-heur-init3*)
subgoal by (*simp add*: *empty-init-code-def model-stat-rel-def*)
subgoal by *simp*
subgoal by (*simp add*: *empty-conflict-code-def model-stat-rel-def*)
subgoal by (*simp add*: *mset-set-empty-iff extract-atms-clss-alt-def*)
subgoal for *uu ba S T Ta baa*
by (*rule rewatch-heur-st-fast-pre2*; *assumption?*)
(*clarsimp-all simp add*: *convert-state-def*)
apply (*rule rewatch-heur-st-rewatch-st3*[*unfolded virtual-copy-def id-apply*]; *assumption?*)
subgoal by *auto*
subgoal by (*clarsimp simp add*: *isasat-fast-init-def convert-state-def learned-clss-count-init-def*)
apply (*rule finalise-init-code2*; *assumption?*)
subgoal by *clarsimp*
subgoal by (*clarsimp simp add*: *isasat-fast-def isasat-fast-init-def convert-state-def*)
subgoal by (*clarsimp simp add*: *isasat-fast-def isasat-fast-init-def convert-state-def*)
subgoal by (*clarsimp simp add*: *isasat-fast-def isasat-fast-init-def convert-state-def ac-simps*
learned-clss-count-init-def learned-clss-count-def)
apply (*rule-tac r1* = ‹*length* (*get-clauses-wl-heur Td*)› **in**
cdcl-twl-stgy-restart-prog-bounded-wl-heur-cdcl-twl-stgy-restart-prog-bounded-wl-D[*THEN fref-to-Down*])
subgoal by (*simp add*: *isasat-fast-def snat64-max-def unat32-max-def*
unat64-max-def)
subgoal by *fast*
subgoal by *simp*
subgoal premises *p*
using *p*(*26*−)
by (*auto simp*: *twl-st-heur-loop-def get-conflict-wl-is-None-heur-def*
extract-stats-def extract-state-stat-def
option-lookup-clause-rel-def trail-pol-def
extract-model-of-state-def rev-map
extract-model-of-state-stat-def model-stat-rel-def
dest!: *ann-lits-split-reasons-map-lit-of*)
done
qed

lemma *ISASAT-bounded-SAT-l-bounded'*:
assumes
‹*isasat-input-bounded* (*mset-set* (⋃ *C*∈*set CS*. *atm-of* ‘ *set C*))›
shows ‹*IsaSAT-bounded CS* ≤ ⇓ {((*b*, *S*), (*b'*, *S'*)). *b* = *b'* ∧ (¬*b* ⟶ *S* = *S'*)} (*SAT-l-bounded' CS*)›
unfolding *IsaSAT-bounded-def SAT-l-bounded'-def*
apply *refine-vcg*
apply (*rule SAT-wl-bounded-SAT-l-bounded*)
subgoal using *assms* **by** *auto*
subgoal by (*auto simp*: *extract-model-of-state-def*)
done

lemma *IsaSAT-bounded-heur-model-if-sat*:
assumes
‹*isasat-input-bounded* (*mset-set* (⋃ *C*∈*set CS*. *atm-of* ‘ *set C*))›
shows ‹*IsaSAT-bounded-heur opts CS* ≤ ⇓ {((*b*, *m*), (*b'*, *m'*)). *b*=*b'* ∧ (¬*b* ⟶ (*m*,*m'*) ∈ *model-stat-rel*)}
(*model-if-satisfiable-bounded* (*mset* ‘# *mset CS*))›
apply (*rule IsaSAT-heur-bounded-IsaSAT-bounded*[*THEN order-trans*])
apply (*rule order-trans*)
apply (*rule ref-two-step'*)
apply (*rule ISASAT-bounded-SAT-l-bounded'*)
subgoal using *assms* **by** *auto*

unfolding *conc-fun-chain*
apply (*rule order-trans*)
apply (*rule ref-two-step'*)
apply (*rule SAT-l-bounded'-SAT0-bounded'*)

unfolding *conc-fun-chain*
apply (*rule order-trans*)
apply (*rule ref-two-step'*)
apply (*rule SAT0-bounded'-SAT-bounded'*)

unfolding *conc-fun-chain*
apply (*rule order-trans*)
apply (*rule ref-two-step'*)
apply (*rule SAT-bounded-model-if-satisfiable*[*THEN fref-to-Down*, *of* ‹*mset* ‘# *mset CS*›])
subgoal using *assms* **by** *auto*
apply (*rule IdI*)

unfolding *conc-fun-chain*
apply (*rule conc-fun-R-mono*)
apply (*clarsimp simp*: *model-stat-rel-def*)
done

lemma *IsaSAT-bounded-heur-model-if-sat'*:
‹(*uncurry IsaSAT-bounded-heur*, *uncurry* (λ-. *model-if-satisfiable-bounded*)) ∈
[λ(-, *CS*). (∀ *C*∈#*CS*. ∀ *L*∈#*C*. *nat-of-lit L* ≤ *unat32-max*)]$_f$
Id ×$_r$ *list-mset-rel O* ⟨*list-mset-rel*⟩*mset-rel* → ⟨{((*b*, *m*), (*b'*, *m'*)). *b*=*b'* ∧ (¬*b* ⟶ (*m*,*m'*) ∈ *model-stat-rel*)}⟩*nres-rel*›
proof −
have *H*: ‹*isasat-input-bounded* (*mset-set* (⋃ *C*∈*set CS*. *atm-of* ‘ *set C*))›
if *CS-p*: ‹∀ *C*∈#*CS'*. ∀ *L*∈#*C*. *nat-of-lit L* ≤ *unat32-max*› **and**
‹(*CS*, *CS'*) ∈ *list-mset-rel O* ⟨*list-mset-rel*⟩*mset-rel*›
for *CS CS'*

```
    unfolding isasat-input-bounded-def
  proof
    fix L
    assume L: ‹L ∈# 𝓛_all (mset-set (⋃C∈set CS. atm-of ‘ set C))›
    then obtain C where
      L: ‹C∈set CS ∧ (L ∈set C ∨ - L ∈ set C)›
      apply (cases L)
      apply (auto simp: extract-atms-clss-alt-def unat32-max-def
          𝓛_all-def)+
      apply (metis literal.exhaust-sel)+
      done
    have ‹nat-of-lit L ≤ unat32-max ∨ nat-of-lit (-L) ≤ unat32-max›
      using L CS-p that by (auto simp: list-mset-rel-def mset-rel-def br-def
      br-def mset-rel-def p2rel-def rel-mset-def
        rel2p-def[abs-def] list-all2-op-eq-map-right-iff')
    then show ‹nat-of-lit L ≤ unat32-max›
      using L
      by (cases L) (auto simp: extract-atms-clss-alt-def unat32-max-def)
  qed
  show ?thesis
    apply (intro frefI nres-relI)
    unfolding uncurry-def
    apply clarify
    subgoal for opt1 CS opt2 CS'
      apply (rule IsaSAT-bounded-heur-model-if-sat[THEN order-trans, of CS - opt1])
      subgoal by (rule H) auto
        apply (auto simp: list-mset-rel-def mset-rel-def br-def
          br-def mset-rel-def p2rel-def rel-mset-def
          rel2p-def[abs-def] list-all2-op-eq-map-right-iff')
        done
    done
qed

end
theory IsaSAT-Print-LLVM
  imports IsaSAT-Literals-LLVM
begin

definition print-propa :: ‹64 word ⇒ unit› where
  ‹print-propa - = ()›

definition print-confl :: ‹64 word ⇒ unit› where
  ‹print-confl - = ()›

definition print-dec :: ‹64 word ⇒ unit› where
  ‹print-dec - = ()›

definition print-res :: ‹64 word ⇒ 64 word ⇒ unit› where
  ‹print-res - - = ()›

definition print-lres :: ‹64 word ⇒ 64 word ⇒ unit› where
  ‹print-lres - - = ()›

definition print-uset :: ‹64 word ⇒ unit› where
  ‹print-uset - = ()›
```

definition *print-gcs* :: ‹*64 word* ⇒ *64 word* ⇒ *unit*› **where**
‹*print-gcs* - - = ()›

definition *print-binary-unit* :: ‹*64 word* ⇒ *unit*› **where**
‹*print-binary-unit* - = ()›

definition *print-binary-red-removed* :: ‹*64 word* ⇒ *unit*› **where**
‹*print-binary-red-removed* - = ()›

definition *print-purelit-elim* :: ‹*64 word* ⇒ *unit*› **where**
‹*print-purelit-elim* - = ()›

definition *print-purelit-rounds* :: ‹*64 word* ⇒ *64 word* ⇒ *unit*› **where**
‹*print-purelit-rounds* - - = ()›

definition *print-lbds* :: ‹*64 word* ⇒ *unit*› **where**
‹*print-lbds* - = ()›

definition *print-forward-rounds* :: ‹*64 word* ⇒ *64 word* ⇒ *unit*› **where**
‹*print-forward-rounds* - - = ()›

definition *print-forward-subsumed* :: ‹*64 word* ⇒ *64 word* ⇒ *unit*› **where**
‹*print-forward-subsumed* - - = ()›

definition *print-forward-tried* :: ‹*64 word* ⇒ *64 word* ⇒ *unit*› **where**
‹*print-forward-tried* - - = ()›

definition *print-forward-strengthened* :: ‹*64 word* ⇒ *64 word* ⇒ *unit*› **where**
‹*print-forward-strengthened* - - = ()›

definition *print-rephased* :: ‹*64 word* ⇒ *64 word* ⇒ *unit*› **where**
‹*print-rephased* - - = ()›

sepref-def *print-propa-impl*
is ‹*RETURN o print-propa*›
:: ‹*word-assn*^{k} →$_a$ *unit-assn*›
unfolding *print-propa-def*
by *sepref*

sepref-def *print-confl-impl*
is ‹*RETURN o print-confl*›
:: ‹*word-assn*^{k} →$_a$ *unit-assn*›
unfolding *print-confl-def*
by *sepref*

sepref-def *print-dec-impl*
is ‹*RETURN o print-dec*›
:: ‹*word-assn*^{k} →$_a$ *unit-assn*›
unfolding *print-dec-def*
by *sepref*

sepref-def *print-res-impl*
is ‹*uncurry* (*RETURN oo print-res*)›
:: ‹*word-assn*^{k} *$_a$ *word-assn*^{k} →$_a$ *unit-assn*›
unfolding *print-res-def*
by *sepref*

sepref-def *print-lres-impl*
  is ‹*uncurry* (*RETURN oo print-lres*)›
  :: ‹*word-assn*^{k} $*_a$ *word-assn*^{k} $\rightarrow_a$ *unit-assn*›
  unfolding *print-lres-def*
  by *sepref*

sepref-def *print-uset-impl*
  is ‹*RETURN o print-uset*›
  :: ‹*word-assn*^{k} $\rightarrow_a$ *unit-assn*›
  unfolding *print-uset-def*
  by *sepref*

definition *print-irred-clss* :: ‹*64 word* $\Rightarrow$ *unit*› **where**
  ‹*print-irred-clss* - = ()›

sepref-def *print-gc-impl*
  is ‹*uncurry* (*RETURN oo print-gcs*)›
  :: ‹*word-assn*^{k} $*_a$ *word-assn*^{k} $\rightarrow_a$ *unit-assn*›
  unfolding *print-gcs-def*
  by *sepref*

sepref-def *print-irred-clss-impl*
  is ‹*RETURN o print-irred-clss*›
  :: ‹*word-assn*^{k} $\rightarrow_a$ *unit-assn*›
  unfolding *print-irred-clss-def*
  by *sepref*

sepref-def *print-binary-unit-impl*
  is ‹*RETURN o print-binary-unit*›
  :: ‹*word-assn*^{k} $\rightarrow_a$ *unit-assn*›
  unfolding *print-binary-unit-def*
  by *sepref*

sepref-def *print-purelit-rounds-impl*
  is ‹*uncurry* (*RETURN oo print-purelit-rounds*)›
  :: ‹*word-assn*^{k} $*_a$ *word-assn*^{k} $\rightarrow_a$ *unit-assn*›
  unfolding *print-purelit-rounds-def*
  by *sepref*

sepref-def *print-purelit-elim-impl*
  is ‹*RETURN o print-purelit-elim*›
  :: ‹*word-assn*^{k} $\rightarrow_a$ *unit-assn*›
  unfolding *print-purelit-elim-def*
  by *sepref*

sepref-def *print-binary-red-removed-impl*
  is ‹*RETURN o print-binary-red-removed*›
  :: ‹*word-assn*^{k} $\rightarrow_a$ *unit-assn*›
  unfolding *print-binary-red-removed-def*
  by *sepref*

sepref-def *print-forward-rounds-impl*
  is ‹*uncurry* (*RETURN oo print-forward-rounds*)›
  :: ‹*word-assn*^{k} $*_a$ *word-assn*^{k} $\rightarrow_a$ *unit-assn*›
  unfolding *print-forward-rounds-def*

by *sepref*

sepref-def *print-forward-subsumed-impl*
is ‹*uncurry* (*RETURN oo print-forward-subsumed*)›
:: ‹*word-assn*^{k} $*_a$ *word-assn*^{k} $\rightarrow_a$ *unit-assn*›
unfolding *print-forward-subsumed-def*
by *sepref*

sepref-def *print-forward-tried-impl*
is ‹*uncurry* (*RETURN oo print-forward-tried*)›
:: ‹*word-assn*^{k} $*_a$ *word-assn*^{k} $\rightarrow_a$ *unit-assn*›
unfolding *print-forward-tried-def*
by *sepref*

sepref-def *print-forward-strengthened-impl*
is ‹*uncurry* (*RETURN oo print-forward-strengthened*)›
:: ‹*word-assn*^{k} $*_a$ *word-assn*^{k} $\rightarrow_a$ *unit-assn*›
unfolding *print-forward-strengthened-def*
by *sepref*

sepref-def *print-rephased-impl*
is ‹*uncurry* (*RETURN oo print-rephased*)›
:: ‹*word-assn*^{k} $*_a$ *word-assn*^{k} $\rightarrow_a$ *unit-assn*›
unfolding *print-rephased-def*
by *sepref*

end
theory *IsaSAT-LLVM*
imports
IsaSAT-Print-LLVM
IsaSAT-CDCL-LLVM
IsaSAT-Initialisation-LLVM
IsaSAT-Restart-Simp-LLVM
Version
IsaSAT-Defs
begin

hide-const (**open**)*array-assn*

hide-const (**open**)*IICF-Multiset.mset-rel*
hide-const (**open**) *NEMonad.ASSERT NEMonad.RETURN*

lemma *convert-state-hnr*:
‹(*uncurry* (*Mreturn oo* (λ- *S*. *S*)), *uncurry* (*RETURN oo convert-state*))
∈ *ghost-assn*^{k} $*_a$ (*isasat-init-assn*)^d $\rightarrow_a$
isasat-init-assn›
unfolding *convert-state-def*
by *sepref-to-hoare vcg*

declare *convert-state-hnr*[*sepref-fr-rules*]

Chapter 25

Code of Full IsaSAT

abbreviation *model-stat-assn* **where**
‹*model-stat-assn* ≡ *bool1-assn* ×$_a$ (*arl64-assn unat-lit-assn*) ×$_a$ *isasat-stats-assn*›

abbreviation *model-stat-assn*$_0$ ::
‹*bool* ×
nat literal list ×
isasat-stats
⇒ *1 word* ×
(*64 word* × *64 word* × *32 word ptr*) ×
-
⇒ *llvm-amemory* ⇒ *bool*›
where
‹*model-stat-assn*$_0$ ≡ *bool1-assn* ×$_a$ (*al-assn unat-lit-assn*) ×$_a$ *isasat-stats-assn*›

abbreviation *lits-with-max-assn* :: ‹*nat multiset*
⇒ (*64 word* × *64 word* × *32 word ptr*) × *32 word* ⇒ *llvm-amemory* ⇒ *bool*› **where**
‹*lits-with-max-assn* ≡ *hr-comp* (*arl64-assn atom-assn* ×$_a$ *uint32-nat-assn*) *lits-with-max-rel*›

abbreviation *lits-with-max-assn*$_0$ :: ‹*nat multiset*
⇒ (*64 word* × *64 word* × *32 word ptr*) × *32 word* ⇒ *llvm-amemory* ⇒ *bool*› **where**
‹*lits-with-max-assn*$_0$ ≡ *hr-comp* (*al-assn atom-assn* ×$_a$ *unat32-assn*) *lits-with-max-rel*›

lemma *lits-with-max-assn-alt-def*: ‹*lits-with-max-assn* = *hr-comp* (*arl64-assn atom-assn* ×$_a$ *uint32-nat-assn*)
(*lits-with-max-rel O* ⟨*nat-rel*⟩*IICF-Multiset.mset-rel*)›
proof −
have *1*: ‹(*lits-with-max-rel O* ⟨*nat-rel*⟩*IICF-Multiset.mset-rel*) = *lits-with-max-rel*›
by (*auto simp*: *mset-rel-def p2rel-def rel2p-def*[*abs-def*] *br-def*
rel-mset-def lits-with-max-rel-def list-rel-def list-all2-op-eq-map-right-iff' list.rel-eq)
show *?thesis*
unfolding *1*
by *auto*
qed

lemma *tuple15-rel-Id*: ‹(⟨*Id*, *Id*, *Id*, *nat-rel*, ⟨⟨*Id*⟩*list-rel*⟩*list-rel*, *Id*, ⟨*bool-rel*⟩*list-rel*,*nat-rel*,
Id, *Id*, *Id*, *Id*, *Id*, *Id*, *Id*⟩*tuple15-rel*) = *Id*›
apply (*standard*; *standard*)
apply (*case-tac x*)
apply (*auto simp*: *hr-comp-def*[*abs-def*] *tuple15-rel-def split*: *tuple15.splits*)
done

lemma *init-state-wl-D'-code-isasat*: ‹*hr-comp isasat-init-assn*

(⟨*Id*, *Id*, *Id*, *nat-rel*, ⟨⟨*Id*⟩*list-rel*⟩*list-rel*, *Id*, ⟨*bool-rel*⟩*list-rel*,*nat-rel*,
Id, *Id*, *Id*, *Id*, *Id*, *Id*, *Id*⟩*tuple15-rel*) = *isasat-init-assn*›
unfolding *tuple15-rel-Id*
by (*auto simp*: *split*: *tuple15.splits*)

definition *split-trail* **where** ‹*split-trail x* =*x*›

sepref-def *split-trail-impl*
is ‹*RETURN o split-trail*›
:: ‹*trail-pol-fast-assn*^{d} →$_a$ *arl64-assn unat-lit-assn* ×$_a$ *larray64-assn* (*tri-bool-assn*) ×$_a$
larray64-assn uint32-nat-assn ×$_a$
larray64-assn sint64-nat-assn ×$_a$ *uint32-nat-assn* ×$_a$
arl64-assn uint32-nat-assn ×$_a$ *sint64-nat-assn*›
unfolding *trail-pol-fast-assn-def split-trail-def*
by *sepref*

lemma *extract-model-of-state-stat-alt-def*:
‹*RETURN o extract-model-of-state-stat* = (λ*S*. *case S of Tuple17 MM′ N′ D′ j W′ vm clvls cach lbd*
outl stats
heur vdom lcount opts old-arena occs ⇒
do {- ← *print-trail2* (*MM′*);
(*M*,*M′*) ← *RETURN* (*split-trail MM′*);
mop-free M′; *mop-free N′*; *mop-free D′*; *mop-free j*; *mop-free W′*; *mop-free vm*;
mop-free clvls;
mop-free cach; *mop-free lbd*; *mop-free outl*; *mop-free heur*;
mop-free vdom; *mop-free opts*;
mop-free old-arena; *mop-free lcount*; *mop-free occs*;
RETURN (*False*, *M*, *stats*)
})›
by (*auto simp*: *extract-model-of-state-stat-def mop-free-def print-trail2-def split-trail-def*
intro!: *ext split*: *isasat-int-splits*)

schematic-goal *mk-free-lbd-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE aivdom-assn ?fr*›
unfolding *aivdom-assn-def code-hider-assn-def* **by** *synthesize-free+*

sepref-def *extract-model-of-state-stat*
is ‹*RETURN o extract-model-of-state-stat*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *model-stat-assn*›
supply [[*goals-limit*=*1*]]
unfolding *extract-model-of-state-stat-alt-def*
trail-pol-fast-assn-def
by *sepref*

lemma *extract-state-stat-alt-def*:
‹*RETURN o extract-state-stat* = (λ*S*. *case S of Tuple17 M N′ D′ j W′ vm clvls cach lbd outl stats*
heur vdom lcount opts old-arena occs ⇒
do {
mop-free M; *mop-free N′*; *mop-free D′*; *mop-free j*; *mop-free W′*; *mop-free vm*;
mop-free clvls;
mop-free cach; *mop-free lbd*; *mop-free outl*; *mop-free heur*;
mop-free vdom; *mop-free opts*;
mop-free old-arena; *mop-free lcount*; *mop-free occs*;
RETURN (*True*, [], *stats*)})›
by (*auto simp*: *extract-state-stat-def mop-free-def split*: *isasat-int-splits intro*!: *ext*)

sepref-def *extract-state-stat*

is ‹*RETURN o extract-state-stat*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *model-stat-assn*›
supply [[*goals-limit=1*]]
unfolding *extract-state-stat-alt-def isasat-bounded-assn-def*
al-fold-custom-empty[**where** *'l=64*]
by *sepref*

sepref-def *empty-conflict-code'*
is ‹*uncurry0* (*empty-conflict-code*)›
:: ‹*unit-assn*^{k} →$_a$ *model-stat-assn*›
unfolding *empty-conflict-code-def*
apply (*rewrite* **in** ‹*let - =* ⌑ *in -*› *al-fold-custom-empty*[**where** *'l=64*])
apply (*rewrite* **in** ‹*let - =* ⌑ *in -*› *annotate-assn*[**where** *A=‹arl64-assn unat-lit-assn›*])
by *sepref*

declare *empty-conflict-code'.refine*[*sepref-fr-rules*]

sepref-def *empty-init-code'*
is ‹*uncurry0* (*RETURN empty-init-code*)›
:: ‹*unit-assn*^{k} →$_a$ *model-stat-assn*›
unfolding *empty-init-code-def al-fold-custom-empty*[**where** *'l=64*]
apply (*rewrite* **in** ‹*RETURN* (-, ⌑,-)› *annotate-assn*[**where** *A=‹arl64-assn unat-lit-assn›*])
by *sepref*

sepref-register *init-dt-wl-heur-full*

sepref-register *to-init-state from-init-state get-conflict-wl-is-None-init*
init-dt-wl-heur

sepref-def *isasat-fast-bound-impl*
is ‹*uncurry0* (*RETURN isasat-fast-bound*)›
:: ‹*unit-assn*^{k} →$_a$ *sint64-nat-assn*›
unfolding *isasat-fast-bound-alt-def*
apply (*annot-snat-const* ‹*TYPE*(*64*)›)
by *sepref*

lemma *isasat-fast-init-alt-def*:
‹*RETURN o isasat-fast-init* = (λ*S*. *do*{
let n = *length* (*get-clauses-wl-heur-init S*);
let lcountUE = *clss-size-lcountUE* (*get-learned-count-init S*);
let lcount = *clss-size-lcount* (*get-learned-count-init S*);
let lcountUEk = *clss-size-lcountUEk* (*get-learned-count-init S*);
let lcountUS = *clss-size-lcountUS* (*get-learned-count-init S*);
let lcountU0 = *clss-size-lcountU0* (*get-learned-count-init S*);
ASSERT(*18446744073709551615* ∈ *unats LENGTH*(*64*));
c ← *RETURN 18446744073709551615*;
if ¬(*n* ≤ *isasat-fast-bound* ∧ *lcount* < *c* − *lcountUE*) *then RETURN False*
else do {
ASSERT(*lcount* + *lcountUE* ∈ *unats LENGTH*(*64*));
a ← *RETURN* (*lcount* + *lcountUE*);
if ¬*a* < *c* − *lcountUS then RETURN False*
else do {
ASSERT(*a* + *lcountUS* ∈ *unats LENGTH*(*64*));
a ← *RETURN* (*a* + *lcountUS*);
if ¬*a* < *c* − *lcountU0 then RETURN False*
else do {

```
        ASSERT(a + lcountU0 ∈ unats LENGTH(64));
        a ← RETURN (a + lcountU0);
        if ¬a < c − lcountUEk then RETURN False
        else do {
        ASSERT(a + lcountUEk ∈ unats LENGTH(64));
        a ← RETURN (a + lcountUEk);
          RETURN(a < c)
        }
      }

    }
  }})›
  by (auto simp: isasat-fast-init-def unat64-max-def unat32-max-def isasat-fast-bound-def max-uint-def
    clss-size-lcountUS-def clss-size-lcountUE-def clss-size-lcount-def learned-clss-count-init-def unats-def
     clss-size-lcountU0-def clss-size-lcountUEk-def Let-def bind-ASSERT-eq-if split: if-splits intro!: AS-
SERT-leI
  intro!: ext)

lemma isasat-fast-init-codeI: ‹(aa :: 64 word, b) ∈ unat-rel64 ⟹ b ≤ 18446744073709551615›
  using unat-lt-max-unat[of aa]
  by (auto simp: unat-rel-def unat.rel-def br-def max-unat-def)

sepref-def isasat-fast-init-code
  is ‹RETURN o isasat-fast-init›
  :: ‹isasat-init-assn^k →_a bool1-assn›
  supply [[goals-limit=1]]
  supply [sepref-bounds-simps del] = max-snat-def max-unat-def max-sint-def min-sint-def
  supply [intro] = isasat-fast-init-codeI
  unfolding isasat-fast-init-alt-def  if-not-swap isasat-fast-init-def
    clss-size-lcountUS-st-init-def[symmetric]
    clss-size-lcountUEk-st-init-def[symmetric]
    clss-size-lcountUE-st-init-def[symmetric] full-arena-length-st-init-def[symmetric]
    clss-size-lcount-st-init-def[symmetric]
    clss-size-lcountU0-st-init-def[symmetric]
  apply (rewrite at ‹RETURN ¤› unat-const-fold[where 'a=64])
  by sepref

sepref-register
   cdcl-twl-stgy-restart-prog-wl-heur

declare init-state-wl-D'-code.refine[FCOMP init-state-wl-D',
  unfolded lits-with-max-assn-alt-def[symmetric] init-state-wl-heur-fast-def[symmetric],
  unfolded init-state-wl-D'-code-isasat, sepref-fr-rules]

lemma [sepref-fr-rules]: ‹(init-state-wl-D'-code, init-state-wl-heur-fast)
∈ [λx. distinct-mset x ∧
      (∀ L∈#\mathcal{L}_{all} x. nat-of-lit L ≤ unat32-max)]_a lits-with-max-assn^k → isasat-init-assn›
  using init-state-wl-D'-code.refine[FCOMP init-state-wl-D']
  unfolding lits-with-max-assn-alt-def[symmetric] init-state-wl-D'-code-isasat
    init-state-wl-heur-fast-def
  by auto


definition ghost-assn where ‹ghost-assn = hr-comp unit-assn virtual-copy-rel›

lemma [sepref-fr-rules]: ‹(Mreturn o (λ-. ()), RETURN o virtual-copy) ∈ lits-with-max-assn^k →_a ghost-assn›
```

proof −
  have [*simp*]: ‹(λ*s*. (∃ *xa*. (↑(*xa* = *x*)) *s*)) = (↑*True*)› **for** *s* :: ‹'*b*::*sep-algebra*› **and** *x* :: '*a*
    by (*auto simp*: *pred-lift-extract-simps*)
  show *?thesis*
    unfolding *virtual-copy-def ghost-assn-def virtual-copy-rel-def*
    apply *sepref-to-hoare*
    apply *vcg'*
    apply (*auto simp*: *ENTAILS-def hr-comp-def snat-rel-def pure-true-conv*)
    apply (*rule Defer-Slot.remove-slot*)
    done
qed

sepref-register *virtual-copy empty-conflict-code empty-init-code*
  isasat-fast-init is-failed-heur-init
  extract-model-of-state-stat extract-state-stat
  isasat-information-banner
  finalise-init-code
  IsaSAT-Initialisation.rewatch-heur-st-fast
  get-conflict-wl-is-None-heur
  cdcl-twl-stgy-prog-bounded-wl-heur
  get-conflict-wl-is-None-heur-init
  convert-state

lemma *isasat-information-banner-alt-def*:
  ‹*isasat-information-banner S* =
    RETURN (())›
  by (*auto simp*: *isasat-information-banner-def*)

schematic-goal *mk-free-ghost-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE ghost-assn ?fr*›
  unfolding *ghost-assn-def*
  by *synthesize-free*

lemma *IsaSAT-bounded-heur-alt-def*:
  ‹*IsaSAT-bounded-heur opts CS* = *do*{
    - ← *RETURN* (*IsaSAT-Profile.start-initialisation*);
    ASSERT(*isasat-input-bounded* (*mset-set* (*extract-atms-clss CS* {})));
    ASSERT(∀ *C*∈*set CS*. ∀ *L*∈*set C*. *nat-of-lit L* ≤ *unat32-max*);
    let $\mathcal{A}_{in}'$ = *mset-set* (*extract-atms-clss CS* {});
    ASSERT(*isasat-input-bounded* $\mathcal{A}_{in}'$);
    ASSERT(*distinct-mset* $\mathcal{A}_{in}'$);
    let $\mathcal{A}_{in}''$ = *virtual-copy* $\mathcal{A}_{in}'$;
    let b = *opts-unbounded-mode opts*;
    S ← *init-state-wl-heur-fast* $\mathcal{A}_{in}'$;
    (*T*::*twl-st-wl-heur-init*) ← *init-dt-wl-heur-b CS S*;
    let T = *convert-state* $\mathcal{A}_{in}''$ *T*;
    - ← *RETURN* (*IsaSAT-Profile.stop-initialisation*);
    if isasat-fast-init T ∧ ¬*is-failed-heur-init T*
    then do {
      if ¬*get-conflict-wl-is-None-heur-init T*
      then RETURN (*False*, *empty-init-code*)
      else if CS = [] *then do* {*mop-free CS*; *stat* ← *empty-conflict-code*; *RETURN* (*False*, *stat*)}
      else do {
        - ← *RETURN* (*IsaSAT-Profile.start-initialisation*);
        mop-free CS;
        ASSERT($\mathcal{A}_{in}''$ ≠ {#});
        ASSERT(*isasat-input-bounded-nempty* $\mathcal{A}_{in}''$);

```
        - ← isasat-information-banner T;
        ASSERT(rewatch-heur-st-fast-pre T);
        T ← rewatch-heur-st-init T;
        ASSERT(isasat-fast-init T);
        T ← finalise-init-code opts (T::twl-st-wl-heur-init);
        - ← RETURN (IsaSAT-Profile.stop-initialisation);
        ASSERT(isasat-fast T);
        (b, U) ← cdcl-twl-stgy-restart-prog-bounded-wl-heur T;
       RETURN (b, if ¬b ∧ get-conflict-wl-is-None-heur U then IsaSAT-Defs.extract-model-of-state-stat
U
          else IsaSAT-Defs.extract-state-stat U)
      }
    }
    else do {mop-free CS; RETURN (True, empty-init-code)}
       }›
  unfolding mop-free-def
 by (auto simp: IsaSAT-bounded-heur-def cong: if-cong)

sepref-def IsaSAT-code
  is ‹uncurry IsaSAT-bounded-heur›
  :: ‹opts-assn^d *_a (clauses-ll-assn)^d →_a bool1-assn ×_a model-stat-assn›
  supply [[goals-limit=1]] isasat-fast-init-def[simp]
  unfolding IsaSAT-bounded-heur-alt-def empty-conflict-def[symmetric]
    get-conflict-wl-is-None
     init-dt-wl-heur-b-def[symmetric]

    init-dt-step-wl-heur-unb-def[symmetric] init-dt-wl-heur-unb-def[symmetric]
    length-0-conv[symmetric]  op-list-list-len-def[symmetric]
    isasat-information-banner-alt-def
  supply get-conflict-wl-is-None-heur-init-def[simp]
  supply  get-conflict-wl-is-None-def[simp]
   option.splits[split]

  apply (rewrite at ‹extract-atms-clss - ⌑› op-extract-list-empty-def[symmetric])
  apply (rewrite at ‹extract-atms-clss - ⌑› op-extract-list-empty-def[symmetric])
  apply (annot-snat-const ‹TYPE(64)›)
  by sepref


sepref-register print-forward-rounds print-forward-subsumed print-forward-strengthened

abbreviation (input) C-bool-to-bool :: ‹8 word ⇒ bool› where
  ‹C-bool-to-bool g ≡ g ≠ 0›

definition IsaSAT-bounded-heur-wrapper :: ‹8 word ⇒ 8 word ⇒ 8 word ⇒ 64 word ⇒ 64 word ⇒ nat
⇒
  8 word ⇒ 64 word ⇒ 64 word ⇒ 64 word ⇒ 8 word ⇒ - ⇒ (nat) nres›where
  ‹IsaSAT-bounded-heur-wrapper red res unbdd mini res1 res2 target-option fema sema units subsume C
= do {
      let opts = IsaOptions (C-bool-to-bool red) (C-bool-to-bool res)
        (C-bool-to-bool unbdd) mini res1 res2
        (if target-option = 2 then TARGET-ALWAYS else if target-option = 0 then TARGET-NEVER
else TARGET-STABLE-ONLY)
        fema sema units (C-bool-to-bool subsume);
      (b, (b', -, stats)) ← IsaSAT-bounded-heur (opts) C;
      let (- :: unit) = print-propa (stats-propagations stats);
```

```
    let (- :: unit) = print-confl (stats-conflicts stats);
    let (- :: unit) = print-dec (stats-decisions stats);
    let (- :: unit) = print-res (stats-restarts stats) (stats-conflicts stats);
    let (- :: unit) = print-lres (stats-reductions stats) (stats-conflicts stats);
    let (- :: unit) = print-uset (stats-fixed stats);
    let (- :: unit) = print-gcs (stats-gcs stats) (stats-conflicts stats);
    let (- :: unit) = print-irred-clss (stats-irred stats);
    let (- :: unit) = print-binary-unit (stats-binary-units stats);
    let (- :: unit) = print-binary-red-removed (stats-binary-removed stats);
    let (- :: unit) = print-purelit-elim (stats-pure-lits-removed stats);
    let (- :: unit) = print-purelit-rounds (stats-pure-lits-rounds stats) (stats-conflicts stats);
    let (- :: unit) = print-forward-rounds (stats-forward-rounds stats) (stats-conflicts stats);
    let (- :: unit) = print-forward-tried (stats-forward-tried stats) (stats-forward-rounds stats);
    let (- :: unit) = print-forward-subsumed (stats-forward-subsumed stats) (stats-forward-tried stats);
     let (- :: unit) = print-forward-strengthened (stats-forward-strengthened stats) (stats-forward-tried
stats);
    let (- :: unit) = print-rephased (stats-rephase stats) (stats-conflicts stats);
    RETURN ((if b then 2 else 0) + (if b' then 1 else 0))
  }›
```

The calling convention of LLVM and clang is not the same, so returning the model is currently unsupported. We return only the flags (as ints, not as bools) and the statistics.

```
sepref-register IsaSAT-bounded-heur default-opts

abbreviation bool-C-assn where
  ‹bool-C-assn ≡ (word-assn' (TYPE(8)))›

sepref-def IsaSAT-wrapped
  is ‹uncurry11 IsaSAT-bounded-heur-wrapper›
  :: ‹bool-C-assn^k *_a bool-C-assn^k *_a bool-C-assn^k *_a word64-assn^k *_a word64-assn^k *_a
     (snat-assn' (TYPE(64)))^k *_a bool-C-assn^k *_a word64-assn^k *_a word64-assn^k *_a
     word64-assn^k *_a bool-C-assn^k *_a (clauses-ll-assn)^d →_a sint64-nat-assn›
  supply [[goals-limit=1]] if-splits[split]
  unfolding IsaSAT-bounded-heur-wrapper-def
  apply (annot-snat-const ‹TYPE(64)›)
  by sepref
```

The setup to transmit the version is a bit complicated, because Isabelle does not support direct export of string literals. Therefore, we actually convert the version to an array chars (more precisely, of machine words – ended with 0) that can be read and printed by the C layer. Note the conversion must be automatic, because the version depends on the underlying git repository, hence the call to auto.

```
function array-of-version where
  ‹array-of-version i str arr =
    (if i ≥ length str then arr
    else array-of-version (i+1) str (arr[i := str ! i]))›
  by pat-completeness auto
termination
  apply (relation ‹measure (λ(i,str, arr). length str − i)›)
  apply auto
  done
```

Using this as version number makes our work on the cluster easier and makes the version checking slightly easier (because the git hash is never up-to-date).

```
definition internal-version :: ‹string› where ‹internal-version = ''0i''›
```

sepref-definition *llvm-version*
  is ‹*uncurry0* (*RETURN* (
      let str = map (*nat-of-integer o* (*of-char* :: - ⇒ *integer*)) (*internal-version* @ ''−'' @ *String.explode Version.version*) @ [*0*] *in*
       array-of-version 0 str (*replicate* (*length str*) *0*)))›
  :: ‹*unit-assn*^{k} →$_a$ *IICF-Array.array-assn sint32-nat-assn*›
  supply[[*goals-limit=1*]]
  unfolding *Version.version-def String.explode-code internal-version-def*
    String.asciis-of-Literal
  apply (*simp add*: *String.asciis-of-Literal of-char-of char-of-char nat-of-integer-def*
    del: *list-update.simps replicate.simps*)
  apply (*annot-snat-const* ‹*TYPE*(*32*)›)
  unfolding *array-fold-custom-replicate*
  unfolding *hf-pres.simps*[*symmetric*]
  by *sepref*

experiment
begin
  lemmas [*llvm-code*] = *llvm-version-def*

  lemmas [*llvm-inline*] =
    unit-propagation-inner-loop-body-wl-fast-heur-code-def
    FOCUSED-MODE-def DEFAULT-INIT-PHASE-def STABLE-MODE-def
    find-unwatched-wl-st-heur-fast-code-def
    update-clause-wl-fast-code-def

lemmas [*unfolded inline-direct-return-node-case*, *llvm-code*] = *units-since-last-GC-st-code-def*[*unfolded read-all-st-code-def*]
lemmas [*llvm-code del*] = *units-since-last-GC-st-code-def*

export-llvm
    llvm-version **is** ‹*STRING-VERSION llvm-version*()›
    IsaSAT-code
    IsaSAT-wrapped
    IsaSAT-Profile-PROPAGATE **is** ‹*PROFILE-CST IsaSAT-Profile-PROPAGATE*()›
    IsaSAT-Profile-REDUCE **is** ‹*PROFILE-CST IsaSAT-Profile-REDUCE*()›
    IsaSAT-Profile-GC **is** ‹*PROFILE-CST IsaSAT-Profile-GC*()›
    IsaSAT-Profile-ANALYZE **is** ‹*PROFILE-CST IsaSAT-Profile-ANALYZE*()›
    IsaSAT-Profile-MINIMIZATION **is** ‹*PROFILE-CST IsaSAT-Profile-MINIMIZATION*()›
    IsaSAT-Profile-INITIALISATION **is** ‹*PROFILE-CST IsaSAT-Profile-INITIALISATION*()›
    IsaSAT-Profile-SUBSUMPTION **is** ‹*PROFILE-CST IsaSAT-Profile-SUBSUMPTION*()›
    IsaSAT-Profile-PURE-LITERAL **is** ‹*PROFILE-CST IsaSAT-Profile-PURE-LITERAL*()›
    IsaSAT-Profile-BINARY-SIMP **is** ‹*PROFILE-CST IsaSAT-Profile-BINARY-SIMP*()›
  defines ‹
     typedef int8-t CBOOL;
     typedef int8-t PROFILE-CST;
     typedef struct {*int64-t size*; *struct* {*int64-t used*; *uint32-t* ∗*clause*;};} *CLAUSE*;
     typedef struct {*int64-t num-clauses*; *CLAUSE* ∗*clauses*;} *CLAUSES*;
     typedef int32-t∗ *STRING-VERSION*;
  ›
  file ‹*code/src/isasat-restart.ll*›

end

end

theory *IsaSAT-All-LLVM*
imports *IsaSAT-LLVM IsaSAT*
begin

definition *model-assn* **where**
‹*model-assn* = *hr-comp model-stat-assn model-stat-rel*›

definition *model-bounded-assn* **where**
‹*model-bounded-assn* =
hr-comp (*bool1-assn* $\times_a$ *model-stat-assn*$_0$)
{((*b*, *m*), (*b*′, *m*′)). *b*=*b*′ ∧ (¬*b* ⟶ (*m*,*m*′) ∈ *model-stat-rel*)}›

definition *clauses-l-assn* **where**
‹*clauses-l-assn* = *hr-comp* (*IICF-Array-of-Array-List.aal-assn unat-lit-assn*)
(*list-mset-rel O* ⟨*list-mset-rel*⟩*mset-rel*)›

theorem *IsaSAT-full-correctness*:
‹(*uncurry IsaSAT-code*, *uncurry* (λ-. *model-if-satisfiable-bounded*))
∈ [λ(-, *a*). (∀ *C*∈#*a*. ∀ *L*∈#*C*. *nat-of-lit L* ≤ *unat32-max*)]$_a$ *opts-assn*^{d} $*_a$ *clauses-l-assn*^{d} → *model-bounded-assn*›
using *IsaSAT-code.refine*[*FCOMP IsaSAT-bounded-heur-model-if-sat*′]
unfolding *model-bounded-assn-def clauses-l-assn-def*
by *auto*

end