IsaSAT: Heuristics and Code Generation

Mathias Fleury, Jasmin Blanchette, Peter Lammich

August 2, 2023

Contents

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*⟩

lemma *bitXOR-1-if-mod-2-int*: ‹*L OR 1* = (*if L mod 2* = *0 then L* + *1 else L*)› **for** *L* :: *int*
⟨*proof*⟩

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*⟩

lemma *bin-pos-same-XOR3*:
‹*a XOR a XOR c* = *c*›
‹*a XOR c XOR a* = *c*› **for** *a c* :: *int*
⟨*proof*⟩

lemma *bin-pos-same-XOR3-nat*:
‹*a XOR a XOR c* = *c*›
‹*a XOR c XOR a* = *c*› **for** *a c* :: *nat*
⟨*proof*⟩

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)›
⟨*proof*⟩
lemma *Neg-div2-iff*:
‹*Neg* ((*bb* :: *nat*) *div* 2) = *b* ⟷ *is-neg b* ∧ (*bb* = 2 ∗ *atm-of b* ∨ *bb* = 2 ∗ *atm-of b* + 1)›
⟨*proof*⟩

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)›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *nat-of-lit-lit-of-nat*[*simp*]: ‹*nat-of-lit* (*literal-of-nat n*) = *n*›
⟨*proof*⟩

lemma *atm-of-lit-of-nat*: ‹*atm-of* (*literal-of-nat n*) = *n div* 2›
⟨*proof*⟩

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))›
⟨*proof*⟩

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

⟨*proof*⟩

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*›
⟨*proof*⟩

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*))›
⟨*proof*⟩

lemma *ann-lit-of-pair-pair-of-ann-lit*: ‹*ann-lit-of-pair* (*pair-of-ann-lit L*) = *L*›
⟨*proof*⟩

lemma *pair-of-ann-lit-ann-lit-of-pair*: ‹*pair-of-ann-lit* (*ann-lit-of-pair L*) = *L*›
⟨*proof*⟩

lemma *literal-of-neq-eq-nat-of-lit-eq-iff*: ‹*literal-of-nat b* = *L* ⟷ *b* = *nat-of-lit L*›
⟨*proof*⟩

lemma *nat-of-lit-eq-iff*[*iff*]: ‹*nat-of-lit xa* = *nat-of-lit x* ⟷ *x* = *xa*›
⟨*proof*⟩

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*›
  ⟨*proof*⟩

lemma *unat64-max-uint-def*: ‹*unat* (−*1* :: *64 Word.word*) = *unat64-max*›
⟨*proof*⟩

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*⟩

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* ∈# $\mathcal{L}_{all}$ $\mathcal{A}_{in}$. *nat-of-lit L* ≤ *unat32-max*)›

definition *isasat-input-nempty* **where**
  [*simp*]: ‹*isasat-input-nempty* = (*set-mset* $\mathcal{A}_{in}$ ≠ {#})›

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-*$\mathcal{L}_{all}$*-less-unat32-max*: ‹*isasat-input-bounded*›
begin

lemma *in-*$\mathcal{L}_{all}$*-less-unat32-max'*: ‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}_{in}$ ⟹ *nat-of-lit L* ≤ *unat32-max*›
  ⟨*proof*⟩

lemma *in-*$\mathcal{A}_{in}$*-less-than-unat32-max-div-2*:
  ‹*L* ∈# $\mathcal{A}_{in}$ ⟹ *L* ≤ *unat32-max div 2*›
  ⟨*proof*⟩

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*› **and**

in-$\mathcal{L}_{all}$-less-unat32-max: ‹$\forall L \in\# \mathcal{L}_{all}\ \mathcal{A}_{in}$. *nat-of-lit* $L <$ *unat32-max* $- 1$›
shows ‹*size* $C \leq$ *unat32-max div* 2›
⟨*proof*⟩

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: ‹$\neg$*tautology* C›
shows ‹*size* $C \leq 1 +$ *unat32-max div* 2›
⟨*proof*⟩

lemma *clss-size-unat32-max*:
assumes
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}_{in}$ C› **and**
dist: ‹*distinct-mset* C›
shows ‹*size* $C \leq$ *unat32-max* $+ 2$›
⟨*proof*⟩

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: ‹$\forall L \in\# \mathcal{L}_{all}\ \mathcal{A}_{in}$. *nat-of-lit* $L <$ *unat32-max* $- 1$›
shows ‹*size* $C \leq$ *unat32-max*›
⟨*proof*⟩

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* $M \leq$ *Suc* (*unat32-max div* 2)› **and**
literals-are-in-$\mathcal{L}_{in}$*-trail-count-decided-unat32-max*:
‹*count-decided* $M \leq$ *Suc* (*unat32-max div* 2)› **and**
literals-are-in-$\mathcal{L}_{in}$*-trail-get-level-unat32-max*:
‹*get-level* M $L \leq$ *Suc* (*unat32-max div* 2)›
⟨*proof*⟩

lemma *length-trail-unat32-max-div2*:
fixes M :: ‹(*nat*, $'b$) *ann-lits*›
assumes
M-$\mathcal{L}_{all}$: ‹$\forall L \in$*set* M. *lit-of* $L \in\# \mathcal{L}_{all}\ \mathcal{A}_{in}$› **and**
n-d: ‹*no-dup* M›
shows ‹*length* $M \leq$ *unat32-max div* $2 + 1$›
⟨*proof*⟩

end

end

1.6 Instantion for code generation

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

definition *default-literal* **where**
‹*default-literal* = *Pos default*›
instance ⟨*proof*⟩

end

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

definition *default-fmap* **where**
‹*default-fmap* = *fmempty*›
instance ⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

definition *find-decomp-wl-imp* :: ‹*(nat, nat) ann-lits ⇒ nat clause ⇒ nat literal ⇒ (nat, nat) ann-lits nres*› **where**
‹*find-decomp-wl-imp = (λM₀ D L. do {*
let lev = get-maximum-level M₀ (remove1-mset (−L) D);
let k = count-decided M₀;
(-, M) ←
WHILE_T λ(j, M). j = count-decided M ∧ j ≥ lev ∧ (M = [] ⟶ j = lev) ∧ (∃ M'. M₀ = 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, M₀);
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)›
⟨*proof*⟩

lemma *count-decided-tl-if*:
‹*M ≠ [] ⟹ count-decided (tl M) = (if is-decided (hd M) then count-decided M − 1 else count-decided M)*›
⟨*proof*⟩

lemma *count-decided-butlast*:
‹*count-decided (butlast xs) = (if is-decided (last xs) then count-decided xs − 1 else count-decided xs)*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *hd-decided-count-decided-ge-1*:
‹*x ≠ [] ⟹ is-decided (hd x) ⟹ Suc 0 ≤ count-decided x*›
⟨*proof*⟩

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′* = (λ*M 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*›
⟨*proof*⟩

lemma *set-mset-all-lits-of-mm-atms-of-ms-iff*:
‹*set-mset* (*all-lits-of-mm A*) = *set-mset* (*$\mathcal{L}_{all}$ $\mathcal{A}$*) ⟷ *atms-of-ms* (*set-mset A*) = *atms-of* (*$\mathcal{L}_{all}$ $\mathcal{A}$*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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$xo = *RETURN*$(*f15*abcdefghijklmn$xo)›
‹(*RETURN* o_{16} *f16*)abcdefghijklmnxop = *RETURN*$(*f16*$abcdefghijklm$n$xo$p)›
‹(*RETURN* o_{17} *f17*)abcdefghijklmnxop$q = *RETURN*$(*f17*abcdefghijklmnxop$
‹(*RETURN* o_{18} *f18*)abcdefghijklmnxopqr = *RETURN*$(*f18*$abcdefghijklm$n$xo$
‹(*RETURN* o_{19} *f19*)abcdefghijklmnxopqr$s = *RETURN*$(*f19*abcdefghijklmn$x
⟨*proof*⟩

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*$(λ$_2$*a b*. *fp*$a$b)$p›
⟨*proof*⟩

lemma *case-prod-open-comb*[*sepref-monadify-comb*]:
‹⋀*fp p*. *case-prod-open*$*fp*$*p* ≡ *Refine-Basic.bind*$(*EVAL*$*p*)$(λ$_2$*p*. (*SP case-prod-open*$*fp*$*p*))›
⟨*proof*⟩

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*$*p*)$(λ$_2$*p*. *case-prod-open*$(λ$_2$*a b*. *EVAL*$(*fp a b*))$*p*)
⟨*proof*⟩

lemma *hn-case-prod-open'*[*sepref-comb-rules*]:
assumes *FR*: ‹Γ ⊢ *hn-ctxt* (*prod-assn P1 P2*) *p' p* ∗∗ Γ*1*›
assumes *Pair*: ⋀*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*: ‹⋀*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*$(λ$_2$*a b*. *f' a b*)$*p'*)› (**is** ‹*?G* Γ›)
⟨*proof*⟩
apply1 (*rule hn-refine-cons-pre*[*OF FR*])

apply1 (*cases p*; *cases p'*; *simp add*: *prod-assn-pair-conv*[*THEN prod-assn-ctxt*])
⟨*proof*⟩
applyS (*simp add*: *hn-ctxt-def*)
applyS *simp* ⟨*proof*⟩

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*)›
⟨*proof*⟩

lemma *op-list-list-upd-alt-def*: ‹*op-list-list-upd xss i j x* = *xss*[*i* := (*xss* ! *i*)[*j* := *x*]]›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*
⟨*proof*⟩

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

lemma *snat64-max-refine*[*sepref-import-param*]: ‹(*0x7FFFFFFFFFFFFFFF*, *snat64-max*)∈*snat-rel'* *TYPE*(*64*)›
⟨*proof*⟩

lemma *snat32-max-refine*[*sepref-import-param*]: ‹(*0x7FFFFFFF*, *snat32-max*)∈*snat-rel'* *TYPE*(*32*)›
⟨*proof*⟩

lemma *unat64-max-refine*[*sepref-import-param*]: ‹(*0xFFFFFFFFFFFFFFFF*, *unat64-max*)∈*unat-rel'* *TYPE*(*64*)›
⟨*proof*⟩

lemma *unat32-max-refine*[*sepref-import-param*]: ‹(*0xFFFFFFFF*, *unat32-max*)∈*unat-rel'* *TYPE*(*32*)›
⟨*proof*⟩

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*)› ⟨*proof*⟩
lemma *is-down′-64-32*[*simp*,*intro*!]: ‹*is-down′ UCAST*(*64* → *32*)› ⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

interpretation *atom*: *dflt-pure-option-private* ‹*2^32*−*1*› *atom-assn* ‹*ll-icmp-eq* (*2^32*−*1*)›
⟨*proof*⟩

lemma *atm-of-refine*: ‹(λx. x div 2 , atm-of) ∈ nat-lit-rel → nat-rel›
⟨*proof*⟩

sepref-def *atm-of-impl* **is** [] ‹RETURN o (λx::nat. x div 2)›
:: ‹uint32-nat-assn^k →$_a$ atom-assn›
⟨*proof*⟩

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›
⟨*proof*⟩

lemma *Neg-refine-aux*: ‹(λx. 2*x + 1,Neg)∈nat-rel → nat-lit-rel›
⟨*proof*⟩

sepref-def *Pos-impl* **is** [] ‹RETURN o Pos-rel› :: ‹atom-assn^d →$_a$ uint32-nat-assn›
⟨*proof*⟩

sepref-def *Neg-impl* **is** [] ‹RETURN o (λx. 2*x+1)› :: ‹atom-assn^d →$_a$ uint32-nat-assn›
⟨*proof*⟩

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›
⟨*proof*⟩

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›
⟨*proof*⟩

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

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*: ‹(λx. value-of-atm x, index-of-atm) ∈ nat-rel → nat-rel›
⟨*proof*⟩

sepref-def *index-of-atm-impl*
is [] ‹RETURN o (λx. value-of-atm x)›

:: ‹*atom-assn*^{d} $\rightarrow_a$ *snat-assn′ TYPE(64)*›
⟨*proof*⟩

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*]›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *index-atm-of-impl*
is ‹*RETURN o index-atm-of*›
:: ‹*unat-lit-assn*^{d} $\rightarrow_a$ *snat-assn′ TYPE(64)*›
⟨*proof*⟩

lemma *nat-of-lit-refine-aux*: ‹((λ*x*. *x*), *nat-of-lit*) ∈ *nat-lit-rel* → *nat-rel*›
⟨*proof*⟩

sepref-def *nat-of-lit-rel-impl* **is** [] ‹*RETURN o* (λ*x*::*nat*. *x*)› :: ‹*uint32-nat-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*›
⟨*proof*⟩
lemmas [*sepref-fr-rules*] = *nat-of-lit-rel-impl.refine*[*FCOMP nat-of-lit-refine-aux*]

lemma *uminus-refine-aux*: ‹(λ*x*. *x XOR 1*, *uminus*) ∈ *nat-lit-rel* → *nat-lit-rel*›
⟨*proof*⟩

sepref-def *uminus-impl* **is** [] ‹*RETURN o* (λ*x*::*nat*. *x XOR 1*)› :: ‹*uint32-nat-assn*^{k} $\rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

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

lemma *lit-eq-refine-aux*: ‹((=), (=)) ∈ *nat-lit-rel* → *nat-lit-rel* → *bool-rel*›
⟨*proof*⟩

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

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

lemma *is-pos-refine-aux*: ‹(λ*x*. *x AND 1* = *0*, *is-pos*) ∈ *nat-lit-rel* → *bool-rel*›
⟨*proof*⟩

sepref-def *is-pos-impl* **is** [] ‹*RETURN o* (λ*x*. *x AND 1* = *0*)› :: ‹*uint32-nat-assn*^{k} →$_a$ *bool1-assn*›
⟨*proof*⟩

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*› ⟨*proof*⟩

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

lemma *nat-lit-rel*: ‹((=), *op-nat-lit-eq*) ∈ *nat-lit-rel* → *nat-lit-rel* → *bool-rel*›
⟨*proof*⟩

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*› ⟨*proof*⟩

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

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

lemma *hn-aal-free-raw*: ‹(*aal-free*,*RETURN o op-list-list-free*) ∈ *AA*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-decl-impl *aal-free*: *hn-aal-free-raw*
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *of-nat-unat*:
‹(*id*,*of-nat*) ∈ *unat-rel′ TYPE*('*a*::*len2*) → *word-rel*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *tri-bool-UNSET-impl* **is** [] ‹*uncurry0* (*RETURN 0*)› :: ‹*unit-assn*^{k} →$_a$ *unat-assn′ TYPE*(*8*)›
⟨*proof*⟩

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

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

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*›
⟨*proof*⟩

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 $\times_a$
IICF-Array.array-assn atom.option-assn $\times_a$
IICF-Array.array-assn uint64-nat-assn $\times_a$ *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} $\to_a$ *atom.option-assn*›
⟨*proof*⟩

sepref-def *mop-hp-read-nxt-imp-code*
is ‹*uncurry mop-hp-read-nxt-imp*›
:: ‹*atom-assn*^{k} $*_a$ *hp-assn*^{k} $\to_a$ *atom.option-assn*›
⟨*proof*⟩

sepref-def *mop-hp-read-parent-imp-code*
is ‹*uncurry mop-hp-read-parent-imp*›
:: ‹*atom-assn*^{k} $*_a$ *hp-assn*^{k} $\to_a$ *atom.option-assn*›
⟨*proof*⟩

sepref-def *mop-hp-read-child-imp-code*
is ‹*uncurry mop-hp-read-child-imp*›
:: ‹*atom-assn*^{k} $*_a$ *hp-assn*^{k} $\to_a$ *atom.option-assn*›
⟨*proof*⟩

sepref-def *mop-hp-read-score-imp-code*
is ‹*uncurry mop-hp-read-score-imp*›
:: ‹*atom-assn*^{k} $*_a$ *hp-assn*^{k} $\to_a$ *uint64-nat-assn*›
⟨*proof*⟩

lemma *source-node-impl-alt-def*:
‹*source-node-impl* = (λ(*prevs*, *nxts*, *parents*, *children*, *scores*,*i*). *i*)›
⟨*proof*⟩

sepref-def *source-node-impl-code*
is ‹(*RETURN o source-node-impl*)›
:: ‹*hp-assn*^{k} $\to_a$ *atom.option-assn*›
⟨*proof*⟩

lemma *update-source-node-impl-alt-def*:
‹*update-source-node-impl* = (λ*i* (*prevs*, *nxts*, *parents*, *children*, *scores*,-). (*prevs*, *nxts*, *parents*, *children*, *scores*, *i*))›
⟨*proof*⟩

sepref-def *update-source-node-impl-code*
is ‹*uncurry* (*RETURN oo update-source-node-impl*)›
:: ‹*atom.option-assn*^{k} $*_a$ *hp-assn*^{d} $\to_a$ *hp-assn*›
⟨*proof*⟩

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

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

⟨*proof*⟩

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

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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 →$_a$ hp-assn ×$_a$ atom-assn*›
  ⟨*proof*⟩

sepref-def *mop-vsids-pass$_2$-imp-code*
  is ‹*uncurry mop-vsids-pass$_2$-imp*›
  :: ‹*hp-assn^d *$_a$ atom-assn^k →$_a$ hp-assn*›
  ⟨*proof*⟩

sepref-def *mop-merge-pairs-imp-code*
  is ‹*uncurry mop-merge-pairs-imp*›
  :: ‹*hp-assn^d *$_a$ atom-assn^k →$_a$ hp-assn*›
  ⟨*proof*⟩

sepref-def *mop-vsids-pop-min-impl-code*
  is *mop-vsids-pop-min-impl*
  :: ‹*hp-assn^d →$_a$ atom.option-assn ×$_a$ hp-assn*›
  ⟨*proof*⟩

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-assn^k →$_a$ atom.option-assn*›
  ⟨*proof*⟩

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*)
}
}›
⟨*proof*⟩

sepref-def *mop-unroot-hp-tree-code*
is ‹*uncurry* (*mop-unroot-hp-tree* :: (*nat*,*nat*)*pairing-heaps-imp* ⇒ -)›
:: ‹*hp-assn*^{d} ∗$_a$ *atom-assn*^{k} →$_a$ *hp-assn*›
⟨*proof*⟩

sepref-def *mop-hp-update-score-imp-code*
is ‹*uncurry2 mop-hp-update-score-imp*›
:: ‹*atom-assn*^{k} ∗$_a$ *uint64-nat-assn*^{k} ∗$_a$ *hp-assn*^{d} →$_a$ *hp-assn*›
⟨*proof*⟩

lemma *Some-eq-not-None-sepref-id-work-around*: ‹*Some h* = *a* ⟷ (*a* ≠ *None* ∧ *h* = *the a*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *mop-hp-is-in-code*
is ‹*uncurry mop-hp-is-in*›
:: ‹*atom-assn*^{k} ∗$_a$ *hp-assn*^{k} →$_a$ *bool1-assn*›
⟨*proof*⟩

sepref-def *mop-vsids-pop-min2-impl-code*
is *mop-vsids-pop-min2-impl*
:: ‹*hp-assn*^{d} →$_a$ *atom-assn* ×$_a$ *hp-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›

⟨*proof*⟩

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*
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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 ⟨*proof*⟩

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* $\longleftrightarrow i = j$›

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

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

‹*header-size C* $\Longrightarrow j \geq$ *header-size C'* $\Longrightarrow i -$ *SIZE-SHIFT* $= j -$ *SIZE-SHIFT* $\longleftrightarrow i = j$›
‹*header-size C* $\Longrightarrow j \geq$ *header-size C'* $\Longrightarrow i -$ *LBD-SHIFT* $= j -$ *LBD-SHIFT* $\longleftrightarrow i = j$›
‹*header-size C* $\Longrightarrow j \geq$ *header-size C'* $\Longrightarrow i -$ *STATUS-SHIFT* $= j -$ *STATUS-SHIFT* $\longleftrightarrow i = j$›
‹*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* $\longleftrightarrow i = j$›
⟨*proof*⟩

lemma *header-size-ge0*[*simp*]: ‹*0* $<$ *header-size x1*›
⟨*proof*⟩

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* $= (\lambda(C,$ *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*$\propto$*i*) $-$ *SIZE-SHIFT*)) $\wedge$
xarena-length(*arena*!(*header-size* (*N*$\propto$*i*) $-$ *SIZE-SHIFT*)) $+$ *2* $=$ *length* (*N*$\propto$*i*) $\wedge$

drop (*header-size* (*N*∝*i*)) *arena* = *map ALit* (*N*∝*i*)
))›
⟨*proof*⟩

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* {}›
⟨*proof*⟩

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*⟩

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*⟩

At first we had the weaker (*1*::′*a*) ≤ *i* − *j* which we replaced by (*4*::′*a*) ≤ *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*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*⟩

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*⟩

lemma *length-extra-information-mark-to-delete*[*simp*]:
‹*length* (*extra-information-mark-to-delete arena i*) = *length arena*›
⟨*proof*⟩

lemma *valid-arena-mono*: ‹*valid-arena ab ar vdom1* ⟹ *vdom2* ⊆ *vdom1* ⟹ *valid-arena ab ar vdom2*›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

Removable from addressable space **lemma** *valid-arena-remove-from-vdom*:
assumes ‹*valid-arena arena N* (*insert i vdom*)›
shows ‹*valid-arena arena N vdom*›
⟨*proof*⟩

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

lemma *MAX-LBD-alt-def*:
‹*MAX-LBD* = (*2^26−1*)›
⟨*proof*⟩

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*⟩

lemma *length-update-lbd*[*simp*]:
‹*length* (*update-lbd i lbd arena*) = *length arena*›
⟨*proof*⟩

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*⟩

lemma *xarena-active-clause-update-lbd-same*:
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* (*update-lbd* (*header-size* (*N*∝*i*)) *lbd* (*clause-slice arena N i*))
(*the* (*fmlookup N i*))›
⟨*proof*⟩

lemma *valid-arena-update-lbd*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *i*: ‹*i* ∈# *dom-m N*›
shows ‹*valid-arena* (*update-lbd i lbd arena*) *N vdom*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*⟩

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*⟩

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*))›
⟨*proof*⟩

lemma *length-update-pos*[*simp*]:
‹*length* (*update-pos-direct i pos arena*) = *length arena*›
⟨*proof*⟩

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*⟩

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*⟩

lemma *length-swap-lits*[*simp*]:
  ‹*length* (*swap-lits i k l arena*) = *length arena*›
  ⟨*proof*⟩

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*⟩

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*))›
  ⟨*proof*⟩

lemma *is-short-clause-swap*[*simp*]: ‹*is-short-clause* (*swap* (*N* ∝ *i*) *k l*) = *is-short-clause* (*N* ∝ *i*)›
  ⟨*proof*⟩

lemma *header-size-swap*[*simp*]: ‹*header-size* (*swap* (*N* ∝ *i*) *k l*) = *header-size* (*N* ∝ *i*)›
  ⟨*proof*⟩

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*⟩

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
‹$i \geq$ *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* (*clause-slice* (*append-clause-skeleton pos st used lbd C arena*) *N i*)
(*the* (*fmlookup N i*))›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)))›
⟨*proof*⟩

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*⟩

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*⟩

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*)›
⟨*proof*⟩

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*⟩

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*⟩

lemma *valid-arena-one-notin-vdomD*:
‹*valid-arena M N vdom* ⟹ *Suc 0* ∉ *vdom*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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* ∝ *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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

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*⟩

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*⟩

lemma *valid-arena-vdom-subset*:
assumes ‹*valid-arena arena N* (*set vdom*)› **and** ‹*distinct vdom*›
shows ‹*length vdom* ≤ *length arena*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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'*)›
⟨*proof*⟩

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'*)›
⟨*proof*⟩

lemma *arena-lit-pre2-arena-lit*[*dest*]:
‹*arena-lit-pre2 N i j* ⟹ *arena-lit-pre N* (*i*+*j*)›
⟨*proof*⟩

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'*)›
⟨*proof*⟩

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'*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*⟩

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: ‹$j \geq 2$› **and**
‹$(C,C') \in$ *nat-rel*› ‹$(j,j') \in$ *nat-rel*›
shows ‹*mop-arena-shorten C j arena* $\leq$ *SPEC*(λc. (c, $N(C' \hookrightarrow$ *take* j' $(N \propto C'))) \in$
$\{(arena', N)$. *valid-arena arena' N vdom* $\wedge$ *length arena* $=$ *length arena'*$\}$)›
⟨*proof*⟩

definition *arenap-update-lit* :: ‹*nat* $\Rightarrow$ *nat* $\Rightarrow$ *nat literal* $\Rightarrow$ *arena* $\Rightarrow$ *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*›
⟨*proof*⟩

lemma *valid-arena-arenap-update-lit*:
assumes *C*: ‹$C \in\#$ *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*) ($N(C \hookrightarrow (N \propto C)[j := L])$) *vdom*›
⟨*proof*⟩

definition *mop-arena-update-lit* :: ‹*nat* $\Rightarrow$ *nat* $\Rightarrow$ *nat literal* $\Rightarrow$ *arena* $\Rightarrow$ *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*: ‹$C \in\#$ *dom-m N*› **and**
j: ‹$j <$ *arena-length arena C*› **and**
valid: ‹*valid-arena arena N vdom*› **and**
‹$(j,j') \in$ *nat-rel*› **and**
‹$(C,C') \in$ *nat-rel*› **and**
‹$(L,L') \in$ *Id*›
shows
‹*mop-arena-update-lit C j L arena* $\leq$ *SPEC* (λc. (c, ($N(C' \hookrightarrow (N \propto C')[j' := L']$))) $\in$
$\{(arena', N)$. *valid-arena arena' N vdom* $\wedge$ *length arena'* $=$ *length arena*$\}$)›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *header-size-Suc-def*:
‹*header-size C* $=$
(*if is-short-clause C then* (*Suc* (*Suc 0*)) *else* (*Suc* (*Suc* (*Suc 0*))))›
⟨*proof*⟩

lemma *valid-arena-arena-set-status*:
assumes
valid: ‹*valid-arena arena N vdm*› **and**
C: ‹$C \in\#$ *dom-m N*› **and**
b: ‹$b =$ *IRRED* $\vee$ $b =$ *LEARNED*› **and**

b': ‹$b' \longleftrightarrow b = IRRED$›
shows ‹*valid-arena* (*arena-set-status arena C b*) (*fmupd C* (*N* $\propto$ *C*, *b'*) *N*) *vdm*›
⟨*proof*⟩

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*)})›
⟨*proof*⟩

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*)})›
⟨*proof*⟩

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*))))})›
⟨*proof*⟩

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* $\mathcal{A}$ *W N* = ⋃(((‘) *fst*) ‘ *set* ‘ *W* ‘ *set-mset* ($\mathcal{L}_{all}$ $\mathcal{A}$)) ∪ *set-mset* (*dom-m N*)›

lemma *vdom-m-simps*[*simp*]:

‹*bh* ∈# *dom-m N* ⟹ *vdom-m* $\mathcal{A}$ *W* (*N*(*bh* ↪ *C*)) = *vdom-m* $\mathcal{A}$ *W N*›
‹*bh* ∉# *dom-m N* ⟹ *vdom-m* $\mathcal{A}$ *W* (*N*(*bh* ↪ *C*)) = *insert bh* (*vdom-m* $\mathcal{A}$ *W N*)›
⟨*proof*⟩

lemma *vdom-m-simps2*[*simp*]:
‹*i* ∈# *dom-m N* ⟹ *vdom-m* $\mathcal{A}$ (*W*(*L* := *W L* @ [(*i*, *C*)])) *N* = *vdom-m* $\mathcal{A}$ *W N*›
‹*bi* ∈# *dom-m ax* ⟹ *vdom-m* $\mathcal{A}$ (*bp*(*L*:= *bp L* @ [(*bi*, *av*′)])) *ax* = *vdom-m* $\mathcal{A}$ *bp ax*›
⟨*proof*⟩

lemma *vdom-m-simps3*[*simp*]:
‹*fst biav*′ ∈# *dom-m ax* ⟹ *vdom-m* $\mathcal{A}$ (*bp*(*L*:= *bp L* @ [*biav*′])) *ax* = *vdom-m* $\mathcal{A}$ *bp ax*›
⟨*proof*⟩

What is the difference with the next lemma?

lemma [*simp*]:
‹*bf* ∈# *dom-m ax* ⟹ *vdom-m* $\mathcal{A}$ *bj* (*ax*(*bf* ↪ *C*′)) = *vdom-m* $\mathcal{A}$ *bj* (*ax*)›
⟨*proof*⟩

lemma *vdom-m-simps4*[*simp*]:
‹*i* ∈# *dom-m N* ⟹
vdom-m $\mathcal{A}$ (*W* (*L1* := *W L1* @ [(*i*, *C1*)], *L2* := *W L2* @ [(*i*, *C2*)])) *N* = *vdom-m* $\mathcal{A}$ *W N*›
⟨*proof*⟩

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

lemma *vdom-m-simps4*′[*simp*]:
‹*i* ∈# *dom-m N* ⟹
vdom-m $\mathcal{A}$ (*W* (*L1* := *W L1* @ [(*i*, *C1*), (*i*, *C2*)])) *N* = *vdom-m* $\mathcal{A}$ *W N*›
⟨*proof*⟩

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* $\mathcal{A}$ = (λ-. [])›

lemma *vdom-m-empty-watched*[*simp*]:
‹*vdom-m* $\mathcal{A}$ (*empty-watched* $\mathcal{A}$′) *N* = *set-mset* (*dom-m N*)›
⟨*proof*⟩

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* $\mathcal{A}$ *W* (*fmupd i C N*) = *insert i* (*vdom-m* $\mathcal{A}$ *W N*)›
⟨*proof*⟩

lemma *in-watch-list-in-vdom*:
assumes ‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and** ‹*w* < *length* (*watched-by S L*)›
shows ‹*fst* (*watched-by S L* ! *w*) ∈ *vdom-m* $\mathcal{A}$ (*get-watched-wl S*) (*get-clauses-wl S*)›
⟨*proof*⟩

lemma *in-watch-list-in-vdom*′:
assumes ‹*L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and** ‹*A* ∈ *set* (*watched-by S L*)›
shows ‹*fst A* ∈ *vdom-m* $\mathcal{A}$ (*get-watched-wl S*) (*get-clauses-wl S*)›
⟨*proof*⟩

lemma *in-dom-in-vdom*[*simp*]:
‹$x \in\# \mathit{dom\text{-}m}\ N \Longrightarrow x \in \mathit{vdom\text{-}m}\ \mathcal{A}\ W\ N$›
⟨*proof*⟩

lemma *in-vdom-m-upd*:
‹$\mathit{x1f} \in \mathit{vdom\text{-}m}\ \mathcal{A}\ (g(\mathit{x1e} := (g\ \mathit{x1e})[\mathit{x2} := (\mathit{x1f}, \mathit{x2f})]))\ b$›
if ‹$\mathit{x2} < \mathit{length}\ (g\ \mathit{x1e})$› **and** ‹$\mathit{x1e} \in\# \mathcal{L}_{all}\ \mathcal{A}$›
⟨*proof*⟩

lemma *in-vdom-m-fmdropD*:
‹$x \in \mathit{vdom\text{-}m}\ \mathcal{A}\ ga\ (\mathit{fmdrop}\ C\ baa) \Longrightarrow x \in (\mathit{vdom\text{-}m}\ \mathcal{A}\ ga\ baa)$›
⟨*proof*⟩

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* = ‹$(32\ \mathit{word}, 64)\ \mathit{array\text{-}list}$›

lemma *protected-bind-assoc*:
‹*Refine-Basic.bind*\$(*Refine-Basic.bind*\$*m*\$($\lambda_2 x$. *f x*))\$($\lambda_2 y$. *g y*) =
Refine-Basic.bind\$*m*\$($\lambda_2 x$. *Refine-Basic.bind*\$(*f x*)\$($\lambda_2 y$. *g y*))› ⟨*proof*⟩

lemma *convert-swap*: ‹*WB-More-Refinement-List.swap* = *More-List.swap*›
⟨*proof*⟩

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*› ⟨*proof*⟩
sepref-def *xarena-lit-impl* [*llvm-inline*]
is [] ‹*RETURN o* (λ*eli*. *eli*)› :: ‹*uint32-nat-assn*^{k} →$_a$ *uint32-nat-assn*› ⟨*proof*⟩
lemmas [*sepref-fr-rules*] = *xarena-lit-impl.refine*[*FCOMP xarena-lit-refine1*]

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

LBD

lemma *xarena-lbd-refine1*: ‹(λeli. eli >> 5, xarena-lbd) ∈ [*is-Status*]$_f$ arena-el-rel → nat-rel›
⟨proof⟩

sepref-def *xarena-lbd-impl* [*llvm-inline*]
is [] ‹(RETURN o (λeli. eli >> 5))› :: ‹uint32-nat-assn^k →$_a$ uint32-nat-assn›
⟨proof⟩

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

Size

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

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

Position

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

lemma *APos-refine1*: ‹(λx. x,APos) ∈ nat-rel → arena-el-rel› ⟨proof⟩
sepref-def *APos-impl* [*llvm-inline*] **is** [] ‹RETURN o (λx. x)› :: ‹uint32-nat-assn^k →$_a$ uint32-nat-assn›
⟨proof⟩
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*: ‹(λeli. eli AND 0b11, xarena-status) ∈ [*is-Status*]$_f$ arena-el-rel → status-rel› ⟨proof⟩
sepref-def *xarena-status-impl* [*llvm-inline*] **is** [] ‹RETURN o (λeli. eli AND 0b11)› :: ‹uint32-nat-assn^k
→$_a$ uint32-nat-assn›
⟨proof⟩
lemmas [*sepref-fr-rules*] = *xarena-status-impl.refine*[*FCOMP xarena-status-refine1*]

lemma *xarena-used-refine1*: ‹(λeli. (eli AND 0b1100) >> 2, xarena-used) ∈ [*is-Status*]$_f$ arena-el-rel
→ nat-rel›
⟨proof⟩

lemma *is-down′-32-2*[*simp*]: ‹*is-down′ UCAST*(32 → 2)›
⟨proof⟩

lemma *bitAND-mod*: ‹*L AND* (*2^n* − *1*) = *L mod* (*2^n*)› **for** *L* :: *nat*
⟨*proof*⟩

lemma *nat-ex-numeral*: ‹∃ *m*. *n*=*0* ∨ *n* = *numeral m*› **for** *n* :: *nat*
⟨*proof*⟩

lemma *xarena-used-implI*: ‹*x AND 12* >> *2* < *max-unat 2*› **for** *x* :: *nat*
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *bang-eq-nat*:
fixes *x* :: *nat*
shows (*x* = *y*) = (∀ *n*. *x* !! *n* = *y* !! *n*)
⟨*proof*⟩

lemma *sum-bitAND-shift-pow2*:
‹(*a* + (*b* << (*n* + *m*))) *AND* (*2^n* − *1*) = *a AND* (*2^n* − *1*)› **for** *a b n* :: *nat*
⟨*proof*⟩

lemma *and-bang-nat*: ‹(*x AND y*) !! *n* = (*x* !! *n* ∧ *y* !! *n*)› **for** *x y n* :: *nat*
⟨*proof*⟩

lemma *AND-12-AND-15-AND-12*: ‹*a AND 12* = (*a AND 15*) *AND 12*› **for** *a* :: *nat*
⟨*proof*⟩

lemma *AStatus-shift-safe*:

‹$c \geq 2 \Longrightarrow x42 + (x43 << c)$ *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›
‹$x42 \leq 3 \Longrightarrow$ *Suc* $((x42 << 2) + (x43 << 5)) >> 5 = x43$›
for $x42$ $x43$ $x53$ :: *nat*
⟨*proof*⟩

lemma *less-unat-AND-shift*: ‹$x42 < 2\hat{}n \Longrightarrow x42 >> n = 0$› **for** $x42$ :: *nat*
⟨*proof*⟩

lemma [*simp*]: ‹$(a + (w << n)) >> n = (a >> n) + w$› ‹$((w << n)) >> n = w$›
‹$n \leq m \Longrightarrow ((w << n)) >> m = w >> (m - n)$›
‹$n \geq m \Longrightarrow ((w << n)) >> m = w << (n - m)$› **for** w n :: *nat*
⟨*proof*⟩

lemma *less-numeral-pred*:
‹$a \leq$ *numeral* $b \longleftrightarrow a =$ *numeral* $b \vee a \leq$ *pred-numeral* b› **for** a :: *nat*
⟨*proof*⟩

lemma *nat-shiftl-numeral* [*simp*]:
(*numeral* w :: *nat*) << *numeral* $w' =$ *numeral* (*num.Bit0* w) << *pred-numeral* w'
⟨*proof*⟩

lemma *nat-shiftl-numeral'* [*simp*]:
(*numeral* w :: *nat*) << $1 =$ *numeral* (*num.Bit0* w)
(1 :: *nat*) << $n = 2\hat{}n$
⟨*proof*⟩

lemma *shiftr-nat-alt-def*: ‹(*a* :: *nat*) >> *b* = *nat* (*int a* >> *b*)›
⟨*proof*⟩

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'
⟨*proof*⟩

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
⟨*proof*⟩

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

⟨*proof*⟩

lemma *nat-numeral-and-one*: ‹(*1* :: *nat*) *AND 1* = *1*›
⟨*proof*⟩

lemma *AStatus-refine1*: ‹(*AStatus-impl1*, *AStatus*) ∈ *status-rel* → *br id* (λ*n*. *n* ≤ *3*) → *nat-rel* → *arena-el-rel*›
⟨*proof*⟩

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*⟩

lemma *nat-shiftr-mono*: ‹*a* < *b* ⟹ *a* >> *n* < *b* >> *n*› **for** *a b* :: *nat*
⟨*proof*⟩

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*⟩

lemma *AStatus-implI2*: ‹(*ac* :: *2 word*, *ba*) ∈ *unat-rel* ⟹ *ba* << (*2*::*nat*) < *max-unat* (*32*::*nat*)›
⟨*proof*⟩

lemma *is-up-2-32*[*simp*]: ‹*is-up′ UCAST*(*2* → *32*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*}›
⟨*proof*⟩

lemma *unat-rel-2-br*: ‹(((*unat-rel* :: (*2 word* × -) *set*) *O br id* (λ*n*. *n* ≤ *3*))) = ((*unat-rel*))›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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)›
⟨*proof*⟩

Literal at given position **lemma** *arena-lit-implI*:
assumes ‹*arena-lit-pre a b*›
shows ‹*b* < *length a*› ‹*is-Lit* (*a* ! *b*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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} → *unat-lit-assn*›
⟨*proof*⟩

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))›
⟨*proof*⟩

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} → *status-impl-assn*›
⟨*proof*⟩

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} → *arena-fast-assn*›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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} →*uint32-nat-assn*›
⟨*proof*⟩

sepref-def *mop-arena-lbd-impl*
is ‹*uncurry mop-arena-lbd*›
:: ‹*arena-fast-assn*^{k} ∗$_a$ *sint64-nat-assn*^{k} →$_a$ *uint32-nat-assn*›
⟨*proof*⟩

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} → *unat-assn′ TYPE*(*2*)›
⟨*proof*⟩

Get Saved Position **lemma** *arena-posI*:
assumes ‹*get-saved-pos-pre a b*›
shows ‹*3 ≤ b*›
and ‹*b < length a*›
and ‹*is-Pos* (*a* ! (*b* − *3*))›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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} → *snat-assn′ TYPE*(*64*)›
⟨*proof*⟩

Update LBD **lemma** *update-lbdI*:
assumes ‹*update-lbd-pre* ((*b*, *lbd*), *a*)›
shows ‹*2 ≤ b*›
and ‹*b* −*2 < length a*›
and ‹*arena-is-valid-clause-vdom a b*›
and ‹*get-clause-LBD-pre a b*›
⟨*proof*⟩

lemma *shorten-lbd-le*: ‹*shorten-lbd baa* << *5* < *max-unat 32*›
⟨*proof*⟩

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*›

⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

Update Saved Position **lemma** *update-posI*:
assumes ‹*isa-update-pos-pre* ((*b*, *pos*), *a*)›
shows ‹*3* ≤ *b*› ‹*2* ≤ *pos*› ‹*b*−*3* < *length a*›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

sepref-register *IRRED LEARNED DELETED*
lemma *IRRED-impl*[*sepref-import-param*]: ‹(*0*,*IRRED*) ∈ *status-impl-rel*›
⟨*proof*⟩

lemma *LEARNED-impl*[*sepref-import-param*]: ‹(*1*,*LEARNED*) ∈ *status-impl-rel*›
⟨*proof*⟩

lemma *DELETED-impl*[*sepref-import-param*]: ‹(*3*,*DELETED*) ∈ *status-impl-rel*›
⟨*proof*⟩

lemma *mark-garbageI*:
assumes ‹*mark-garbage-pre* (*a*, *b*)›
shows ‹*2* ≤ *b*› ‹*b*−*2* < *length a*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *bit-shiftr-shiftl-same-le*:
‹*a* << *b* >> *b* ≤ *a*› **for** *a b c* :: *nat*

⟨*proof*⟩

lemma *bit-shiftl-shiftr-same-le*:
‹*a* >> *b* << *b* ≤ *a*› **for** *a b c* :: *nat*
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

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*])})›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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))*›
⟨*proof*⟩

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)*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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

lemma *arena-other-watched-as-swap-arena-other-watched*:
assumes
N: ‹(*N*, *N'*) ∈ ⟨*arena-el-rel*⟩*list-rel*› **and**
L: ‹(*L*, *L'*) ∈ *nat-lit-rel*› **and**
C: ‹(*C*, *C'*) ∈ *nat-rel*› **and**
i: ‹(*i*, *i'*) ∈ *nat-rel*›
shows
‹*arena-other-watched-as-swap N L C i* ≤ ⇓*nat-lit-rel*
(*arena-other-watched N' L' C' i'*)›
⟨*proof*⟩

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 $\to_a$ *uint32-nat-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *arena-fast-al-unat-assn*:
‹*hr-comp* (*al-assn unat-assn*) (⟨*arena-el-rel*⟩*list-rel*) = *arena-fast-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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} →$_a$ *arena-fast-assn*›
⟨*proof*⟩

end

sepref-def *mop-arena-length-impl*
is ‹*uncurry mop-arena-length*›
:: ‹*arena-fast-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} →$_a$ *sint64-nat-assn*›
⟨*proof*⟩

sepref-def *mop-arena-status-impl*
is ‹*uncurry mop-arena-status*›
:: ‹*arena-fast-assn*^{k} *$*_a$ *sint64-nat-assn*^{k} →$_a$ *status-impl-assn*›
⟨*proof*⟩

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

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*›

⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

lemma *fm-add-new-append-clause*:
‹*fm-add-new b C N* ≤ *RETURN* (*append-clause b C N*, *length N* + *header-size C*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)
}›
⟨*proof*⟩

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*⟩

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(λ*new-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*⟩

lemma *size-learned-clss-dom-m*: ‹*size* (*learned-clss-l N*) ≤ *size* (*dom-m N*)›
⟨*proof*⟩

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*⟩

lemma *valid-arena-size-dom-m-le-arena*: ‹*valid-arena arena N vdom* ⟹ *size* (*dom-m N*) ≤ *length arena*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *header-size-code*
is ‹*RETURN o header-size*›
:: ‹*clause-ll-assn*^{k} →$_a$ *sint64-nat-assn*›
⟨*proof*⟩

lemma *header-size-bound*: ‹*header-size x* ≤ *MAX-HEADER-SIZE*› ⟨*proof*⟩

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

‹*length b* + *length baa* + *MAX-HEADER-SIZE* ≤ *snat64-max* ⟹ *length b* + *header-size baa* < *max-snat 64*›
⟨*proof*⟩

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 →
arena-fast-assn ×$_a$ *sint64-nat-assn*›
⟨*proof*⟩

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*$_o$ ≤ *snat64-max* ∧ *length arena* + *arena-length arena*$_o$ *n* +
(*if arena-length arena*$_o$ *n* ≤ *4 then MIN-HEADER-SIZE else MAX-HEADER-SIZE*) ≤ *snat64-max*]$_a$
sint64-nat-assn^k *$*_a$ *arena-fast-assn*^{k} *$*_a$ *arena-fast-assn*^{d} → *arena-fast-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *control-stack-le-length-M*:
‹*control-stack cs M ⟹ c∈set cs ⟹ c < length M*›
⟨*proof*⟩

lemma *control-stack-propa*[*simp*]:
‹*control-stack cs (Propagated x21 x22 # list) ⟷ control-stack cs list*›
⟨*proof*⟩

lemma *control-stack-filter-map-nth*:
‹*control-stack cs M ⟹ filter is-decided (rev M) = map (nth (rev M)) cs*›
⟨*proof*⟩

lemma *control-stack-empty-cs*[*simp*]: ‹*control-stack* [] *M ⟷ count-decided M = 0*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*⟩

lemma *control-stack-alt-def*: ‹*no-dup M* ⟹ *control-stack′ cs M* ⟷ *control-stack cs M*›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

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*⟩

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*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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

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*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

lemma (**in** −) *get-level-get-level-atm*: ‹*get-level M L* = *get-level-atm M* (*atm-of L*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *polarity-pol-pre*:
‹(*M'*, *M*) ∈ *trail-pol* $\mathcal{A}$ ⟹ *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹ *polarity-pol-pre M' L*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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'*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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'*)›
⟨*proof*⟩

lemma *undefined-lit-count-decided-unat32-max*:
assumes
M-$\mathcal{L}_{all}$: ‹∀ *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*⟩

lemma *length-trail-unat32-max*:
assumes
M-$\mathcal{L}_{all}$: ‹∀ *L*∈*set M*. *lit-of L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and** *n-d*: ‹*no-dup M*› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
shows ‹*length M* ≤ *unat32-max*›
⟨*proof*⟩

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'*)›
⟨*proof*⟩

lemma *control-stack-dec-butlast*:
‹*control-stack b* (*Decided x1* # *M's*) ⟹ *control-stack* (*butlast b*) *M's*›
⟨*proof*⟩

lemma *tl-trail-tr*:
‹((*RETURN o tl-trailt-tr*), (*RETURN o tl*)) ∈
[λ*M*. *M* ≠ [] ∧ *count-decided M* > *0*]$_f$ *trail-pol* 𝒜 → ⟨*trail-pol* 𝒜⟩*nres-rel*›
⟨*proof*⟩

lemma *tl-trailt-tr-pre*:
assumes ‹*M* ≠ []›
‹(*M'*, *M*) ∈ *trail-pol* 𝒜›
shows ‹*tl-trailt-tr-pre M'*›
⟨*proof*⟩

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*) ∈ *trail-pol* 𝒜› **and**
‹*M* ≠ []›
shows ‹*tl-trail-propedt-tr-pre M'*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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'*)›
⟨*proof*⟩

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*)) ∈

$[\lambda(M,\ L).\ Pos\ L \in\#\ \mathcal{L}_{all}\ \mathcal{A}]_f$ *trail-pol* $\mathcal{A} \times_r$ *Id* $\to \langle$*bool-rel*$\rangle$ *nres-rel*$\rangle$ (**is** *?A*) **and**
defined-atm-pol-pre:
$\langle(M',\ M) \in$ *trail-pol* $\mathcal{A} \Longrightarrow L \in\#\ \mathcal{A} \Longrightarrow$ *defined-atm-pol-pre* $M'\ L\rangle$
⟨*proof*⟩

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* $\mathcal{A} \times_r$ *Id* → ⟨⟨*nat-rel*⟩*option-rel*⟩ *nres-rel*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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* $\mathcal{A} \times_r$ *Id* → ⟨⟨*nat-rel*⟩*option-rel*⟩ *nres-rel*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*⟩

lemma *tl-trail-tr-no-CS*:
‹((*RETURN o tl-trailt-tr-no-CS*), (*RETURN o tl*)) ∈

$[\lambda M.\ M \neq [] \wedge \textit{count-decided}\ M > 0]_f\ \textit{trail-pol-no-CS}\ \mathcal{A} \to \langle \textit{trail-pol-no-CS}\ \mathcal{A} \rangle \textit{nres-rel}\rangle$
⟨*proof*⟩

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* $\mathcal{A} \to_f$ ⟨*trail-pol-no-CS* $\mathcal{A}$⟩*nres-rel*›
⟨*proof*⟩

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* $\mathcal{A} \to$ ⟨*trail-pol* $\mathcal{A}$⟩*nres-rel*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

lemma *control-stack-is-decided*:
‹*control-stack cs M* ⟹ *c*∈*set cs* ⟹ *is-decided* ((*rev M*)!*c*)›
⟨*proof*⟩

lemma *control-stack-distinct*:
‹*control-stack cs M* ⟹ *distinct cs*›
⟨*proof*⟩

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*⟩

definition *get-pos-of-level-in-trail* **where**
‹*get-pos-of-level-in-trail* M_0 *lev* =
SPEC(λ*i*. *i* < *length* M_0 ∧ *is-decided* (*rev* M_0!*i*) ∧ *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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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* $\mathcal{A}$)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›

⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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* A φ ⟷ (∀ *L*∈*atms-of* (*L*$_{all}$ 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* A (*save-phase L* φ) ⟷ *phase-saving* A φ›
  ⟨*proof*⟩

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*⟩

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′
 }›
  ⟨*proof*⟩

lemma *copy-phase-spec*:
  ‹*length* φ = *length* φ′ ⟹ *copy-phase* φ φ′ ≤ *SPEC*(λψ. *length* ψ = *length* φ)›
  ⟨*proof*⟩

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)›
  ⟨*proof*⟩

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* $\varphi \leq$ *SPEC*($\lambda\psi$. *length* ψ = *length* φ)›
⟨*proof*⟩

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* $\mathcal{A}$ φ›
  shows ‹*reset-target-phase* φ ≤ ⇓*Id* (*SPEC*(*phase-save-heur-rel* $\mathcal{A}$))›
  ⟨*proof*⟩

lemma *reset-best-phase-spec*:
  assumes ‹*phase-save-heur-rel* $\mathcal{A}$ φ›
  shows ‹*reset-best-phase* φ ≤ ⇓*Id* (*SPEC*(*phase-save-heur-rel* $\mathcal{A}$))›
  ⟨*proof*⟩

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* φ ≤ ⇓*Id* (*SPEC*(*phase-save-heur-rel* $\mathcal{A}$))›
⟨*proof*⟩

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* A φ›
shows ‹*phase-save-phase n* φ ≤ ⇓*Id* (*SPEC*(*phase-save-heur-rel* A))›
⟨*proof*⟩

definition *get-next-phase-pre* :: ‹*bool* ⇒ *nat* ⇒ *phase-save-heur* ⇒ *bool*› **where**
‹*get-next-phase-pre* = (λ*use-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* = (λ*use-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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*,*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*}
⟨*proof*⟩

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* $\mathcal{A}$ = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, φ, -, -, *lit-st*, *schedule*). *phase-save-heur-rel* $\mathcal{A}$ φ ∧ *phase-saving* $\mathcal{A}$ *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⟩

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* A *heur* ⟹ *heuristic-rel-stats* A (*incr-restart-phase-stats heur*)›
‹*heuristic-rel-stats* A *heur* ⟹ *heuristic-rel-stats* A (*save-phase-heur-stats L b heur*)›
‹*heuristic-rel-stats* A *heur* ⟹ *heuristic-rel-stats* A (*mark-added-heur-stats L b heur*)›
‹*heuristic-rel-stats* A *heur* ⟹ *heuristic-rel-stats* A (*heuristic-reluctant-tick-stats heur*)›
‹*heuristic-rel-stats* A *heur* ⟹ *heuristic-rel-stats* A (*heuristic-reluctant-enable-stats heur*)›
‹*heuristic-rel-stats* A *heur* ⟹ *heuristic-rel-stats* A (*heuristic-reluctant-untrigger-stats heur*)›
‹*heuristic-rel-stats* A *heur* ⟹ *heuristic-rel-stats* A (*set-fully-propagated-heur-stats heur*)›
‹*heuristic-rel-stats* A *heur* ⟹ *heuristic-rel-stats* A (*unset-fully-propagated-heur-stats heur*)›
‹*heuristic-rel-stats* A *heur* ⟹ *heuristic-rel-stats* A (*swap-emas-stats heur*)›
⟨*proof*⟩

lemma *heuristic-rel-stats-heuristic-reluctant-triggered-statsD*:
‹*heuristic-rel-stats* A *heur* ⟹
heuristic-rel-stats A (*fst* (*heuristic-reluctant-triggered-stats heur*))›
⟨*proof*⟩

lemma *save-phase-heur-pre-statsI*:
‹*heuristic-rel-stats* A *heur* ⟹ *a* ∈# A ⟹ *save-phase-heur-pre-stats a b heur*›
⟨*proof*⟩

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* ∘ *get-restart-heuristics*›

definition *current-restart-phase* :: ‹*isasat-restart-heuristics* ⇒ *64 word*› **where**
‹*current-restart-phase* = *current-restart-phase-stats* ∘ *get-restart-heuristics*›

definition *incr-restart-phase* :: ‹*isasat-restart-heuristics* ⇒ *isasat-restart-heuristics*› **where**
‹*incr-restart-phase* = *Restart-Heuristics* ∘ *incr-restart-phase-stats* ∘ *get-restart-heuristics*›

definition *incr-wasted* :: ‹*64 word* ⇒ *isasat-restart-heuristics* ⇒ *isasat-restart-heuristics*› **where**
‹*incr-wasted waste* = *Restart-Heuristics* ∘ *incr-wasted-stats waste* ∘ *get-restart-heuristics*›

definition *set-zero-wasted* :: ‹*isasat-restart-heuristics* ⇒ *isasat-restart-heuristics*› **where**
‹*set-zero-wasted* = *Restart-Heuristics* ∘ *set-zero-wasted-stats* ∘ *get-restart-heuristics*›

definition *wasted-of* :: ‹*isasat-restart-heuristics* ⇒ *64 word*› **where**
‹*wasted-of* = *wasted-of-stats* ∘ *get-restart-heuristics*›

definition *swap-emas* :: ‹*isasat-restart-heuristics* ⇒ *isasat-restart-heuristics*› **where**
‹*swap-emas* = *Restart-Heuristics* ∘ *swap-emas-stats* ∘ *get-restart-heuristics*›

definition *heuristic-rel* :: ‹*nat multiset* ⇒ *isasat-restart-heuristics* ⇒ *bool*› **where**
‹*heuristic-rel* $\mathcal{A}$ = *heuristic-rel-stats* $\mathcal{A}$ ∘ *get-restart-heuristics*›

definition *save-phase-heur* :: ‹*nat* ⇒ *bool* ⇒ *isasat-restart-heuristics* ⇒ *isasat-restart-heuristics*› **where**
‹*save-phase-heur L b* = *Restart-Heuristics* ∘ *save-phase-heur-stats L b* ∘ *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* ∘ *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* ∘ *mark-added-heur-stats L b* ∘ *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* ∘ *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* ∘ *reset-added-heur-stats* ∘ *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* $\mathcal{A}$ *heur* $\Longrightarrow$ *heuristic-rel* $\mathcal{A}$ (*incr-wasted wast heur*)›
‹*heuristic-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *heuristic-rel* $\mathcal{A}$ (*set-zero-wasted heur*)›
‹*heuristic-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *heuristic-rel* $\mathcal{A}$ (*incr-restart-phase heur*)›
‹*heuristic-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *heuristic-rel* $\mathcal{A}$ (*save-phase-heur L b heur*)›
‹*heuristic-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *heuristic-rel* $\mathcal{A}$ (*heuristic-reluctant-tick heur*)›
‹*heuristic-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *heuristic-rel* $\mathcal{A}$ (*heuristic-reluctant-enable heur*)›
‹*heuristic-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *heuristic-rel* $\mathcal{A}$ (*heuristic-reluctant-untrigger heur*)›
‹*heuristic-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *heuristic-rel* $\mathcal{A}$ (*set-fully-propagated-heur heur*)›
‹*heuristic-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *heuristic-rel* $\mathcal{A}$ (*unset-fully-propagated-heur heur*)›
‹*heuristic-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *heuristic-rel* $\mathcal{A}$ (*mark-added-heur L b heur*)›
‹*heuristic-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *heuristic-rel* $\mathcal{A}$ (*swap-emas heur*)›
⟨*proof*⟩

lemma *heuristic-rel-heuristic-reluctant-triggeredD*:
‹*heuristic-rel* $\mathcal{A}$ *heur* $\Longrightarrow$
heuristic-rel $\mathcal{A}$ (*fst* (*heuristic-reluctant-triggered heur*))›
⟨*proof*⟩

lemma *save-phase-heur-preI*:
‹*heuristic-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *a* $\in\#$ $\mathcal{A}$ $\Longrightarrow$ *save-phase-heur-pre a b heur*›
⟨*proof*⟩

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* $\Longrightarrow$ *heuristic-rel* $\mathcal{A}$ (*incr-restart-phase-end lcount heur*)›
⟨*proof*⟩

lemma [*intro*!]:
‹*heuristic-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *heuristic-rel* $\mathcal{A}$ (*heuristic-reluctant-disable heur*)›
‹*heuristic-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *heuristic-rel* $\mathcal{A}$ (*schedule-next-pure-lits heur*)›
‹*heuristic-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *heuristic-rel* $\mathcal{A}$ (*heuristic-reluctant-disable heur*)›
‹*heuristic-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *heuristic-rel* $\mathcal{A}$ (*schedule-next-reduce delta heur*)›
⟨*proof*⟩

lemma [*simp*]:
‹*heuristic-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *heuristic-rel* $\mathcal{A}$ (*restart-info-restart-done-heur heur*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›

⟨*proof*⟩

lemma [*simp*]: ‹*clss-size-resetUS* (*clss-size-incr-lcountUE st*) =
clss-size-incr-lcountUE (*clss-size-resetUS st*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*›
⟨*proof*⟩

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'*)›
⟨*proof*⟩

lemma *print-trail-print-trail2-rel*:
‹(*print-trail*, *print-trail2*) ∈ *Id* →$_f$ ⟨*unit-rel*⟩*nres-rel*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)›

⟨*proof*⟩

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*›

⟨*proof*⟩

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*)›
⟨*proof*⟩

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›
⟨*proof*⟩

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* = (λ*b 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* = (λ*b 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* = (λ*b 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*›
⟨*proof*⟩

sepref-def *opts-rel-reduce-code*
is ‹*RETURN o opts-rel-reduce*›
:: ‹*opts-rel-assn*^{k} →$_a$ *bool1-assn*›
⟨*proof*⟩

sepref-def *opts-rel-unbounded-mode-code*
is ‹*RETURN o opts-rel-unbounded-mode*›
:: ‹*opts-rel-assn*^{k} →$_a$ *bool1-assn*›
⟨*proof*⟩

sepref-def *opts-rel-miminum-between-restart-code*
is ‹*RETURN o opts-rel-miminum-between-restart*›
:: ‹*opts-rel-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

sepref-def *opts-rel-restart-coeff1-code*
is ‹*RETURN o opts-rel-restart-coeff1*›
:: ‹*opts-rel-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

sepref-def *opts-rel-restart-coeff2-code*
is ‹*RETURN o opts-rel-restart-coeff2*›
:: ‹*opts-rel-assn*^{k} →$_a$ *snat-assn'* *TYPE*(*64*)›
⟨*proof*⟩

sepref-def *opts-rel-target-code*
is ‹*RETURN o opts-rel-target*›
:: ‹*opts-rel-assn*^{k} →$_a$ *word-assn'* *TYPE*(*3*)›
⟨*proof*⟩

sepref-def *opts-rel-fema-code*
is ‹*RETURN o opts-rel-fema*›
:: ‹*opts-rel-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *opts-rel-sema-code*
is ‹*RETURN o opts-rel-sema*›
:: ‹*opts-rel-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *opts-rel-GC-units-lim-code*
is ‹*RETURN o opts-rel-GC-units-lim*›
:: ‹*opts-rel-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *opts-rel-subsumption-code*
is ‹*RETURN o opts-rel-subsumption*›
:: ‹*opts-rel-assn*^{k} →$_a$ *bool1-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *ema-bitshifting-impl*
is ‹*uncurry0* (*RETURN ema-bitshifting*)›
:: ‹*unit-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

lemma *ema-reinit-inline*[*llvm-inline*]:
ema-reinit = (λ(*value*, α, β, *wait*, *period*).
(*value*, α, *ema-bitshifting*, *1*::- *word*, *0*:: - *word*))
⟨*proof*⟩

sepref-def *EMA-MULT-SHIFT-impl*
is ‹*uncurry0* (*RETURN EMA-MULT-SHIFT*)›
:: ‹*unit-assn*^{k} →$_a$ *uint64-nat-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *ema-init-impl*
is ‹*RETURN o ema-init*›
:: ‹*word64-assn*^{k} →$_a$ *ema-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *rephase-random-impl*
is ‹*uncurry rephase-random*›
:: ‹*word-assn*^{k} *$*_a$ *phase-saver-assn*^{d} →$_a$ *phase-saver-assn*›
⟨*proof*⟩

sepref-def *rephase-init-impl*
is ‹*uncurry rephase-init*›
:: ‹*bool1-assn*^{k} *$*_a$ *phase-saver-assn*^{d} →$_a$ *phase-saver-assn*›
⟨*proof*⟩

sepref-def *copy-phase-impl*
is ‹*uncurry copy-phase*›
:: ‹*phase-saver-assn*^{k} *$*_a$ *phase-saver′-assn*^{d} →$_a$ *phase-saver′-assn*›
⟨*proof*⟩

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} →$_a$ *phase-saver-assn*›

⟨*proof*⟩

sepref-def *rephase-flipped-impl*
is ‹*rephase-flipped*›
:: ‹*phase-saver-assn*^{d} →$_a$ *phase-saver-assn*›
⟨*proof*⟩

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} →$_a$ *phase-heur-assn*›
⟨*proof*⟩

sepref-def *reset-target-phase-impl*
is ‹*reset-target-phase*›
:: ‹*phase-heur-assn*^{d} →$_a$ *phase-heur-assn*›
⟨*proof*⟩

sepref-def *phase-save-phase-impl*
is ‹*uncurry phase-save-phase*›
:: ‹*word64-assn*^{k} *$_a$ *phase-heur-assn*^{d} →$_a$ *phase-heur-assn*›
⟨*proof*⟩

sepref-def *get-next-phase-imp*
is ‹*uncurry2 get-next-phase-stats*›
:: ‹*bool1-assn*^{k} *$_a$ *atom-assn*^{k} *$_a$ *phase-heur-assn*^{k} →$_a$ *bool1-assn*›
⟨*proof*⟩

sepref-register *current-phase-letter*
sepref-def *current-phase-letter-impl*
is ‹*RETURN o current-phase-letter*›
:: ‹*word64-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

schematic-goal *mk-free-reluctant-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE reluctant-assn ?fr*›
⟨*proof*⟩

lemma [*safe-constraint-rules*]:
‹*CONSTRAINT Sepref-Basic.is-pure reluctant-rel-assn*›
‹*CONSTRAINT Sepref-Basic.is-pure reluctant-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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}
→$_a$ *reluctant-rel-assn*›
⟨*proof*⟩

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} $\rightarrow_a$ *bool1-assn*›
⟨*proof*⟩

sepref-def *reluctant-triggered-impl*
is ‹*RETURN o reluctant-c-triggered*›
:: ‹*reluctant-rel-assn*^{k} $\rightarrow_a$ *bool1-assn*›
⟨*proof*⟩

sepref-def *reluctant-u-impl*
is ‹*RETURN o reluctant-c-u*›
:: ‹*reluctant-rel-assn*^{k} $\rightarrow_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *reluctant-v-impl*
is ‹*RETURN o reluctant-c-v*›
:: ‹*reluctant-rel-assn*^{k} $\rightarrow_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *reluctant-wait-impl*
is ‹*RETURN o reluctant-c-wait*›
:: ‹*reluctant-rel-assn*^{k} $\rightarrow_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *reluctant-period-impl*
is ‹*RETURN o reluctant-c-period*›
:: ‹*reluctant-rel-assn*^{k} $\rightarrow_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *reluctant-limit-impl*
is ‹*RETURN o reluctant-c-limit*›
:: ‹*reluctant-rel-assn*^{k} $\rightarrow_a$ *word64-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)))›
⟨*proof*⟩

sepref-register ‹(*AND*) :: *'a* :: *len word* ⇒ - ⇒ -›
sepref-def *reluctant-tick-impl*
is ‹*RETURN o reluctant-tick*›
:: ‹*reluctant-assn*^{k} →$_a$ *reluctant-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *reluctant-set-trigger-impl*
is ‹*uncurry* (*RETURN oo reluctant-set-trigger*)›
:: ‹*bool1-assn*^{k} ∗$_a$ *reluctant-assn*^{k} →$_a$ *reluctant-assn*›
⟨*proof*⟩

sepref-def *reluctant-triggered-ether-impl*
is ‹(*RETURN o reluctant-triggered*)›
:: ‹*reluctant-assn*^{k} →$_a$ *reluctant-assn* ×$_a$ *bool1-assn*›
⟨*proof*⟩

sepref-def *reluctant-triggered2-impl*
is ‹(*RETURN o reluctant-triggered2*)›
:: ‹*reluctant-assn*^{k} →$_a$ *bool1-assn*›
⟨*proof*⟩

sepref-def *reluctant-untrigger-impl*
is ‹(*RETURN o reluctant-untrigger*)›
:: ‹*reluctant-assn*^{k} →$_a$ *reluctant-assn*›
⟨*proof*⟩

sepref-def *reluctant-disable-impl*

is ‹*RETURN o reluctant-disable*›
:: ‹*reluctant-assn*^{k} →$_a$ *reluctant-assn*›
⟨*proof*⟩

sepref-def *reluctant-init-impl*
is ‹*uncurry0* (*RETURN reluctant-init*)›
:: ‹*unit-assn*^{k} →$_a$ *reluctant-assn*›
⟨*proof*⟩

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
⟨*proof*⟩
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*)]
⟨*proof*⟩

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')
⟨*proof*⟩

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
⟨*proof*⟩

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
  }
  ⟨proof⟩
```

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
}
  ⟨proof⟩
```

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)
}
```

⟨*proof*⟩

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*)
}
⟨*proof*⟩

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*) ⇒

$('n \Rightarrow - \Rightarrow llvm\text{-}amemory \Rightarrow bool) \Rightarrow$
$('o \Rightarrow - \Rightarrow llvm\text{-}amemory \Rightarrow bool) \Rightarrow$
$('p \Rightarrow - \Rightarrow llvm\text{-}amemory \Rightarrow bool) \Rightarrow$
$('a, 'b, 'c, 'd, 'e, 'f, 'g, 'h, 'i, 'j,$
$'k, 'l, 'm, 'n, 'o, 'p)\ tuple16 \Rightarrow - \Rightarrow assn\rangle$ **where**
$\langle tuple16\text{-}assn\ a\text{-}assn\ b\text{-}assn'\ c\text{-}assn\ d\text{-}assn\ e\text{-}assn\ f\text{-}assn\ g\text{-}assn\ h\text{-}assn\ i\text{-}assn\ j\text{-}assn\ k\text{-}assn\ l\text{-}assn\ m\text{-}assn\ n\text{-}assn\ o\text{-}assn\ p\text{-}assn\ S\ T =$
$(case\ (S,\ T)\ of$
$(Tuple16\ M\ N\ D\ i\ W\ ivmtf\ icount\ ccach\ lbd\ outl\ heur\ stats\ aivdom\ clss\ opts\ arena,$
$Tuple16\ M'\ N'\ D'\ i'\ W'\ ivmtf'\ icount'\ ccach'\ lbd'\ outl'\ heur'\ stats'\ aivdom'\ clss'\ opts'\ arena')$
$\Rightarrow$
$(a\text{-}assn\ M\ (M') \wedge* b\text{-}assn'\ N\ (N') \wedge* c\text{-}assn\ D\ (D') \wedge* d\text{-}assn\ i\ (i') \wedge*$
$e\text{-}assn\ W\ (W') \wedge* f\text{-}assn\ ivmtf\ (ivmtf') \wedge* g\text{-}assn\ icount\ (icount') \wedge* h\text{-}assn\ ccach\ (ccach') \wedge*$
$i\text{-}assn\ lbd\ (lbd') \wedge* j\text{-}assn\ outl\ (outl') \wedge* k\text{-}assn\ heur\ (heur') \wedge* l\text{-}assn\ stats\ (stats') \wedge*$
$m\text{-}assn\ aivdom\ (aivdom') \wedge* n\text{-}assn\ clss\ (clss') \wedge* o\text{-}assn\ opts\ (opts') \wedge* p\text{-}assn\ arena\ (arena')))$
$\rangle$

locale *tuple16-ops* =
fixes
$a\text{-}assn :: \langle 'a \Rightarrow 'xa :: llvm\text{-}rep \Rightarrow assn\rangle$ **and**
$b\text{-}assn :: \langle 'b \Rightarrow 'xb :: llvm\text{-}rep \Rightarrow assn\rangle$ **and**
$c\text{-}assn :: \langle 'c \Rightarrow 'xc :: llvm\text{-}rep \Rightarrow assn\rangle$ **and**
$d\text{-}assn :: \langle 'd \Rightarrow 'xd :: llvm\text{-}rep \Rightarrow assn\rangle$ **and**
$e\text{-}assn :: \langle 'e \Rightarrow 'xe :: llvm\text{-}rep \Rightarrow assn\rangle$ **and**
$f\text{-}assn :: \langle 'f \Rightarrow 'xf :: llvm\text{-}rep \Rightarrow assn\rangle$ **and**
$g\text{-}assn :: \langle 'g \Rightarrow 'xg :: llvm\text{-}rep \Rightarrow assn\rangle$ **and**
$h\text{-}assn :: \langle 'h \Rightarrow 'xh :: llvm\text{-}rep \Rightarrow assn\rangle$ **and**
$i\text{-}assn :: \langle 'i \Rightarrow 'xi :: llvm\text{-}rep \Rightarrow assn\rangle$ **and**
$j\text{-}assn :: \langle 'j \Rightarrow 'xj :: llvm\text{-}rep \Rightarrow assn\rangle$ **and**
$k\text{-}assn :: \langle 'k \Rightarrow 'xk :: llvm\text{-}rep \Rightarrow assn\rangle$ **and**
$l\text{-}assn :: \langle 'l \Rightarrow 'xl :: llvm\text{-}rep \Rightarrow assn\rangle$ **and**
$m\text{-}assn :: \langle 'm \Rightarrow 'xm :: llvm\text{-}rep \Rightarrow assn\rangle$ **and**
$n\text{-}assn :: \langle 'n \Rightarrow 'xn :: llvm\text{-}rep \Rightarrow assn\rangle$ **and**
$o\text{-}assn :: \langle 'o \Rightarrow 'xo :: llvm\text{-}rep \Rightarrow assn\rangle$ **and**
$p\text{-}assn :: \langle 'p \Rightarrow 'xp :: llvm\text{-}rep \Rightarrow assn\rangle$ **and**
$a\text{-}default :: 'a$ **and**
$a :: \langle 'xa\ llM\rangle$ **and**
$b\text{-}default :: 'b$ **and**
$b :: \langle 'xb\ llM\rangle$ **and**
$c\text{-}default :: 'c$ **and**
$c :: \langle 'xc\ llM\rangle$ **and**
$d\text{-}default :: 'd$ **and**
$d :: \langle 'xd\ llM\rangle$ **and**
$e\text{-}default :: 'e$ **and**
$e :: \langle 'xe\ llM\rangle$ **and**
$f\text{-}default :: 'f$ **and**
$f :: \langle 'xf\ llM\rangle$ **and**
$g\text{-}default :: 'g$ **and**
$g :: \langle 'xg\ llM\rangle$ **and**
$h\text{-}default :: 'h$ **and**
$h :: \langle 'xh\ llM\rangle$ **and**
$i\text{-}default :: 'i$ **and**
$i :: \langle 'xi\ llM\rangle$ **and**
$j\text{-}default :: 'j$ **and**
$j :: \langle 'xj\ llM\rangle$ **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'*)
⟨*proof*⟩
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*›
⟨*proof*⟩

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*$(λ2*a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16*. *f′ a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16*)$*p′*)› (**is** ‹*?G* Γ›)
⟨*proof*⟩
apply1 (*rule hn-refine-cons-pre*[*OF FR*])
apply1 (*cases p*; *cases p′*; *simp add*: *tuple16-assn-conv*[*THEN tuple16-assn-ctxt*])
⟨*proof*⟩
applyS (*simp add*: *hn-ctxt-def*)
applyS *simp* ⟨*proof*⟩

lemma *case-tuple16-arity*[*sepref-monadify-arity*]:
‹*case-tuple16* ≡ λ2*fp p*. *SP case-tuple16*$(λ2*a b*. *fp*$*a*$*b*)$*p*›
⟨*proof*⟩

lemma *case-tuple16-comb*[*sepref-monadify-comb*]:
‹⋀*fp p*. *case-tuple16*$*fp*$*p* ≡ *Refine-Basic.bind*$(*EVAL*$*p*)$(λ2*p*. (*SP case-tuple16*$*fp*$*p*))›
⟨*proof*⟩

lemma *case-tuple16-plain-comb*[*sepref-monadify-comb*]:

EVAL$(case-tuple16$(λ2a1 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$p)$(λ2p. case-tuple16$(λ2a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16. EVAL$(fp a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16))$p)
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*))
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *RETURN-case-tuple16-inverse*: ‹*RETURN*
(*let* - = *M*
in ff) =
(*do* {- ← *mop-free M*;
RETURN (*ff*)})›

⟨*proof*⟩

sepref-def *update-a-code*
is ‹*uncurry* (*RETURN oo update-a*)›
:: ‹$a\text{-}assn^d *_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-b-code*
is ‹*uncurry* (*RETURN oo update-b*)›
:: ‹$b\text{-}assn^d *_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-c-code*
is ‹*uncurry* (*RETURN oo update-c*)›
:: ‹$c\text{-}assn^d *_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-d-code*
is ‹*uncurry* (*RETURN oo update-d*)›
:: ‹$d\text{-}assn^d *_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-e-code*
is ‹*uncurry* (*RETURN oo update-e*)›
:: ‹$e\text{-}assn^d *_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-f-code*
is ‹*uncurry* (*RETURN oo update-f*)›
:: ‹$f\text{-}assn^d *_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-g-code*
is ‹*uncurry* (*RETURN oo update-g*)›
:: ‹$g\text{-}assn^d *_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-h-code*
is ‹*uncurry* (*RETURN oo update-h*)›
:: ‹$h\text{-}assn^d *_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-i-code*
is ‹*uncurry* (*RETURN oo update-i*)›
:: ‹$i\text{-}assn^d *_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-j-code*
is ‹*uncurry* (*RETURN oo update-j*)›
:: ‹$j\text{-}assn^d *_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-k-code*
is ‹*uncurry* (*RETURN oo update-k*)›
:: ‹$k\text{-}assn^d *_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-l-code*
  is ‹*uncurry* (*RETURN oo update-l*)›
  :: ‹*l-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
  ⟨*proof*⟩

sepref-def *update-m-code*
  is ‹*uncurry* (*RETURN oo update-m*)›
  :: ‹*m-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
  ⟨*proof*⟩

sepref-def *update-n-code*
  is ‹*uncurry* (*RETURN oo update-n*)›
  :: ‹*n-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
  ⟨*proof*⟩

sepref-def *update-o-code*
  is ‹*uncurry* (*RETURN oo update-o*)›
  :: ‹*o-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
  ⟨*proof*⟩

sepref-def *update-p-code*
  is ‹*uncurry* (*RETURN oo update-p*)›
  :: ‹*p-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
  ⟨*proof*⟩

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*))›
  ⟨*proof*⟩

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*›

⟨*proof*⟩

sepref-definition *remove-b-code*
is ‹*RETURN o remove-b*›
:: ‹ *isasat-assn*^{d} →$_a$ *b-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-c-code*
is ‹*RETURN o remove-c*›
:: ‹ *isasat-assn*^{d} →$_a$ *c-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-d-code*
is ‹*RETURN o remove-d*›
:: ‹ *isasat-assn*^{d} →$_a$ *d-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-e-code*
is ‹*RETURN o remove-e*›
:: ‹ *isasat-assn*^{d} →$_a$ *e-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-f-code*
is ‹*RETURN o remove-f*›
:: ‹ *isasat-assn*^{d} →$_a$ *f-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-g-code*
is ‹*RETURN o remove-g*›
:: ‹ *isasat-assn*^{d} →$_a$ *g-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-h-code*
is ‹*RETURN o remove-h*›
:: ‹ *isasat-assn*^{d} →$_a$ *h-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-i-code*
is ‹*RETURN o remove-i*›
:: ‹ *isasat-assn*^{d} →$_a$ *i-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-j-code*
is ‹*RETURN o remove-j*›
:: ‹ *isasat-assn*^{d} →$_a$ *j-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-k-code*
is ‹*RETURN o remove-k*›
:: ‹ *isasat-assn*^{d} →$_a$ *k-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-l-code*
is ‹*RETURN o remove-l*›
:: ‹ *isasat-assn*^{d} →$_a$ *l-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-m-code*
is ‹*RETURN o remove-m*›
:: ‹ *isasat-assn*^{d} →$_a$ *m-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-n-code*
is ‹*RETURN o remove-n*›
:: ‹ *isasat-assn*^{d} →$_a$ *n-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-o-code*
is ‹*RETURN o remove-o*›
:: ‹ *isasat-assn*^{d} →$_a$ *o-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-p-code*
is ‹*RETURN o remove-p*›
:: ‹ *isasat-assn*^{d} →$_a$ *p-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

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*)
}›
⟨*proof*⟩

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*)
}›
⟨*proof*⟩

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) (λ*S*. *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*
})›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
  ⟨*proof*⟩

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*: ‹$(\exists x.\ (P\ x \wedge* \uparrow (b = x))\ s) \longleftrightarrow P\ b\ s$›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *Constructor-assn-destroyed*:
‹(*Mreturn o id*, *RETURN o Constructor*) ∈ (*hr-comp R S*)^d →$_a$ *code-hider-assn R S*›
⟨*proof*⟩

lemma *get-content-destroyed*:
‹(*Mreturn o id*, *RETURN o get-content*) ∈ (*code-hider-assn R S*)^d →$_a$ *hr-comp R S*›
⟨*proof*⟩

lemma *get-content-hnr*[*sepref-fr-rules*]:
‹(*id*, *get-content*) ∈ ⟨*S*⟩*code-hider-rel* →$_f$ *S*›
⟨*proof*⟩

lemma *Constructor-hnr*[*sepref-fr-rules*]:
‹(*id*, *Constructor*) ∈ *S* →$_f$ ⟨*S*⟩*code-hider-rel*›
⟨*proof*⟩

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* ×$_a$ *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* ×$_a$ *word64-assn*›

definition *empty-search-stats* **where**
‹*empty-search-stats* = (*0*,*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*›
⟨*proof*⟩

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} →$_a$ *binary-stats-assn*›
⟨*proof*⟩

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} →$_a$ *subsumption-stats-assn*›
⟨*proof*⟩

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} →$_a$ *pure-lits-stats-assn*›
⟨*proof*⟩

definition *lbd-size-limit-assn* **where**
‹*lbd-size-limit-assn* = *uint32-nat-assn* ×$_a$ *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} →$_a$ *lbd-size-limit-assn*›
⟨*proof*⟩

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} →$_a$ *ema-assn*›
⟨*proof*⟩

lemma *stats-bottom*:
‹(*uncurry0* (*return*$_M$ *0*), *uncurry0* (*RETURN 0*)) ∈ *unit-assn*^{k} →$_a$ *word64-assn*›
‹(*uncurry0* (*return*$_M$ *0*), *uncurry0* (*RETURN 0*)) ∈ *unit-assn*^{k} →$_a$ *word32-assn*›
⟨*proof*⟩

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} →$_a$ *rephase-stats-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *free-search-stats-assn*
is ‹*mop-free*›
:: ‹*search-stats-assn*^{d} →$_a$ *unit-assn*›

⟨*proof*⟩

sepref-def *free-binary-stats-assn*
is ‹*mop-free*›
:: ‹*binary-stats-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-def *free-subsumption-stats-assn*
is ‹*mop-free*›
:: ‹*subsumption-stats-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-def *free-pure-lits-stats-assn*
is ‹*mop-free*›
:: ‹*pure-lits-stats-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-def *free-ema-assn*
is ‹*mop-free*›
:: ‹*ema-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-def *free-word64-assn*
is ‹*mop-free*›
:: ‹*word64-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-def *free-word32-assn*
is ‹*mop-free*›
:: ‹*word32-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-def *free-lbd-size-limit-assn*
is ‹*mop-free*›
:: ‹*lbd-size-limit-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-def *free-rephase-stats-assn*
is ‹*mop-free*›
:: ‹*rephase-stats-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

lemma *mop-free-hnr'*: ‹(*f*, *mop-free*) ∈ *R*^{d} →$_a$ *unit-assn* ⟹ *MK-FREE R f*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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)›
⟨proof⟩

lemma *pure-keep-detroy*: $(pure\ R)^k = (pure\ R)^d$
⟨proof⟩
lemma *is-pure R* $\Longrightarrow R^k = R^d$
⟨proof⟩

lemma *isasat-stats-assn-pure-keep*:
‹$isasat\text{-}stats\text{-}assn^d = isasat\text{-}stats\text{-}assn^k$›
⟨proof⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *id-unat*[*sepref-fr-rules*]:
‹(*Mreturn o id*, *RETURN o unat*) ∈ *word32-assn*^{k} →$_a$ *uint32-nat-assn*›
⟨*proof*⟩

lemma [*sepref-fr-rules*]:
‹*CONSTRAINT is-pure B* ⟹ (*Mreturn o* (λ(*a*,*b*). *a*), *RETURN o fst*) ∈ (*A* ×$_a$ *B*)^d →$_a$ *A*›
⟨*proof*⟩

sepref-def *Search-Stats-conflicts-impl*
is ‹*RETURN o Search-Stats-conflicts*›
:: ‹*search-stats-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *Search-Stats-units-since-gcs-impl*
is ‹*RETURN o Search-Stats-units-since-gcs*›
:: ‹*search-stats-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *Search-Stats-fixed-impl*
is ‹*RETURN o Search-Stats-fixed*›
:: ‹*search-stats-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *add-lbd-stats-impl*
is ‹*uncurry* (*RETURN oo add-lbd*)›
:: ‹*word32-assn*^{k} *$_a$ *isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
⟨*proof*⟩

sepref-def *get-conflict-count-stats-impl*
is ‹(*RETURN o get-conflict-count*)›
:: ‹*isasat-stats-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

sepref-def *units-since-last-GC-stats-impl*
is ‹(*RETURN o units-since-last-GC*)›
:: ‹*isasat-stats-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

sepref-def *reset-units-since-last-GC-stats-impl*
is ‹(*RETURN o reset-units-since-last-GC*)›
:: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
⟨*proof*⟩

sepref-def *Search-Stats-incr-irred-impl*
is ‹*RETURN o Search-Stats-incr-irred*›
:: ‹*search-stats-assn*^{k} →$_a$ *search-stats-assn*›
⟨*proof*⟩

sepref-def *Search-Stats-decr-irred-impl*
is ‹*RETURN o Search-Stats-decr-irred*›
:: ‹*search-stats-assn*^{k} →$_a$ *search-stats-assn*›
⟨*proof*⟩

sepref-def *Search-Stats-incr-propagation-impl*

is ‹*RETURN o Search-Stats-incr-propagation*›
:: ‹*search-stats-assn*^{k} →$_a$ *search-stats-assn*›
⟨*proof*⟩

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} →$_a$ *search-stats-assn*›
⟨*proof*⟩

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} →$_a$ *search-stats-assn*›
⟨*proof*⟩

sepref-def *Search-Stats-no-conflict-until-impl*
is ‹*RETURN o Search-Stats-no-conflict-until*›
:: ‹*search-stats-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *Search-Stats-incr-conflicts-impl*
is ‹*RETURN o Search-Stats-incr-conflicts*›
:: ‹*search-stats-assn*^{k} →$_a$ *search-stats-assn*›
⟨*proof*⟩

sepref-def *Search-Stats-incr-decisions-impl*
is ‹*RETURN o Search-Stats-incr-decisions*›
:: ‹*search-stats-assn*^{k} →$_a$ *search-stats-assn*›
⟨*proof*⟩

sepref-def *Search-Stats-incr-restarts-impl*
is ‹*RETURN o Search-Stats-incr-restarts*›
:: ‹*search-stats-assn*^{k} →$_a$ *search-stats-assn*›
⟨*proof*⟩

sepref-def *Search-Stats-incr-reductions-impl*
is ‹*RETURN o Search-Stats-incr-reductions*›
:: ‹*search-stats-assn*^{k} →$_a$ *search-stats-assn*›
⟨*proof*⟩

sepref-def *Search-Stats-incr-fixed-impl*
is ‹*RETURN o Search-Stats-incr-fixed*›
:: ‹*search-stats-assn*^{k} →$_a$ *search-stats-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *Search-Stats-incr-gcs-impl*
is ‹*RETURN o Search-Stats-incr-gcs*›
:: ‹*search-stats-assn*^{k} →$_a$ *search-stats-assn*›
⟨*proof*⟩

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} $\rightarrow_a$ *search-stats-assn*›
⟨*proof*⟩

sepref-def *Search-Stats-incr-units-since-gc-impl*
is ‹*RETURN o Search-Stats-incr-units-since-gc*›
:: ‹*search-stats-assn*^{k} $\rightarrow_a$ *search-stats-assn*›
⟨*proof*⟩

sepref-def *Pure-lits-Stats-incr-rounds-impl*
is ‹*RETURN o Pure-lits-Stats-incr-rounds*›
:: ‹*pure-lits-stats-assn*^{k} $\rightarrow_a$ *pure-lits-stats-assn*›
⟨*proof*⟩

sepref-def *Pure-lits-Stats-incr-removed-impl*
is ‹*RETURN o Pure-lits-Stats-incr-removed*›
:: ‹*pure-lits-stats-assn*^{k} $\rightarrow_a$ *pure-lits-stats-assn*›
⟨*proof*⟩

sepref-def *Binary-Stats-incr-units-impl*
is ‹*RETURN o Binary-Stats-incr-units*›
:: ‹*binary-stats-assn*^{k} $\rightarrow_a$ *binary-stats-assn*›
⟨*proof*⟩

sepref-def *Binary-Stats-incr-removed-def*
is ‹*RETURN o Binary-Stats-incr-removed*›
:: ‹*binary-stats-assn*^{k} $\rightarrow_a$ *binary-stats-assn*›
⟨*proof*⟩

sepref-def *Search-Stats-restarts-impl*
is ‹*RETURN o Search-Stats-restarts*›
:: ‹*search-stats-assn*^{k} $\rightarrow_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *Search-Stats-reductions-impl*
is ‹*RETURN o Search-Stats-reductions*›
:: ‹*search-stats-assn*^{k} $\rightarrow_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *Search-Stats-irred-impl*
is ‹*RETURN o Search-Stats-irred*›
:: ‹*search-stats-assn*^{k} $\rightarrow_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *Search-Stats-propagations-impl*
is ‹*RETURN o Search-Stats-propagations*›
:: ‹*search-stats-assn*^{k} $\rightarrow_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *Search-Stats-gcs-impl*
is ‹*RETURN o Search-Stats-gcs*›
:: ‹*search-stats-assn*^{k} $\rightarrow_a$ *word64-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *Binary-stats-rounds-impl*
is ‹*RETURN o Binary-Stats-rounds*›
:: ‹*binary-stats-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *Binary-stats-units-impl*
is ‹*RETURN o Binary-Stats-units*›
:: ‹*binary-stats-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *Binary-stats-removed-impl*
is ‹*RETURN o Binary-Stats-removed*›
:: ‹*binary-stats-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *Pure-Lits-Stats-removed-impl*
is ‹*RETURN o Pure-Lits-Stats-removed*›
:: ‹*pure-lits-stats-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *Pure-Lits-Stats-removed-rounds-impl*
is ‹*RETURN o Pure-Lits-Stats-rounds*›
:: ‹*pure-lits-stats-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *LSize-Stats-lbd-impl*
is ‹*RETURN o LSize-Stats-lbd*›
:: ‹*lbd-size-limit-assn*^{k} →$_a$ *uint32-nat-assn*›
⟨*proof*⟩

sepref-def *LSize-Stats-size-impl*
is ‹*RETURN o LSize-Stats-size*›
:: ‹*lbd-size-limit-assn*^{k} →$_a$ *sint64-nat-assn*›
⟨*proof*⟩

sepref-def *LSize-Stats-impl*
is ‹*uncurry* (*RETURN oo LSize-Stats*)›
:: ‹*uint32-nat-assn*^{k} *$_a$ *sint64-nat-assn*^{k} →$_a$ *lbd-size-limit-assn*›
⟨*proof*⟩

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} →$_a$ *word64-assn*›

⟨*proof*⟩

sepref-def *stats-irred-impl*
  is ‹*RETURN o stats-irred*› :: ‹*isasat-stats-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *stats-binary-units-impl*
  is ‹*RETURN o stats-binary-units*› :: ‹*isasat-stats-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *stats-binary-removed-impl*
  is ‹*RETURN o stats-binary-removed*› :: ‹*isasat-stats-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩
sepref-def *stats-binary-rounds-impl*
  is ‹*RETURN o stats-binary-rounds*› :: ‹*isasat-stats-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *stats-pure-lits-rounds-impl*
  is ‹*RETURN o stats-pure-lits-rounds*› :: ‹*isasat-stats-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *stats-pure-lits-removed-impl*
  is ‹*RETURN o stats-pure-lits-removed*› :: ‹*isasat-stats-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *units-since-beginning-stats-impl*
  is ‹(*RETURN o units-since-beginning*)›
  :: ‹*isasat-stats-assn*^{k} →$_a$ *word-assn*›
  ⟨*proof*⟩

sepref-def *incr-irred-clss-stats-impl*
  is ‹*RETURN o incr-irred-clss*›
  :: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
  ⟨*proof*⟩

sepref-def *decr-irred-clss-stats-impl*
  is ‹*RETURN o decr-irred-clss*›
  :: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
  ⟨*proof*⟩

sepref-def *incr-propagation-stats-impl*
  is ‹*RETURN o incr-propagation*›
  :: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

sepref-def *no-conflict-until-stats-impl*

is ‹*RETURN o no-conflict-until*›
:: ‹*isasat-stats-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *incr-conflict-stats-impl*
is ‹*RETURN o incr-conflict*›
:: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
⟨*proof*⟩

sepref-def *incr-decision-stats-impl*
is ‹*RETURN o incr-decision*›
:: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
⟨*proof*⟩

sepref-def *incr-restart-stats-impl*
is ‹*RETURN o incr-restart*›
:: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
⟨*proof*⟩

sepref-def *incr-reduction-stats-impl*
is ‹*RETURN o incr-reduction*›
:: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
⟨*proof*⟩

sepref-def *incr-uset-stats-impl*
is ‹*RETURN o incr-uset*›
:: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *incr-GC-stats-impl*
is ‹*RETURN o incr-GC*›
:: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
⟨*proof*⟩

sepref-def *stats-conflicts-impl*
is ‹*RETURN o stats-conflicts*›
:: ‹*isasat-stats-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

sepref-def *incr-units-since-last-GC-impl*
is ‹*RETURN o incr-units-since-last-GC*›
:: ‹*isasat-stats-assn*^{d} →$_a$ *isasat-stats-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *incr-purelit-rounds-impl*
is ‹*RETURN o incr-purelit-rounds*›

:: ‹*isasat-stats-assn*^{d} $\rightarrow_a$ *isasat-stats-assn*›
⟨*proof*⟩

sepref-def *incr-purelit-elim-stats-impl*
is ‹*RETURN o incr-purelit-elim*›
:: ‹*isasat-stats-assn*^{d} $\rightarrow_a$ *isasat-stats-assn*›
⟨*proof*⟩

sepref-def *incr-binary-red-removed-impl*
is ‹*RETURN o incr-binary-red-removed*›
:: ‹*isasat-stats-assn*^{d} $\rightarrow_a$ *isasat-stats-assn*›
⟨*proof*⟩

sepref-def *incr-binary-unit-derived-impl*
is ‹*RETURN o incr-binary-unit-derived*›
:: ‹*isasat-stats-assn*^{d} $\rightarrow_a$ *isasat-stats-assn*›
⟨*proof*⟩

sepref-def *get-reduction-count-impl*
is ‹*RETURN o get-reduction-count*›
:: ‹*isasat-stats-assn*^{k} $\rightarrow_a$ *word-assn*›
⟨*proof*⟩

sepref-def *get-restart-count-impl*
is ‹*RETURN o get-restart-count*›
:: ‹*isasat-stats-assn*^{k} $\rightarrow_a$ *word-assn*›
⟨*proof*⟩

sepref-def *get-irredundant-count-impl*
is ‹*RETURN o irredundant-clss*›
:: ‹*isasat-stats-assn*^{k} $\rightarrow_a$ *word-assn*›
⟨*proof*⟩

sepref-def *stats-propagations-impl*
is ‹*RETURN o stats-propagations*›
:: ‹*isasat-stats-assn*^{k} $\rightarrow_a$ *word-assn*›
⟨*proof*⟩

sepref-def *stats-restarts-impl*
is ‹*RETURN o stats-restarts*›
:: ‹*isasat-stats-assn*^{k} $\rightarrow_a$ *word-assn*›
⟨*proof*⟩

sepref-def *stats-reductions-impl*
is ‹*RETURN o stats-reductions*›
:: ‹*isasat-stats-assn*^{k} $\rightarrow_a$ *word-assn*›
⟨*proof*⟩

sepref-def *stats-fixed-impl*
is ‹*RETURN o stats-fixed*›
:: ‹*isasat-stats-assn*^{k} $\rightarrow_a$ *word-assn*›
⟨*proof*⟩

sepref-def *stats-gcs-impl*
is ‹*RETURN o stats-gcs*›

:: ‹*isasat-stats-assn*^{k} $\rightarrow_a$ *word-assn*›
⟨*proof*⟩

sepref-def *stats-avg-lbd-mpl*
is ‹*RETURN o stats-avg-lbd*›
:: ‹*isasat-stats-assn*^{k} $\rightarrow_a$ *ema-assn*›
⟨*proof*⟩

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} $\rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

sepref-def *stats-size-limit-impl*
is ‹*RETURN o stats-size-limit*›
:: ‹*isasat-stats-assn*^{k} $\rightarrow_a$ *sint64-nat-assn*›
⟨*proof*⟩

sepref-def *Subsumption-Stats-incr-strengthening-impl*
is ‹*RETURN o Subsumption-Stats-incr-strengthening*›
:: ‹*subsumption-stats-assn*^{d} $\rightarrow_a$ *subsumption-stats-assn*›
⟨*proof*⟩

sepref-def *incr-forward-strengethening-impl*
is ‹*RETURN o incr-forward-strengethening*›
:: ‹*isasat-stats-assn*^{d} $\rightarrow_a$ *isasat-stats-assn*›
⟨*proof*⟩

sepref-def *Subsumption-Stats-incr-subsumed-impl*
is ‹*RETURN o Subsumption-Stats-incr-subsumed*›
:: ‹*subsumption-stats-assn*^{d} $\rightarrow_a$ *subsumption-stats-assn*›
⟨*proof*⟩

sepref-def *Subsumption-Stats-incr-tried-impl*
is ‹*RETURN o Subsumption-Stats-incr-tried*›
:: ‹*subsumption-stats-assn*^{d} $\rightarrow_a$ *subsumption-stats-assn*›
⟨*proof*⟩

sepref-def *Subsumption-Stats-incr-rounds-impl*
is ‹*RETURN o Subsumption-Stats-incr-rounds*›
:: ‹*subsumption-stats-assn*^{d} $\rightarrow_a$ *subsumption-stats-assn*›
⟨*proof*⟩

sepref-def *incr-forward-subsumed-impl*
is ‹*RETURN o incr-forward-subsumed*›
:: ‹*isasat-stats-assn*^{d} $\rightarrow_a$ *isasat-stats-assn*›
⟨*proof*⟩

sepref-def *incr-forward-rounds-impl*
is ‹*RETURN o incr-forward-rounds*›
:: ‹*isasat-stats-assn*^{d} $\rightarrow_a$ *isasat-stats-assn*›
⟨*proof*⟩

sepref-def *incr-forward-tried-impl*
is ‹*RETURN o incr-forward-tried*›

:: ‹*isasat-stats-assn*^{d} $\rightarrow_a$ *isasat-stats-assn*›
⟨*proof*⟩

sepref-def *Subsumption-Stats-rounds-impl*
is ‹*RETURN o Subsumption-Stats-rounds*›
:: ‹*subsumption-stats-assn*^{k} $\rightarrow_a$ *word64-assn*›
⟨*proof*⟩
sepref-def *Subsumption-Stats-strengthened-impl*
is ‹*RETURN o Subsumption-Stats-strengthened*›
:: ‹*subsumption-stats-assn*^{k} $\rightarrow_a$ *word64-assn*›
⟨*proof*⟩
sepref-def *Subsumption-Stats-subsumed-impl*
is ‹*RETURN o Subsumption-Stats-subsumed*›
:: ‹*subsumption-stats-assn*^{k} $\rightarrow_a$ *word64-assn*›
⟨*proof*⟩
sepref-def *Subsumption-Stats-tried-impl*
is ‹*RETURN o Subsumption-Stats-tried*›
:: ‹*subsumption-stats-assn*^{k} $\rightarrow_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *stats-forward-rounds-impl*
is ‹*RETURN o stats-forward-rounds*›
:: ‹*isasat-stats-assn*^{k} $\rightarrow_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *Rephase-Stats-incr-total-impl*
is ‹*RETURN o Rephase-Stats-incr-total*›
:: ‹*rephase-stats-assn*^{d} $\rightarrow_a$ *rephase-stats-assn*›
⟨*proof*⟩

sepref-def *Rephase-Stats-total-impl*
is ‹*RETURN o Rephase-Stats-total*›
:: ‹*rephase-stats-assn*^{d} $\rightarrow_a$ *word64-assn*›
⟨*proof*⟩

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} $\rightarrow_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *stats-forward-tried-impl*
is ‹*RETURN o stats-forward-tried*›
:: ‹*isasat-stats-assn*^{d} $\rightarrow_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *stats-forward-strengthened-impl*
is ‹*RETURN o stats-forward-strengthened*›
:: ‹*isasat-stats-assn*^{d} $\rightarrow_a$ *word64-assn*›
⟨*proof*⟩

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*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

sepref-def *empty-stats-clss-impl*
  is ‹(*RETURN o empty-stats-clss*)›
  :: ‹*word64-assn*^{k} →$_a$ *isasat-stats-assn*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

sepref-def *incr-rephase-total-impl*
  is ‹*RETURN o incr-rephase-total*›
  :: ‹*isasat-stats-assn*^{k} →$_a$ *isasat-stats-assn*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

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*
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *STABLE-MODE-impl*
is ‹*uncurry0* (*RETURN STABLE-MODE*)›
:: ‹*unit-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *clss-size-lcount-fast-code*
is ‹*RETURN o clss-size-lcount*›
:: ‹*lcount-assn*^{k} →$_a$ *uint64-nat-assn*›
⟨*proof*⟩

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*))›
⟨*proof*⟩

sepref-def *clss-size-resetUS-fast-code*
is ‹*RETURN o clss-size-resetUS*›
:: ‹*lcount-assn*^{d} →$_a$ *lcount-assn*›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *clss-size-resetU0-alt-def*:
‹*RETURN o clss-size-resetU0* =
(λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *lcountU0*). *RETURN* (*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *0*))›
⟨*proof*⟩

sepref-def *clss-size-resetU0-fast-code*
is ‹*RETURN o clss-size-resetU0*›
:: ‹*lcount-assn*^{d} →$_a$ *lcount-assn*›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *clss-size-resetUE-alt-def*:
‹*RETURN o clss-size-resetUE* =
(λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *lcountU0*). *RETURN* (*lcount*, *0*, *lcountUEk*, *lcountUS*, *lcountU0*))›
⟨*proof*⟩

sepref-def *clss-size-resetUE-fast-code*
is ‹*RETURN o clss-size-resetUE*›
:: ‹*lcount-assn*^{d} →$_a$ *lcount-assn*›
⟨*proof*⟩

lemma *clss-size-incr-lcountUE-alt-def*:
‹*RETURN o clss-size-incr-lcountUE* =
(λ(*lcount*, *lcountUE*, *lcountUS*). *RETURN* (*lcount*, *lcountUE* + *1*, *lcountUS*))›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *clss-size-incr-lcountUEk-alt-def*:
‹*RETURN o clss-size-incr-lcountUEk* =
(λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*). *RETURN* (*lcount*, *lcountUE*, *lcountUEk* + *1*, *lcountUS*))›
⟨*proof*⟩

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*›
⟨*proof*⟩

schematic-goal *mk-free-lookup-clause-rel-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE lcount-assn ?fr*›
⟨*proof*⟩

lemma *clss-size-lcountUE-alt-def*:
‹*RETURN o clss-size-lcountUE* = (λ(*lcount*, *lcountUE*, *lcountUS*). *RETURN lcountUE*)›
⟨*proof*⟩

sepref-def *clss-size-lcountUE-fast-code*
is ‹*RETURN o clss-size-lcountUE*›
:: ‹*lcount-assn*^{k} →$_a$ *uint64-nat-assn*›
⟨*proof*⟩

lemma *clss-size-lcountUS-alt-def*:
‹*RETURN o clss-size-lcountUS* = (λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *lcountU0*). *RETURN lcountUS*)›
⟨*proof*⟩

sepref-def *clss-size-lcountUSt-fast-code*
is ‹*RETURN o clss-size-lcountUS*›
:: ‹*lcount-assn*^{k} →$_a$ *uint64-nat-assn*›
⟨*proof*⟩

lemma *clss-size-lcountU0-alt-def*:
‹*RETURN o clss-size-lcountU0* = (λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *lcountU0*). *RETURN lcountU0*)›
⟨*proof*⟩

sepref-def *clss-size-lcountU0-fast-code*
is ‹*RETURN o clss-size-lcountU0*›
:: ‹*lcount-assn*^{k} →$_a$ *uint64-nat-assn*›
⟨*proof*⟩

lemma *clss-size-incr-allcount-alt-def*:
‹*RETURN o clss-size-allcount* =
(λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*, *lcountU0*). *RETURN* (*lcount* + *lcountUE* + *lcountUEk* + *lcountUS* + *lcountU0*))›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *clss-size-decr-lcount-alt-def*:
‹*RETURN o clss-size-decr-lcount* =
(λ(*lcount*, *lcountUE*, *lcountUS*). *RETURN* (*lcount* − *1*, *lcountUE*, *lcountUS*))›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *emag-get-value-alt-def*:
‹*ema-get-value* = (λ(*a*, *b*, *c*, *d*). *a*)›
⟨*proof*⟩

sepref-def *ema-get-value-impl*
is ‹*RETURN o ema-get-value*›
:: ‹*ema-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

lemma *emag-extract-value-alt-def*:
‹*ema-extract-value* = (λ(*a*, *b*, *c*, *d*). *a* >> *EMA-FIXPOINT-SIZE*)›
⟨*proof*⟩

sepref-def *ema-extract-value-impl*
is ‹*RETURN o ema-extract-value*›
:: ‹*ema-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

sepref-def *schedule-next-pure-lits-info-impl*
is ‹*RETURN o schedule-next-pure-lits-info*›
:: ‹*schedule-info-assn*^{k} →$_a$ *schedule-info-assn*›
⟨*proof*⟩

sepref-def *next-pure-lits-schedule-info-impl*
is ‹*RETURN o next-pure-lits-schedule-info*›
:: ‹*schedule-info-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *next-reduce-schedule-info-impl*
is ‹*RETURN o next-reduce-schedule-info*›
:: ‹*schedule-info-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *next-subsume-schedule-info-impl*
is ‹*RETURN o next-subsume-schedule-info*›
:: ‹*schedule-info-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

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 $\times_a$
restart-info-assn $\times_a$
word64-assn $\times_a$ *phase-heur-assn* $\times_a$ *reluctant-assn* $\times_a$ *bool1-assn* $\times_a$ *phase-saver-assn* $\times_a$
schedule-info-assn $\times_a$ *ema-assn* $\times_a$ *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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*))›
⟨*proof*⟩

sepref-def *set-zero-wasted-stats-impl*
is ‹*RETURN o set-zero-wasted-stats*›
:: ‹*heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
⟨*proof*⟩

sepref-def *heuristic-reluctant-tick-stats-impl*
is ‹*RETURN o heuristic-reluctant-tick-stats*›
:: ‹*heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
⟨*proof*⟩

sepref-def *heuristic-reluctant-enable-stats-impl*
is ‹*RETURN o heuristic-reluctant-enable-stats*›
:: ‹*heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
⟨*proof*⟩

sepref-def *heuristic-reluctant-disable-stats-impl*
is ‹*RETURN o heuristic-reluctant-disable-stats*›
:: ‹*heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
⟨*proof*⟩

sepref-def *heuristic-reluctant-triggered-stats-impl*
is ‹*RETURN o heuristic-reluctant-triggered-stats*›
:: ‹*heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn* ×$_a$ *bool1-assn*›

⟨*proof*⟩

sepref-def *heuristic-reluctant-triggered2-stats-impl*
is ‹*RETURN o heuristic-reluctant-triggered2-stats*›
:: ‹*heuristic-int-assn*^{k} →$_a$ *bool1-assn*›
⟨*proof*⟩

sepref-def *heuristic-reluctant-untrigger-stats-impl*
is ‹*RETURN o heuristic-reluctant-untrigger-stats*›
:: ‹*heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
⟨*proof*⟩

sepref-def *end-of-rephasing-phase-impl* [*llvm-inline*]
is ‹*RETURN o end-of-rephasing-phase*›
:: ‹*phase-heur-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *end-of-rephasing-phase-heur-stats-impl*
is ‹*RETURN o end-of-rephasing-phase-heur-stats*›
:: ‹*heuristic-int-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *is-fully-propagated-heur-stats-impl*
is ‹*RETURN o is-fully-propagated-heur-stats*›
:: ‹*heuristic-int-assn*^{k} →$_a$ *bool1-assn*›
⟨*proof*⟩

sepref-def *set-fully-propagated-heur-stats-impl*
is ‹*RETURN o set-fully-propagated-heur-stats*›
:: ‹*heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
⟨*proof*⟩

sepref-def *unset-fully-propagated-heur-stats-impl*
is ‹*RETURN o unset-fully-propagated-heur-stats*›
:: ‹*heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
⟨*proof*⟩

sepref-def *restart-info-restart-done-heur-stats-impl*
is ‹*RETURN o restart-info-restart-done-heur-stats*›
:: ‹*heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
⟨*proof*⟩

sepref-def *set-zero-wasted-impl*
is ‹*RETURN o set-zero-wasted-stats*›
:: ‹*heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
⟨*proof*⟩

lemma *wasted-of-stats-alt-def*:
‹*RETURN o wasted-of-stats* = (λ(*fast-ema*, *slow-ema*, *res-info*, *wasted*, *φ*). *RETURN wasted*)›
⟨*proof*⟩

sepref-def *wasted-of-stats-impl*
is ‹*RETURN o wasted-of-stats*›
:: ‹*heuristic-int-assn*^{k} →$_a$ *word64-assn*›

⟨*proof*⟩

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*)›
⟨*proof*⟩

sepref-def *slow-ema-of-stats-impl*
is ‹*RETURN o slow-ema-of-stats*›
:: ‹*heuristic-int-assn*^{k} →$_a$ *ema-assn*›
⟨*proof*⟩

sepref-def *fast-ema-of-stats-impl*
is ‹*RETURN o fast-ema-of-stats*›
:: ‹*heuristic-int-assn*^{k} →$_a$ *ema-assn*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

sepref-def *current-restart-phase-impl*
is ‹*RETURN o current-restart-phase-stats*›
:: ‹*heuristic-int-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

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))›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *swap-emas-stats-impl*
is ‹*RETURN o swap-emas-stats*›
:: ‹*heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
⟨*proof*⟩

sepref-def *current-rephasing-phase-stats-impl*
is ‹*RETURN o current-rephasing-phase-stats*›
:: ‹*heuristic-int-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *schedule-next-pure-lits-stats-impl*
is ‹*RETURN o schedule-next-pure-lits-stats*›
:: ‹*heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

sepref-def *next-reduce-schedule-info-stats-impl*
is ‹*RETURN o next-reduce-schedule-info-stats*›
:: ‹*heuristic-int-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

sepref-def *next-subsume-schedule-info-stats-impl*
is ‹*RETURN o next-subsume-schedule-info-stats*›
:: ‹*heuristic-int-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

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*›

⟨*proof*⟩

lemma *hn-id-pure*:
‹*CONSTRAINT is-pure A* ⟹ (*Mreturn*, *RETURN o id*) ∈ A^k →$_a$ *A*›
⟨*proof*⟩

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*)})›
⟨*proof*⟩

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} $\to_a$ *heuristic-int-assn*›
⟨*proof*⟩

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*)
}›
⟨*proof*⟩

sepref-def *mop-save-phase-heur-impl*
is ‹*uncurry2* (*mop-save-phase-heur*)›
:: ‹*atom-assn*^{k} $*_a$ *bool1-assn*^{k} $*_a$ *heuristic-assn*^{d} $\to_a$ *heuristic-assn*›
⟨*proof*⟩

schematic-goal *mk-free-heuristic-int-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE heuristic-int-assn ?fr*›
⟨*proof*⟩

schematic-goal *mk-free-heuristic-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE heuristic-assn ?fr*›
⟨*proof*⟩

lemma [*safe-constraint-rules*]: ‹*CONSTRAINT is-pure A* ⟹ *CONSTRAINT is-pure* (*code-hider-assn A B*)›
⟨*proof*⟩

lemma *clss-size-incr-lcount-alt-def*:
‹*RETURN o clss-size-incr-lcount* =
(λ(*lcount*, *lcountUE*, *lcountUS*). *RETURN* (*lcount* + *1*, *lcountUE*, *lcountUS*))›
⟨*proof*⟩

lemma *clss-size-lcountUEk-alt-def*:
‹*RETURN o clss-size-lcountUEk* = (λ(*lcount*, *lcountUE*, *lcountUEk*, *lcountUS*). *RETURN lcountUEk*)›
⟨*proof*⟩

sepref-def *clss-size-lcountUEk-fast-code*
is ‹*RETURN o clss-size-lcountUEk*›
:: ‹*lcount-assn*^{k} $\to_a$ *uint64-nat-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *end-of-restart-phase-impl*
is ‹*RETURN o end-of-restart-phase-stats*›
:: ‹*heuristic-int-assn*^{k} $\to_a$ *word-assn*›
⟨*proof*⟩

lemma *end-of-restart-phase-stats-end-of-restart-phase*:
‹(*end-of-restart-phase-stats*, *end-of-restart-phase*) ∈ *heur-rel* → *word-rel*›

⟨*proof*⟩

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*))›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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))›
⟨*proof*⟩

sepref-def *incr-restart-phase-stats-impl*
is ‹*RETURN o incr-restart-phase-stats*›
:: ‹*heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
⟨*proof*⟩

sepref-def *incr-restart-phase-impl*
is ‹*RETURN o incr-restart-phase*›
:: ‹*heuristic-assn*^{d} →$_a$ *heuristic-assn*›
⟨*proof*⟩

sepref-def *reset-added-heur-stats2-impl*
is *reset-added-heur-stats2*
:: ‹*heuristic-int-assn*^{d} →$_a$ *heuristic-int-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*))
})›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)
})›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *vmtf-ns-distinct*: ‹*vmtf-ns l m xs* ⟹ *distinct l*›
⟨*proof*⟩

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⟩

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›
⟨proof⟩

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)›
⟨proof⟩

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))›
⟨proof⟩

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))])›
⟨proof⟩

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›
⟨proof⟩

lemma vmtf-ns-last-next:
‹vmtf-ns (xs @ [x]) m ns ⟹ get-next (ns ! x) = None›
⟨proof⟩

lemma *vmtf-ns-hd-prev*:
‹*vmtf-ns* (*x* # *xs*) *m ns* ⟹ *get-prev* (*ns* ! *x*) = *None*›
⟨*proof*⟩

lemma *vmtf-ns-last-mid-get-next*:
‹*vmtf-ns* (*xs* @ [*x*, *y*] @ *zs*) *m ns* ⟹ *get-next* (*ns* ! *x*) = *Some y*›
⟨*proof*⟩

lemma *vmtf-ns-last-mid-get-next-option-hd*:
‹*vmtf-ns* (*xs* @ *x* # *zs*) *m ns* ⟹ *get-next* (*ns* ! *x*) = *option-hd zs*›
⟨*proof*⟩

lemma *vmtf-ns-last-mid-get-prev*:
assumes ‹*vmtf-ns* (*xs* @ [*x*, *y*] @ *zs*) *m ns*›
shows ‹*get-prev* (*ns* ! *y*) = *Some x*›
⟨*proof*⟩

lemma *vmtf-ns-last-mid-get-prev-option-last*:
‹*vmtf-ns* (*xs* @ *x* # *zs*) *m ns* ⟹ *get-prev* (*ns* ! *x*) = *option-last xs*›
⟨*proof*⟩

lemma *length-ns-vmtf-dequeue*[*simp*]: ‹*length* (*ns-vmtf-dequeue x ns*) = *length ns*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *stamp-ns-vmtf-dequeue*:
‹*axs* < *length zs* ⟹ *stamp* (*ns-vmtf-dequeue x zs* ! *axs*) = *stamp* (*zs* ! *axs*)›
⟨*proof*⟩

lemma *sorted-many-eq-append*: ‹*sorted* (*xs* @ [*x*, *y*]) ⟷ *sorted* (*xs* @ [*x*]) ∧ *x* ≤ *y*›
⟨*proof*⟩

lemma *vmtf-ns-stamp-sorted*:
assumes ‹*vmtf-ns l m ns*›
shows ‹*sorted* (*map* (λ*a*. *stamp* (*ns*!*a*)) (*rev l*)) ∧ (∀ *a* ∈ *set l*. *stamp* (*ns*!*a*) ≤ *m*)›
⟨*proof*⟩

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*⟩

lemma *vmtf-ns-hd-next*:
  ‹*vmtf-ns* (*x* # *a* # *list*) *m ns* ⟹ *get-next* (*ns* ! *x*) = *Some a*›
  ⟨*proof*⟩

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*⟩

lemma *vmtf-ns-stamp-distinct*:
  assumes ‹*vmtf-ns l m ns*›
  shows ‹*distinct* (*map* (λ*a*. *stamp* (*ns*!*a*)) *l*)›
  ⟨*proof*⟩

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*⟩

lemma *vmtf-ns-rescale*:
  assumes
    ‹*vmtf-ns l m xs*› **and**
    ‹*sorted* (*map* (λ*a*. *st* ! *a*) (*rev l*))› **and** ‹*distinct* (*map* (λ*a*. *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*›
  ⟨*proof*⟩

lemma *vmtf-ns-last-prev*:
  assumes *vmtf*: ‹*vmtf-ns* (*xs* @ [*x*]) *m ns*›
  shows ‹*get-prev* (*ns* ! *x*) = *option-last xs*›
⟨*proof*⟩

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*⟩

lemma *vmtf-consD*′:
assumes *vmtf*: ‹*x* ∈ *vmtf* $\mathcal{A}$ *M*›
shows ‹*x* ∈ *vmtf* $\mathcal{A}$ (*L* # *M*)›
⟨*proof*⟩

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* $\mathcal{A}$ *M*› **and**
L: ‹*L* ∈ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)› **and**
dequeue: ‹(*ns*′, *m*′, *fst-As*′, *lst-As*′, *next-search*′) =

vmtf-dequeue L (ns, m, fst-As, lst-As, next-search)› **and**
$\mathcal{A}_{in}$*-nempty*: ‹*isasat-input-nempty* $\mathcal{A}$›
shows ‹$\exists$ *xs' ys'. vmtf-ns (ys' @ xs') m' ns'* $\wedge$ *fst-As' = option-hd (ys' @ xs')*
$\wedge$ *lst-As' = option-last (ys' @ xs')*
$\wedge$ *next-search' = option-hd xs'*
$\wedge$ *next-search' = (if next-search = Some L then get-next (ns!L) else next-search)*
$\wedge$ *vmtf-*$\mathcal{L}_{all}$ $\mathcal{A}$ *M ((insert L (set xs'), set ys'))*
$\wedge$ *vmtf-ns-notin (ys' @ xs') m' ns'* $\wedge$
L $\notin$ *set (ys' @ xs')* $\wedge$ ($\forall$ *L*$\in$*set (ys' @ xs'). L* $\in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$))›
⟨*proof*⟩

lemma *vmtf-ns-get-prev-not-itself*:
‹*vmtf-ns xs m ns* $\Longrightarrow$ *L* $\in$ *set xs* $\Longrightarrow$ *L* < *length ns* $\Longrightarrow$ *get-prev (ns ! L)* $\neq$ *Some L*›
⟨*proof*⟩

lemma *vmtf-ns-get-next-not-itself*:
‹*vmtf-ns xs m ns* $\Longrightarrow$ *L* $\in$ *set xs* $\Longrightarrow$ *L* < *length ns* $\Longrightarrow$ *get-next (ns ! L)* $\neq$ *Some L*›
⟨*proof*⟩

lemma *abs-vmtf-ns-bump-vmtf-en-dequeue*:
fixes *M*
assumes
vmtf: ‹(*ns*, *m*, *fst-As*, *lst-As*, *next-search*) $\in$ *vmtf* $\mathcal{A}$ *M*› **and**
L: ‹*L* $\in$ *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*) $\in$ *vmtf* $\mathcal{A}$ *M*›
⟨*proof*⟩

lemma *abs-vmtf-ns-bump-vmtf-en-dequeue'*:
fixes *M*
assumes
vmtf: ‹(*vm*) $\in$ *vmtf* $\mathcal{A}$ *M*› **and**
L: ‹*L* $\in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)› **and**
nempty: ‹*isasat-input-nempty* $\mathcal{A}$›
shows ‹*vmtf-en-dequeue M L vm* $\in$ *vmtf* $\mathcal{A}$ *M*›
⟨*proof*⟩

definition (**in** −) *vmtf-unset* :: ‹*nat* $\Rightarrow$ *vmtf* $\Rightarrow$ *vmtf*› **where**
‹*vmtf-unset* = (λ*L* (*ns*, *m*, *fst-As*, *lst-As*, *next-search*).
(*if next-search = None* $\vee$ *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* $\in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
shows ‹*L* $\in$ *set ys'* $\longleftrightarrow$ *next-search = None* $\vee$ *stamp (ns ! (the next-search))* < *stamp (ns ! L)*›
⟨*proof*⟩

lemma *abs-vmtf-ns-unset-vmtf-unset*:
assumes *vmtf*:‹((*ns*, *m*, *fst-As*, *lst-As*, *next-search*)) $\in$ *vmtf* $\mathcal{A}$ *M*› **and**

L-N: ‹$L \in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)›
shows ‹(*vmtf-unset* L ((ns, m, *fst-As*, *lst-As*, *next-search*))) $\in$ *vmtf* $\mathcal{A}$ M› (**is** ‹*?S* $\in$ -›)
⟨*proof*⟩

definition (**in** −) *vmtf-dequeue-pre* **where**
‹*vmtf-dequeue-pre* = (λ(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* = (λ(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))›
⟨*proof*⟩

definition *vmtf-en-dequeue-pre* :: ‹*nat multiset* $\Rightarrow$ ((*nat*, *nat*) *ann-lits* $\times$ *nat*) $\times$ *vmtf* $\Rightarrow$ *bool*› **where**
‹*vmtf-en-dequeue-pre* $\mathcal{A}$ = (λ((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*) $\in$ ⟨*nat-rel*⟩*list-rel* $\rightarrow_f$ ⟨⟨*nat-rel*⟩*list-rel*⟩*nres-rel*›
⟨*proof*⟩

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*⟩

lemma *vmtf-vmtf-en-dequeue-pre-to-remove′*:
assumes *vmtf*: ‹(*vm*) $\in$ *vmtf* $\mathcal{A}$ M› **and**
i: ‹$A \in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)› **and** ‹*fst* (*snd vm*) + 1 $\leq$ *unat64-max*› **and**
A: ‹*isasat-input-nempty* $\mathcal{A}$›
shows ‹*vmtf-en-dequeue-pre* $\mathcal{A}$ ((M, A), *vm*)›
⟨*proof*⟩

lemma *wf-vmtf-get-next*:
assumes *vmtf*: ‹((ns, m, *fst-As*, *lst-As*, *next-search*)) $\in$ *vmtf* $\mathcal{A}$ M›
shows ‹*wf* {(*get-next* (ns ! *the* a), a) |a. $a \neq$ *None* $\wedge$ *the* $a \in$ *atms-of* ($\mathcal{L}_{all}$ $\mathcal{A}$)}› (**is** ‹*wf ?R*›)
⟨*proof*⟩

lemma *vmtf-next-search-take-next*:
assumes
vmtf: ‹((ns, m, *fst-As*, *lst-As*, *next-search*)) $\in$ *vmtf* $\mathcal{A}$ M› **and**
n: ‹*next-search* $\neq$ *None*› **and**
def-n: ‹*defined-lit* M (*Pos* (*the next-search*))›
shows ‹(ns, m, *fst-As*, *lst-As*, *get-next* (ns!*the next-search*)) $\in$ *vmtf* $\mathcal{A}$ M›
⟨*proof*⟩

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* $\mathcal{A}$ *M*›
  shows ‹*vmtf-find-next-undef* $\mathcal{A}$ (*ns*, *m*, *fst-As*, *lst-As*, *next-search*) *M*
    ≤ ⇓ *Id* (*SPEC* (λ*L*. (*ns*, *m*, *fst-As*, *lst-As*, *L*) ∈ *vmtf* $\mathcal{A}$ *M* ∧
      (*L* = *None* ⟶ (∀ *L*∈#$\mathcal{L}_{all}$ $\mathcal{A}$. *defined-lit M L*)) ∧
      (*L* ≠ *None* ⟶ *Pos* (*the L*) ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ *undefined-lit M* (*Pos* (*the L*)))))›
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

lemma *length-vmtf-cons*[*simp*]: ‹*length* (*vmtf-cons ns L n m*) = *length ns*›
  ⟨*proof*⟩

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*⟩

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* $\mathcal{A}$› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
shows
‹*vmtf-rescale vm* ≤ *SPEC* (λ*vm*. *vm* ∈ *vmtf* $\mathcal{A}$ *M* ∧ *fst* (*snd vm*) ≤ *unat32-max*)›
(**is** ‹*?A* ≤ *?R*›)
⟨*proof*⟩

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* $\mathcal{A}$}›
⟨*proof*⟩

lemma *distinct-atoms-rel-empty-hash-iff*:
‹(([], *h*), {}) ∈ *distinct-atoms-rel* $\mathcal{A}$ ⟷ (∀ *L* ∈# $\mathcal{A}$. *L* < *length h*) ∧ (∀ *i*∈*set h*. *i* = *False*)›
⟨*proof*⟩

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*⟩

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›
⟨*proof*⟩

definition (**in** −) *isa-vmtf-unset* :: ‹nat ⇒ vmtf ⇒ vmtf› **where**
‹isa-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))))›

definition *vmtf-unset-pre* **where**
‹vmtf-unset-pre = (λL (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))›
⟨*proof*⟩

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›
⟨proof⟩

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›
⟨proof⟩

lemma *list-update-append2*: ‹i ≥ length xs ⟹ (xs @ ys)[i := x] = xs @ ys[i − length xs := x]›
⟨proof⟩

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›
⟨proof⟩

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-emtpy-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*⟩

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*))›
⟨*proof*⟩

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*))›
⟨*proof*⟩

lemma *lbd-empty-ref-lbd-empty*:
‹(*lbd-empty-ref*, *lbd-empty*) ∈ *lbd-ref* →$_f$ ⟨*lbd-ref*⟩*nres-rel*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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'*)›
⟨*proof*⟩

lemma *get-LBD-ref-get-LBD*:
‹(*get-LBD-ref*, *get-LBD*) ∈ *lbd-ref* →$_f$ ⟨*nat-rel*⟩*nres-rel*›
⟨*proof*⟩

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} →$_a$ *bool1-assn*›
⟨*proof*⟩

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 →$_a$ *bool1-assn*›
⟨*proof*⟩

sepref-def *lbd-empty-loop-code*
is ‹*lbd-empty-loop-ref*›
:: ‹*lbd-int-assn*^{d} →$_a$ *lbd-int-assn*›
⟨*proof*⟩

sepref-def *lbd-empty-cheap-code*
is ‹*lbd-empty-cheap-ref*›
:: ‹[λ(-, *stamp*, -). *stamp* < *unat32-max*]$_a$ *lbd-int-assn*^{d} → *lbd-int-assn*›
⟨*proof*⟩

lemma *unat32-max-alt-def*: *unat32-max* = *4294967295*
⟨*proof*⟩
sepref-register *lbd-empty-cheap-ref lbd-empty-loop-ref*

sepref-def *lbd-empty-code*
is ‹*lbd-empty-ref*›
:: ‹*lbd-int-assn*^{d} →$_a$ *lbd-int-assn*›
⟨*proof*⟩

lemma *lbd-empty-hnr*[*sepref-fr-rules*]:
‹(*lbd-empty-code*, *lbd-empty*) ∈ *lbd-assn*^{d} →$_a$ *lbd-assn*›

⟨*proof*⟩

sepref-def *empty-lbd-code*
is [] ‹*uncurry0* (*RETURN empty-lbd-ref*)›
:: ‹*unit-assn*^{k} →$_a$ *lbd-int-assn*›
⟨*proof*⟩

lemma *empty-lbd-ref-empty-lbd*:
‹(*uncurry0* (*RETURN empty-lbd-ref*), *uncurry0* (*RETURN empty-lbd*)) ∈ *unit-rel* →$_f$ ⟨*lbd-ref*⟩*nres-rel*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *get-LBD-code*
is [] ‹*get-LBD-ref*›
:: ‹*lbd-int-assn*^{k} →$_a$ *uint32-nat-assn*›
⟨*proof*⟩

lemma *get-LBD-hnr*[*sepref-fr-rules*]:
‹(*get-LBD-code*, *get-LBD*) ∈ *lbd-assn*^{k} →$_a$ *uint32-nat-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

schematic-goal *mk-free-lbd-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE lbd-assn ?fr*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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⟩

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
  }›
  ⟨proof⟩

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 W*$_0$ = *do* {
let m = *length W*$_0$;
(-, *W*) ← *WHILE*$_T$ λ(*i*, *W*). *length W* = *length W*$_0$ ∧ (∀ *K*. *mset* (*W* ! *K*) = *mset* (*W*$_0$! *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*, *W*$_0$);
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*
(*Tuple17-get-a*: '*a*)
(*Tuple17-get-b*: '*b*)
(*Tuple17-get-c*: '*c*)
(*Tuple17-get-d*: '*d*)
(*Tuple17-get-e*: '*e*)
(*Tuple17-get-f*: '*f*)
(*Tuple17-get-g*: '*g*)
(*Tuple17-get-h*: '*h*)
(*Tuple17-get-i*: '*i*)
(*Tuple17-get-j*: '*j*)
(*Tuple17-get-k*: '*k*)
(*Tuple17-get-l*: '*l*)
(*Tuple17-get-m*: '*m*)
(*Tuple17-get-n*: '*n*)
(*Tuple17-get-o*: '*o*)
(*Tuple17-get-p*: '*p*)
(*Tuple17-get-q*: '*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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *mset-as-position-atm-le-length*:
‹*mset-as-position xs P* ⟹ *L* ∈# *P* ⟹ *atm-of L* < *length xs*›
⟨*proof*⟩

lemma *mset-as-position-nth*:
‹*mset-as-position xs P* ⟹ *L* ∈# *P* ⟹ *xs* ! (*atm-of L*) = *Some* (*is-pos L*)›
⟨*proof*⟩

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*)›

⟨*proof*⟩

lemma *mset-as-position-tautology*: ‹*mset-as-position xs C* ⟹ ¬*tautology C*›
⟨*proof*⟩

lemma *mset-as-position-right-unique*:
assumes
map: ‹*mset-as-position xs D*› **and**
map': ‹*mset-as-position xs D'*›
shows ‹*D* = *D'*›
⟨*proof*⟩

lemma *mset-as-position-mset-union*:
fixes *P xs*
defines ‹*xs'* ≡ *fold* (λ*L 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'*)›
⟨*proof*⟩

lemma *mset-as-position-empty-iff*: ‹*mset-as-position xs* {#} ⟷ (∃ *n*. *xs* = *replicate n None*)›
⟨*proof*⟩

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* $\mathcal{A}$ = {((*n*, *xs*), *C*). *n* = *size C* ∧ *mset-as-position xs C* ∧
(∀ *L*∈*atms-of* (*$\mathcal{L}_{all}$* $\mathcal{A}$). *L* < *length xs*)}›

lemma *lookup-clause-rel-empty-iff*: ‹((*n*, *xs*), *C*) ∈ *lookup-clause-rel* $\mathcal{A}$ ⟹ *n* = *0* ⟷ *C* = {#}›
⟨*proof*⟩

lemma *conflict-atm-le-length*: ‹((*n*, *xs*), *C*) ∈ *lookup-clause-rel* $\mathcal{A}$ ⟹ *L* ∈ *atms-of* (*$\mathcal{L}_{all}$* $\mathcal{A}$) ⟹
L < *length xs*›
⟨*proof*⟩

lemma *conflict-le-length*:
assumes
c-rel: ‹((*n*, *xs*), *C*) ∈ *lookup-clause-rel* $\mathcal{A}$› **and**
L-$\mathcal{L}_{all}$: ‹*L* ∈# *$\mathcal{L}_{all}$* $\mathcal{A}$›
shows ‹*atm-of L* < *length xs*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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* $\mathcal{A} \Longrightarrow$ *literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ *C* $\Longrightarrow$ *size C* $\leq$ *1 + unat32-max div 2*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma (**in** −) *mset-as-position-length-not-None*:
‹*mset-as-position x2 C* ⟹ *size C* = *length* (*filter* ((≠) *None*) *x2*)›
⟨*proof*⟩

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*⟩

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*#})›
⟨*proof*⟩

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*#})›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*⟩

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*⟩

lemma *id-conflict-from-lookup*:
‹(*RETURN o id*, *conflict-from-lookup*) ∈ [λ(*n*, *xs*). ∃ *D*. ((*n*, *xs*), *D*) ∈ *lookup-clause-rel* $\mathcal{A}$]$_f$ *Id* →
⟨*lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*›
⟨*proof*⟩

lemma *lookup-clause-rel-exists-le-unat32-max*:
assumes *ocr*: ‹((*n*, *xs*), *D*) ∈ *lookup-clause-rel* $\mathcal{A}$› **and** ‹*n* > *0*› **and**
le-i: ‹∀ *k*<*i*. *xs* ! *k* = *None*› **and** *lits*: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ *D*› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$›
shows
‹∃ *j*. *j* ≥ *i* ∧ *j* < *length xs* ∧ *j* < *unat32-max* ∧ *xs* ! *j* ≠ *None*›
⟨*proof*⟩

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⟩

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⟩

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}›
  ⟨proof⟩

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⟩


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* $\mathcal{A}$ ⟷
((*n*, *xs*), *C*) ∈ *lookup-clause-rel* $\mathcal{A}$›
⟨*proof*⟩

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 $\mathcal{A}$ →$_f$ ⟨*bool-rel*⟩*nres-rel*›
⟨*proof*⟩

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* $\mathcal{A}$ → ⟨*bool-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma *option-lookup-clause-rel-update-None*:
assumes ‹((*False*, (*n*, *xs*)), *Some D*) ∈ *option-lookup-clause-rel* $\mathcal{A}$› **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* $\mathcal{A}$›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

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* $\mathcal{A}$› **and**
dist: ‹*distinct D*› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset D*)› **and**
tauto: ‹¬*tautology* (*mset D*)› **and**
lits-C: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ *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* $\mathcal{A}$›
shows

‹*lookup-conflict-merge init' M D* (*b*, *n*, *xs*) *clvls outl* ≤
⇓(*option-lookup-clause-rel* $\mathcal{A}$ ×$_r$ *Id* ×$_r$ *Id*)
(*merge-conflict-m-g init' M D* (*Some C*))›
(**is** ‹- ≤ ⇓ *?Ref ?Spec*›)
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*⟩

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*⟩

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* D_0' = (λ(*D*, *D'*). *D* ⊆# D_0' ∧ *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*›
⟨*proof*⟩

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*⟩

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*))›
  ⟨*proof*⟩

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 = (λcach.
      ∃ 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^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 = 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*)›

lemma *lit-redundant-rec-wl-ref-butlast*:
‹*lit-redundant-rec-wl-ref NU x* ⟹ *lit-redundant-rec-wl-ref NU* (*butlast x*)›
⟨*proof*⟩

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*⟩

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-*$\mathcal{L}_{in}$*-trail* 𝒜 *M*› **and**
ana: ‹(*analysis*, *analysis'*) ∈ *ana-lookups-rel NU*› **and**
lits-NU: ‹*literals-are-in-*$\mathcal{L}_{in}$*-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*⟩

context
fixes *x* :: ‹(*nat* ⇒ *minimize-status*) × (*nat* × *nat* × *bool*) *list* × *bool*› **and**
x' :: ‹(*nat* ⇒ *minimize-status*) × (*nat* × *nat* × *nat* × *nat*) *list* × *bool*›
assumes *x-x'*: ‹(*x*, *x'*) ∈ {((*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*)}›
begin

lemma *ccmin-lit-redundant-rec-wl-inv2*:
assumes ‹*lit-redundant-rec-wl-inv M NU D x'*›
shows ‹*lit-redundant-rec-wl-inv2 M NU D x*›

⟨*proof*⟩

context
assumes
‹*lit-redundant-rec-wl-inv2 M NU D x*› **and**
‹*lit-redundant-rec-wl-inv M NU D x'*›
begin

lemma *ccmin-cond*:
fixes *x1* :: ‹*nat* ⇒ *minimize-status*› **and**
x2 :: ‹(*nat* × *nat* × *bool*) *list* × *bool*› **and**
x1a :: ‹(*nat* × *nat* × *bool*) *list*› **and**
x2a :: ‹*bool*› **and** *x1b* :: ‹*nat* ⇒ *minimize-status*› **and**
x2b :: ‹(*nat* × *nat* × *nat* × *nat*) *list* × *bool*› **and**
x1c :: ‹(*nat* × *nat* × *nat* × *nat*) *list*› **and** *x2c* :: ‹*bool*›
assumes
‹*x2* = (*x1a*, *x2a*)›
‹*x* = (*x1*, *x2*)›
‹*x2b* = (*x1c*, *x2c*)›
‹*x'* = (*x1b*, *x2b*)›
shows ‹(*x1a* ≠ []) = (*x1c* ≠ [])›
⟨*proof*⟩

end

context
assumes
‹*case x of* (*cach*, *analyse*, *b*) ⇒ *analyse* ≠ []› **and**
‹*case x' of* (*cach*, *analyse*, *b*) ⇒ *analyse* ≠ []› **and**
inv2: ‹*lit-redundant-rec-wl-inv2 M NU D x*› **and**
‹*lit-redundant-rec-wl-inv M NU D x'*›
begin

context
fixes *x1* :: ‹*nat* ⇒ *minimize-status*› **and**
x2 :: ‹(*nat* × *nat* × *nat* × *nat*) *list* × *bool*› **and**
x1a :: ‹(*nat* × *nat* × *nat* × *nat*) *list*› **and** *x2a* :: ‹*bool*› **and**
x1b :: ‹*nat* ⇒ *minimize-status*› **and**
x2b :: ‹(*nat* × *nat* × *bool*) *list* × *bool*› **and**
x1c :: ‹(*nat* × *nat* × *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 ≠ []›
⟨*proof*⟩

lemma *ccmin-nempty*:
shows ‹x1c ≠ []›
⟨*proof*⟩

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*›
⟨*proof*⟩ **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*)›
⟨*proof*⟩ **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*))›
⟨*proof*⟩

lemma *ccmin-in-dom*:
shows *x1g-dom*: ‹*x1g* ∈# *dom-m NU*›
⟨*proof*⟩

lemma *ccmin-in-dom-le-length*:
shows ‹*x1h* < *length* (*NU* ∝ *x1g*)›
⟨*proof*⟩

lemma *ccmin-in-trail*:
shows ‹*NU* ∝ *x1g* ! *x1h* ∈ *lits-of-l M*›
⟨*proof*⟩

lemma *ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-x1g*:
shows ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*NU* ∝ *x1g*))›
⟨*proof*⟩

lemma *ccmin-le-unat32-max*:
‹*length* (*NU* ∝ *x1g*) ≤ *Suc* (*unat32-max div* 2)›
⟨*proof*⟩

lemma *ccmin-in-all-lits*:
shows ‹*NU* ∝ *x1g* ! *x1h* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
⟨*proof*⟩

lemma *ccmin-less-length*:
shows ‹*x2i* ≤ *length* (*NU* ∝ *x1g*)›
⟨*proof*⟩

lemma *ccmin-same-cond*:
shows ‹(*x2i* ≤ *x1i*) = (*x2f* ≤ *x1f*)›
⟨*proof*⟩

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*)}›
⟨*proof*⟩

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*›
⟨*proof*⟩ **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*›
⟨*proof*⟩ **lemma** *x1k-x1j*: ‹*x1k* = *x1j*› ‹*x1j* = *NU* ∝ *x1d* ! *x1f*› **and**
x2k-x2j: ‹(*x2k*, *x2j*) ∈ *ana-lookups-rel NU*›
⟨*proof*⟩

lemma *ccmin-x1k-all*:
shows ‹*x1k* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
⟨*proof*⟩

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'*
⟨*proof*⟩

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*)›
⟨*proof*⟩ **lemma** *ccmin-lit-redundant-rec-wl-inv*: ‹*lit-redundant-rec-wl-inv M NU D*
(*x1*, *x2j*, *False*)›
⟨*proof*⟩

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*)}›
⟨*proof*⟩

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)›
⟨*proof*⟩

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›
⟨*proof*⟩

lemma *ccmin-mark-failed-lits-wl-lbd*:
shows ‹mark-failed-lits-wl NU x2k x1b
≤ ⇓ Id
(mark-failed-lits-wl NU x2j x1)›
⟨*proof*⟩

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*)}›
⟨*proof*⟩

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›
⟨*proof*⟩

lemma *ccmin-lit-eq*:
‹− x1k = − x1j›
⟨*proof*⟩

context
fixes *xa* :: ‹*nat option*› **and** *x'a* :: ‹*nat option*›

assumes *xa-x'a*: ‹$(xa,\ x'a) \in \langle nat\text{-}rel\rangle option\text{-}rel$›
begin

lemma *ccmin-lit-eq2*:
‹$(xa,\ x'a) \in Id$›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *ccmin-mark-failed-lits-stack-wl-dec*:
shows ‹*mark-failed-lits-wl NU x2k x1b*
$\leq \Downarrow Id$
(*mark-failed-lits-wl NU x2j x1*)›
⟨*proof*⟩

lemma *ccmin-rel-dec*:
fixes *cach* :: ‹*nat* ⇒ *minimize-status*› **and** *cacha* :: ‹*nat* ⇒ *minimize-status*›
assumes ‹$(cach,\ cacha) \in Id$›
shows ‹$((cach,\ [],\ False),\ cacha,\ [],\ False)$
$\in \{((cach,\ ana,\ b),\ cach',\ ana',\ b').$
$(ana,\ ana') \in$ *ana-lookups-rel NU* $\wedge$
$b = b' \wedge cach = cach' \wedge$ *lit-redundant-rec-wl-inv M NU D* $(cach,\ ana',\ b)\}$›
⟨*proof*⟩

end

context
fixes *xb* :: ‹*nat*› **and** *x'b* :: ‹*nat*›
assumes *H*:
‹$xa = Some\ xb$›
‹$x'a = Some\ x'b$›
‹$(xb,\ x'b) \in$ *nat-rel*›
‹$x'b \in\#$ *dom-m NU*›
‹$2 \leq$ *length* $(NU \propto x'b)$›
‹$x'b > 0$›
‹*distinct* $(NU \propto x'b) \wedge \neg$ *tautology* (*mset* $(NU \propto x'b))$›
begin

lemma *ccmin-stack-pre*:
shows ‹$xb \in\#$ *dom-m NU*› ‹$2 \leq$ *length* $(NU \propto xb)$›
⟨*proof*⟩

lemma *ccmin-literals-are-in-*$\mathcal{L}_{in}$*-NU-xb*:
shows ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* $(NU \propto xb))$›
⟨*proof*⟩

lemma *ccmin-le-unat32-max-xb*:
‹*length* (*NU* ∝ *xb*) ≤ *Suc* (*unat32-max div* 2)›
⟨*proof*⟩ **lemma** *ccmin-lit-redundant-rec-wl-inv3*: ‹*lit-redundant-rec-wl-inv M NU D*
(*x1*, *x2j* @ [*lit-redundant-reason-stack* (− *NU* ∝ *x1d* ! *x1f*) *NU x'b*], *False*)›
⟨*proof*⟩

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*)}›
⟨*proof*⟩

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*⟩

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 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-ℒ_in-trail A M›
    ‹literals-are-in-ℒ_in-mm A ((mset ∘ fst) `# ran-m NU)› and
    ‹isasat-input-bounded A›
  shows
    ‹literal-redundant-wl-lookup 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⟩


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 A = (λM 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 ∈# ℒ_all A);
          (s', -, red) ← literal-redundant-wl-lookup A M NU nxs s L lbd;
          if ¬red
          then RETURN (nxs, x+1, s', outl)
          else do {
            ASSERT (delete-from-lookup-conflict-pre 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* $\models p$ *E*› **and** *L-E*: ‹*L* $\in\#$ *E*›
shows ‹*NU* $\models p$ *remove1-mset L E*›
⟨*proof*⟩

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* $\models as$ *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**
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*⟩

definition *cach-refinement-list*
:: ‹*nat multiset* ⇒ (*minimize-status list* × (*nat conflict-min-cach*)) *set*›
where
‹*cach-refinement-list* $\mathcal{A}_{in}$ = ⟨*Id*⟩*map-fun-rel* {(*a*, *a'*). *a* = *a'* ∧ *a* ∈# $\mathcal{A}_{in}$}›

definition *cach-refinement-nonull*
:: ‹*nat multiset* ⇒ ((*minimize-status list* × *nat list*) × *minimize-status list*) *set*›
where
‹*cach-refinement-nonull* $\mathcal{A}$ = {((*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* $\mathcal{A}$}›

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* $\mathcal{A}_{in}$}›
⟨*proof*⟩

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* $\mathcal{A}_{in}$›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

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* $\subseteq$ *set-mset* $\mathcal{A}$›
  shows ‹*length n* $\leq$ *Suc* (*unat32-max div 2*)›
⟨*proof*⟩

lemma *conflict-min-cach-set-failed*:
  ‹(*uncurry conflict-min-cach-set-failed-l*, *uncurry* (*RETURN oo conflict-min-cach-set-failed*)) $\in$
    [λ(*cach*, *L*). *L* $\in\#$ $\mathcal{A}_{in}$ $\wedge$ *isasat-input-bounded* $\mathcal{A}_{in}$]$_f$ *cach-refinement* $\mathcal{A}_{in}$ $\times_r$ *nat-rel* $\rightarrow$ ⟨*cach-refinement* $\mathcal{A}_{in}$⟩*nres-rel*›
  ⟨*proof*⟩

definition (**in** −) *conflict-min-cach-set-removable*
  :: ‹*nat conflict-min-cach* $\Rightarrow$ *nat* $\Rightarrow$ *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*)) $\in$
    [λ(*cach*, *L*). *L* $\in\#$ $\mathcal{A}_{in}$ $\wedge$ *isasat-input-bounded* $\mathcal{A}_{in}$]$_f$ *cach-refinement* $\mathcal{A}_{in}$ $\times_r$ *nat-rel* $\rightarrow$ ⟨*cach-refinement* $\mathcal{A}_{in}$⟩*nres-rel*›
  ⟨*proof*⟩

definition *isa-mark-failed-lits-stack* **where**
  ‹*isa-mark-failed-lits-stack NU analyse cach* = *do* {
    let l = *length analyse*;
    ASSERT(*length analyse* $\leq$ *1* + *unat32-max div 2*);
    (-, *cach*) $\leftarrow$ *WHILE*$_T^{\lambda(\text{-}, \text{cach}).\ \text{True}}$
      (λ(*i*, *cach*). *i* < *l*)
      (λ(*i*, *cach*). *do* {
        ASSERT(*i* < *length analyse*);
        let (*cls-idx*, *idx*, -) = (*analyse* ! *i*);
        ASSERT(*cls-idx* + *idx* $\geq$ *1*);
        ASSERT(*cls-idx* + *idx* − *1* < *length NU*);
 ASSERT(*arena-lit-pre NU* (*cls-idx* + *idx* − *1*));
 cach $\leftarrow$ *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'* $\Longrightarrow$
      a1' < *length ba* $\Longrightarrow$
      (*a1'a*, *a2'a*) = *ba* ! *a1'* $\Longrightarrow$
      a1'a $\in\#$ *dom-m a*›
  ⟨*proof*⟩

lemma *mark-failed-lits-stack-inv-helper2*: ‹*mark-failed-lits-stack-inv a ba a2'* $\Longrightarrow$
      a1' < *length ba* $\Longrightarrow$

$(a1'a,\ xx,\ a2'a,\ yy) = ba\ !\ a1' \Longrightarrow$
$a2'a - Suc\ 0 < length\ (a \propto a1'a)$›
⟨*proof*⟩

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* $\mathcal{A}_{in}$)) ∈
$[\lambda((N,\ ana),\ cach).\ length\ ana \leq 1 +\ unat32\text{-}max\ div\ 2]_f$
$\{(arena,\ N).\ valid\text{-}arena\ arena\ N\ vdom\} \times_f ana\text{-}lookups\text{-}rel\ NU \times_f cach\text{-}refinement\ \mathcal{A}_{in} \rightarrow$
⟨*cach-refinement* $\mathcal{A}_{in}$⟩*nres-rel*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *arena-lit-pre-le2*: ‹*length a* ≤ *unat64-max* ⟹
arena-lit-pre a i ⟹ *i* < *unat64-max*›
⟨*proof*⟩

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^λ-. 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)›
  ⟨proof⟩

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)›
⟨proof⟩

lemma valid-arena-nempty:
  ‹valid-arena arena N vdom ⟹ i ∈# dom-m N ⟹ N ∝ i ≠ []›
```

⟨*proof*⟩

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*⟩

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*(λ*D'*. *D'* ⊆# *D* ∧
NU + *NUE* ⊨*pm add-mset K D'*))›
⟨*proof*⟩

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'''* ≡ *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**
NUE: ‹*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-*$\mathcal{L}_{in}$*-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* ⊨*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* ($\lambda D'$. $D' \subseteq\# D \wedge NU' + NUE + NUS + N0S \models pm$ *add-mset K* D'))⟩
⟨*proof*⟩

lemma (**in** −) *lookup-conflict-upd-None-RETURN-def*:
⟨*RETURN oo lookup-conflict-upd-None* = ($\lambda(n, xs)$ *i*. *RETURN* ($n-$ *1*, *xs* [*i* := *NOTIN*]))⟩
⟨*proof*⟩

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* ∈# $\mathcal{A}$⟩
⟨*proof*⟩

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*⟩

definition (**in** −) *lookup-conflict-remove1* :: ⟨*nat literal* ⇒ *lookup-clause-rel* ⇒ *lookup-clause-rel*⟩ **where**
⟨*lookup-conflict-remove1* =
(λL (*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*⟩
⟨*proof*⟩

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$_T$λ(*nxs*, *i*, *s*, *outl*). *length outl* ≤ *unat32-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$_T$λ(*nxs*, *i*, *s*, *outl*). *length outl* ≤ *unat32-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*)
})›
⟨*proof*⟩

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-*$\mathcal{L}_{in}$*-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* $\mathcal{A}$ ×$_r$ *cach-refinement* $\mathcal{A}$ ×$_r$ *Id*⟩*nres-rel*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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* $\mathcal{A}$› **and**
dist: ‹*distinct D*› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset D*)› **and**
lits-tr: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ *M*› **and**
n-d: ‹*no-dup M*› **and**
tauto: ‹¬*tautology* (*mset D*)› **and**
lits-C: ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ *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* $\mathcal{A}$› **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* $\mathcal{A}$ ×$_r$ *Id* ×$_r$ *Id*)
(*merge-conflict-m-eq2 L M D* (*Some C*))›
(**is** ‹- ≤ ⇓ *?Ref ?Spec*›)
⟨*proof*⟩

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-*$\mathcal{L}_{in}$*-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
‹*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*⟩

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 ×$_f$ *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*⟩

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*›
⟨*proof*⟩

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* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$› **and** *uaa*: ‹− *aa* ∉# *the ba*› **and**
‹((*b*, *c*, *d*), *ba*) ∈ *option-lookup-clause-rel* $\mathcal{A}$›
shows ‹*is-in-option-lookup-conflict aa* (*b*, *c*, *d*) =
atm-is-in-conflict aa ba›
⟨*proof*⟩

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* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ −*L* ∉# *the D*]$_f$
Id ×$_f$ *option-lookup-clause-rel* $\mathcal{A}$ → ⟨*bool-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma *is-in-option-lookup-conflict-alt-def*:
‹*RETURN oo is-in-option-lookup-conflict* =
RETURN oo (λ*L* (-, *n*, *xs*). *is-in-lookup-conflict* (*n*, *xs*) *L*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

lemma *AIvdom-split*:

‹*aivdom* = *AIvdom* (*get-vdom-aivdom aivdom*, *get-avdom-aivdom aivdom*, *get-ivdom-aivdom aivdom*, *get-tvdom-aivdom aivdom*)›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*}›
⟨*proof*⟩

lemma *distinct-remove-readd-last-set*:
‹*distinct xs* ⟹ *i* < *length xs* ⟹ *set* (*butlast* (*xs*[*i* := *last xs*])) = *set xs* − {*xs*!*i*}›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*⟩
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*)›
⟨*proof*⟩

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*⟩

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*)›
⟨*proof*⟩

lemma *aivdom-inv-remove-clause*:
‹*aivdom-inv ai baa* ⟹ *aivdom-inv ai* (*remove1-mset C baa*)›
⟨*proof*⟩

lemma *aivdom-inv-dec-remove-clause*:
‹*aivdom-inv-dec ai baa* ⟹ *aivdom-inv-dec ai* (*remove1-mset C baa*)›

⟨*proof*⟩

lemma *aivdom-inv-removed-inactive*:
assumes ‹$i < length\ n$› **and** ‹$aivdom\text{-}inv\ (m,\ n,\ s,\ tv)\ baa$› ‹$n!i \notin\# baa$›
shows ‹$aivdom\text{-}inv\ (m,\ butlast\ (n[i := last\ n]),\ s,\ tv)\ baa$›
⟨*proof*⟩

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 ‹$aivdom\text{-}inv\text{-}dec\ (remove\text{-}inactive\text{-}aivdom\ i\ ai)\ baa$›
⟨*proof*⟩

lemmas *aivdom-inv-remove-and-swap-removed* = *aivdom-inv-removed-inactive*

lemma *aivdom-inv-removed-inactive-tvdom*:
assumes ‹$i < length\ tv$› **and** ‹$aivdom\text{-}inv\ (m,\ n,\ s,\ tv)\ baa$› ‹$tv!i \notin\# baa$›
shows ‹$aivdom\text{-}inv\ (m,\ n,\ s,\ butlast\ (tv[i := last\ tv]))\ baa$›
⟨*proof*⟩

lemma *aivdom-inv-dec-removed-inactive-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 ‹$aivdom\text{-}inv\text{-}dec\ (remove\text{-}inactive\text{-}aivdom\text{-}tvdom\ i\ ai)\ baa$›
⟨*proof*⟩

lemma *get-aivdom-remove-inactive-aivdom*[*simp*]:
‹$get\text{-}vdom\text{-}aivdom\ (remove\text{-}inactive\text{-}aivdom\ i\ m) = get\text{-}vdom\text{-}aivdom\ m$›
‹$get\text{-}avdom\text{-}aivdom\ (remove\text{-}inactive\text{-}aivdom\ i\ m) = (delete\text{-}index\text{-}and\text{-}swap\ (get\text{-}avdom\text{-}aivdom\ m)\ i)$›
‹$get\text{-}ivdom\text{-}aivdom\ (remove\text{-}inactive\text{-}aivdom\ i\ m) = get\text{-}ivdom\text{-}aivdom\ m$›
‹$get\text{-}tvdom\text{-}aivdom\ (remove\text{-}inactive\text{-}aivdom\ i\ m) = get\text{-}tvdom\text{-}aivdom\ m$›
‹$get\text{-}vdom\text{-}aivdom\ (remove\text{-}inactive\text{-}aivdom\text{-}tvdom\ i\ m) = get\text{-}vdom\text{-}aivdom\ m$›
‹$get\text{-}tvdom\text{-}aivdom\ (remove\text{-}inactive\text{-}aivdom\text{-}tvdom\ i\ m) = (delete\text{-}index\text{-}and\text{-}swap\ (get\text{-}tvdom\text{-}aivdom\ m)\ i)$›
‹$get\text{-}avdom\text{-}aivdom\ (remove\text{-}inactive\text{-}aivdom\text{-}tvdom\ i\ m) = get\text{-}avdom\text{-}aivdom\ m$›
‹$get\text{-}ivdom\text{-}aivdom\ (remove\text{-}inactive\text{-}aivdom\text{-}tvdom\ i\ m) = get\text{-}ivdom\text{-}aivdom\ m$›
⟨*proof*⟩

definition *vdom-aivdom-at-int* :: ‹$aivdom \Rightarrow nat \Rightarrow nat$› **where**
‹$vdom\text{-}aivdom\text{-}at\text{-}int\ = (\lambda(a,b,c)\ C.\ a\ !\ C)$›

definition *vdom-aivdom-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-aivdom-at-alt-def*:
‹$vdom\text{-}aivdom\text{-}at = vdom\text{-}aivdom\text{-}at\text{-}int\ o\ get\text{-}content$›
⟨*proof*⟩

definition *avdom-aivdom-at-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* :: ‹$isasat\text{-}aivdom \Rightarrow nat \Rightarrow nat$› **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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

lemma *aivdom-inv-dec-intro-learned-strong-add-mset*:
‹*C* ∉# *d* ⟹ *C* ∉ *set* (*get-vdom-aivdom aivdom*) ⟹ *aivdom-inv-strong-dec aivdom d* ⟹ *aivdom-inv-strong-dec* (*add-learned-clause-aivdom-strong C aivdom*) (*add-mset C d*)›
⟨*proof*⟩

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*]›
⟨*proof*⟩

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*) {#}›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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* $\mathcal{B}$ = ($\lambda(\mathcal{A}, xs)$. *set-mset* $\mathcal{B}$ = $\mathcal{A}$)›

Only useful for proofs.

definition *occ-list* :: ‹*occurences* ⇒ *nat literal* ⇒ *nat list*› **where**
‹*occ-list* = ($\lambda(\mathcal{A}, xs)$ *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* = ($\lambda(n, xs)$ *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* = ($\lambda\mathcal{A}xs$ *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* = ($\lambda(\mathcal{A}, xs)$ *L*. *atm-of L* ∈ $\mathcal{A}$)›

definition *occ-list-length* :: ‹*occurences* ⇒ *nat literal* ⇒ *nat*› **where**
‹*occ-list-length* = ($\lambda(\mathcal{A}, xs)$ *L*. *do* {
(*length* (*xs L*))
})›

definition *mop-occ-list-length* :: ‹*occurences* ⇒ *nat literal* ⇒ *nat nres*› **where**
‹*mop-occ-list-length* = ($\lambda\mathcal{A}xs$ *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* = ($\lambda(\mathcal{A}, xs)$ *L*. *atm-of L* ∈ $\mathcal{A}$)›

definition *occ-list-append* :: ‹*nat* ⇒ *occurences* ⇒ *nat literal* ⇒ *occurences*› **where**
‹*occ-list-append* = (λx ($\mathcal{A}$, *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* = (λx $\mathcal{A}xs$ *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(𝒜, *xs*) *L*. *atm-of L* ∈ 𝒜)›

definition *occ-list-clear-at* :: ‹*occurences* ⇒ *nat literal* ⇒ *occurences nres*› **where**
‹*occ-list-clear-at* = (λ(𝒜, *xs*) *L* . *do* {
ASSERT (*occ-list-clear-at-pre* (𝒜, *xs*) *L*);
RETURN (𝒜, *xs* (*L* := []))
})›

definition *occ-list-clear-all-pre* :: ‹*occurences* ⇒ *bool*› **where**
‹*occ-list-clear-all-pre* = (λ(𝒜, *xs*). *True*)›

definition *occ-list-clear-all* :: ‹*occurences* ⇒ *occurences nres*› **where**
‹*occ-list-clear-all* = (λ(𝒜, *xs*) . *do* {
ASSERT (*occ-list-clear-all-pre* (𝒜, *xs*));
RETURN (𝒜, λ-. [])
})›

definition *all-occurrences* :: ‹*nat multiset* ⇒ *occurences* ⇒ *nat multiset*› **where**
‹*all-occurrences* 𝒜 = (λ(*n*, *xs*). ∑# (*mset* ‘# *xs* ‘# *Pos* ‘# *remdups-mset* 𝒜 +
mset ‘# *xs* ‘# *Neg* ‘# *remdups-mset* 𝒜))›

definition *occ-list-create-pre* :: ‹*nat set* ⇒ *nat set* ⇒ *bool*› **where**
‹*occ-list-create-pre n* = (λ𝒜. *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* 𝒜 ≡ (*set-mset* 𝒜, λ-. [])›

lemma *empty-occs-list-cong*:
‹*set-mset A* = *set-mset B* ⟹ *empty-occs-list A* = *empty-occs-list B*›
⟨*proof*⟩

definition *occurrence-list* **where**
‹*occurrence-list* 𝒜 = {((*n*, *ys*), *xs*). (*ys*, *xs*)∈*Id* ∧ *n* = (*set-mset* 𝒜)}›

lemma *mop-occ-list-create*:
shows ‹*mop-occ-list-create* (*set-mset* 𝒜) ≤ *SPEC* (λ*c*. (*c*, *raw-empty-occs-list*) ∈ *occurrence-list* 𝒜)›
(**is** ‹*?A* ≤ *?B*›)
⟨*proof*⟩

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*›)
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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$\mathcal{A}$*xs L i*. *do* {
ASSERT (*cocc-list-at-pre* $\mathcal{A}$*xs L i*);
RETURN (*cocc-list-at* $\mathcal{A}$*xs 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*}›
⟨*proof*⟩

lemma *mop-cocc-list-at-mop-occ-list-at*:
assumes
‹(*xs*, $\mathcal{A}$*xs*) ∈ *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* $\mathcal{A}$*xs 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* $\mathcal{A}$*xs L′ i′*)›

⟨*proof*⟩

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*Axs L*. *do* {
ASSERT (*cocc-list-length-pre Axs L*);
RETURN (*cocc-list-length Axs L*)
})›

lemma *mop-cocc-list-length-mop-occ-list-length*:
assumes
‹(*xs*, *Axs*) ∈ *occurrence-list-ref*›
‹(*L*,*L*′)∈*Id*›
shows
‹*mop-cocc-list-length xs L* ≤ ⇓*nat-rel* (*mop-occ-list-length Axs L*′)›
⟨*proof*⟩

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 Axs L* . *do* {
ASSERT (*cocc-list-append-pre* (*Axs*) *L*);
RETURN (*cocc-list-append x* (*Axs*) *L*)
})›

lemma *mop-cocc-list-append-mop-occ-list-append*:
assumes
‹(*xs*, *Axs*) ∈ *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*′ *Axs L*′)›
⟨*proof*⟩

definition *mop-cocc-list-create* :: ‹*nat* ⇒ *occurences-ref nres*› **where**
‹*mop-cocc-list-create* = (λ*n*. *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*)›

⟨*proof*⟩

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* = (λ*xs 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'*)›
⟨*proof*⟩

definition *cocc-list-clear-all-pre* :: ‹*occurences-ref* ⇒ *bool*› **where**
‹*cocc-list-clear-all-pre* = (λ*xs*. *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*, *Axs*) ∈ *occurrence-list-ref*›
shows
‹*cocc-list-clear-all xs* ≤ ⇓(*occurrence-list-ref*) (*occ-list-clear-all Axs*)›
⟨*proof*⟩

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(𝒜, *C*). (𝒜, *add-mset L C*))›

definition *ch-add-pre* :: ‹*nat literal* ⇒ *clause-hash* ⇒ *bool*› **where**
‹*ch-add-pre L* = (λ(𝒜, *C*). *atm-of L* ∈ 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* = (λ(𝒜, *C*). (𝒜, *remove1-mset L C*))›

definition *ch-remove-pre* :: ‹*nat literal* ⇒ *clause-hash* ⇒ *bool*› **where**
‹*ch-remove-pre L* = (λ(𝒜, *C*). *atm-of L* ∈ 𝒜 ∧ *L* ∈# *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* = (λ(𝒜, *C*). *L* ∈# *C*)›

definition *ch-in-pre* :: ‹*nat literal* ⇒ *clause-hash* ⇒ *bool*› **where**
‹*ch-in-pre L* = (λ(𝒜, *C*). *atm-of L* ∈ 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* = (λ(𝒜, *C*). (𝒜, *C* − *D*))›

definition *ch-remove-clause-pre* :: ‹*nat clause* ⇒ *clause-hash* ⇒ *bool*› **where**
‹*ch-remove-clause-pre D* = (λ(𝒜, *C*). *D* ⊆# *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* = (λ(*A*, *C*). (*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* ∧ *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* = (λ(*A*, *C*). (*A*, *C* + *D*))›

definition *ch-add-all-pre* :: ‹*nat clause* ⇒ *clause-hash* ⇒ *bool*› **where**
‹*ch-add-all-pre D* = (λ(*A*, *C*). *atms-of D* ⊆ *A* ∧ *C* ∩# *D* = {#} ∧ *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 A*) ≤ *SPEC* (λ*c*. (*c*, {#}) ∈ *clause-hash-ref A*)› (**is** ‹*?A* ≤ *?B*›)
⟨*proof*⟩

lemma *mop-ch-add*:
assumes ‹(*C*, *D*) ∈ *clause-hash-ref A*› **and** ‹*atm-of L* ∈# *A*› **and** ‹(*L*,*L′*)∈*Id*› ‹*L*∉#*D*›
shows ‹*mop-ch-add L C* ≤ *SPEC*(λ*c*. (*c*, *add-mset L′ D*) ∈ *clause-hash-ref A*)›
⟨*proof*⟩

lemma *mop-ch-add-all*:
assumes ‹(*C*, *D*) ∈ *clause-hash-ref A*› **and** ‹*atms-of L* ⊆ *set-mset 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 A*)›
⟨*proof*⟩

lemma *mop-ch-in*:
assumes ‹(*C*, *D*) ∈ *clause-hash-ref A*› **and** ‹*atm-of L* ∈# *A*› **and** ‹(*L*,*L′*)∈*Id*›
shows ‹*mop-ch-in L C* ≤ *SPEC*(λ*c*. (*c*, *L′* ∈# *D*) ∈ *bool-rel*)›
⟨*proof*⟩

lemma *mop-ch-remove*:
assumes ‹(*C*, *D*) ∈ *clause-hash-ref A*› **and** ‹*atm-of L* ∈# *A*› **and** ‹(*L*,*L′*)∈*Id*› ‹*L*∈#*D*›
shows ‹*mop-ch-remove L C* ≤ *SPEC*(λ*c*. (*c*, *remove1-mset L′ D*) ∈ *clause-hash-ref A*)›
⟨*proof*⟩

lemma *mop-ch-remove-all*:
assumes ‹(*C*, *D*) ∈ *clause-hash-ref A*› ‹*atm-of* ‘ *set-mset D* ⊆ *set-mset A*›
shows ‹*mop-ch-remove-all D C* ≤ *SPEC*(λ*c*. (*c*, {#}) ∈ *clause-hash-ref A*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

interpretation *ACIDS*: *hmstruct-with-prio* **where**
le = ‹(≥) :: *nat* ⇒ *nat* ⇒ *bool*› **and**
lt = ‹(>)›
⟨*proof*⟩

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* $\mathcal{A}$ *M* ⟹ *L* ∈# $\mathcal{A}$ ⟹ *M* ≠ [] ⟹ *L* = *atm-of* (*lit-of* (*hd M*)) ⟹ *acids-tl L ac* ≤ *RES* (*acids* $\mathcal{A}$ (*tl M*))›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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* $\mathcal{A}$ = (λ*ac M*. *do* {
WHILE$_T$(λ(*L*, *ac*). (*L* = *None* ⟶ *ac* ∈ *acids* $\mathcal{A}$ *M*) ∧ (*L* ≠ *None* ⟶ *ac* ∈ *acids* $\mathcal{A}$ (*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* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$);
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* $\mathcal{A}$ *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* $\mathcal{A}$ *M*) ∧
(*undefined-lit M* (*Pos v*) ⟶ *ac* ∈ *acids* $\mathcal{A}$ (*Decided* (*Pos v*) # *M*)))›
⟨*proof*⟩

lemma *acids-find-next-undef*:
assumes
vmtf: ‹*ac* ∈ *acids* $\mathcal{A}$ *M*›
shows ‹*acids-find-next-undef* $\mathcal{A}$ *ac M*
≤ ⇓ *Id* (*SPEC* (λ(*L*, *ac*).
(*L* = *None* ⟶ *ac* ∈ *acids* $\mathcal{A}$ *M* ∧ (∀ *L*∈#$\mathcal{L}_{all}$ $\mathcal{A}$. *defined-lit M L*)) ∧
(*L* ≠ *None* ⟶ *ac* ∈ *acids* $\mathcal{A}$ (*Decided* (*Pos* (*the L*)) # *M*) ∧ *Pos* (*the L*) ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ *undefined-lit*
M (*Pos* (*the L*)))))›
⟨*proof*⟩

definition *acids-push-literal-pre* **where**
‹*acids-push-literal-pre* $\mathcal{A}$ *L* = (λ*ac*. *L* ∈# $\mathcal{A}$)›

definition *acids-push-literal* :: ‹'a ⇒ ('a, nat) acids ⇒ ('a, nat) acids nres› **where**
‹*acids-push-literal L* = (λ(*ac*, *m*). *do* {
ASSERT (*L* ∈# *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* 𝒜 *M* ⟹ *acids-push-literal-pre* 𝒜 *L ac* ⟹ *acids-push-literal L ac* ≤ *SPEC* (λ*ac*. *ac* ∈ *acids* 𝒜 *M*)›
⟨*proof*⟩

definition *acids-flush-int* :: ‹*nat multiset* ⇒ (*nat*,*nat*) *ann-lits* ⇒ (*nat*, *nat*) *acids* ⇒ - ⇒ ((*nat*, *nat*) *acids* × -)*nres*› **where**
‹*acids-flush-int* 𝒜*in* = (λ*M 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* 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* ∈# 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* 𝒜*in* = (λ*M vm remove-int*. *SPEC* (λ*x*. (*fst x*) ∈ *acids* 𝒜*in M* ∧ *snd x* = {}))›

lemma *acids-change-to-remove-order*:
assumes
vmtf: ‹*ac* ∈ *acids* 𝒜*in M*› **and**
CD-rem: ‹((*C*, *D*), *to-remove*) ∈ *distinct-atoms-rel* 𝒜*in*› **and**
nempty: ‹*isasat-input-nempty* 𝒜*in*› **and**
bounded: ‹*isasat-input-bounded* 𝒜*in*› **and**
t: ‹*to-remove* ⊆ *set-mset* 𝒜*in*›
shows ‹*acids-flush-int* 𝒜*in M ac* (*C*, *D*) ≤ ⇓(*Id* ×$_r$ *distinct-atoms-rel* 𝒜*in*) (*acids-flush* 𝒜*in M ac to-remove*)›
⟨*proof*⟩
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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›

⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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* 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*›
⟨*proof*⟩

lemma *switch-bump-heur*:
assumes ‹*x* ∈ *bump-heur* 𝒜 *M*› **and**
‹*count-decided M* = *0*›
shows ‹*switch-bump-heur x* ∈ *bump-heur* 𝒜 *M*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

lemma *get-unit-trail-tl*[*simp*]: ‹*count-decided M > 0 ⟹ get-unit-trail* (*tl M*) = *get-unit-trail M*›
⟨*proof*⟩

lemma *isa-vmtf-consD*:
‹*x ∈ bump-heur A M ⟹ x ∈ bump-heur A* (*L # M*)›
⟨*proof*⟩

lemma *isa-bump-unset-vmtf-tl*:
fixes *M*
defines [*simp*]: ‹*L ≡ atm-of* (*lit-of* (*hd M*))›
assumes *vmtf*: ‹*x∈ bump-heur A M*› **and**
L-N: ‹*L ∈ atms-of* (*$\mathcal{L}_{all}$ A*)› **and** [*simp*]: ‹*M ≠* []› **and**
nz: ‹*count-decided M > 0*›
shows ‹*isa-bump-unset L x ≤ SPEC* (*λa. a ∈ bump-heur A* (*tl M*))›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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)›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *get-learned-count-learned-clss-countD2*:
‹*get-learned-count S* = (*get-learned-count T*) ⟹
learned-clss-count S = *learned-clss-count T*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)) ∧

$(D', D) \in$ *option-lookup-clause-rel* (*all-atms-st* T) $\wedge$
$(W', W) \in \langle Id \rangle$*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* $\wedge$
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*
}›

lemma *twl-st-heur-twl-st-heur-conflict-ana*:
‹$(S, T) \in$ *twl-st-heur* $\Longrightarrow$ $(S, T) \in$ *twl-st-heur-conflict-ana*›
⟨*proof*⟩

lemma *twl-st-heur-ana-state-simp*:
assumes ‹$(S, S') \in$ *twl-st-heur-conflict-ana*›
shows
‹(*get-trail-wl-heur* S, *get-trail-wl* S') $\in$ *trail-pol* (*all-atms-st* S')› **and**
‹$C \in\# \mathcal{L}_{all}$ (*all-atms-st* S') $\Longrightarrow$ *watched-by-int* S C = *watched-by* S' C›
⟨*proof*⟩

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) $\in$ *trail-pol* (*all-atms-st* T) $\wedge$
valid-arena N' N (*set* (*get-vdom-aivdom vdom*)) $\wedge$
(D', *None*) $\in$ *option-lookup-clause-rel* (*all-atms-st* T) $\wedge$
(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 *None* $\wedge$
cach-refinement-empty (*all-atms-st* T) *cach* $\wedge$
out-learned M *None 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*) ∧
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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)})›
⟨*proof*⟩

lemma *all-lits-st-alt-def*: ‹*set-mset* (*all-lits-st S*) = *set-mset* ($\mathcal{L}_{all}$ (*all-atms-st S*))›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

lemma *trail-pol-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *L* ∈ *trail-pol* $\mathcal{A}$ ⟹ *L* ∈ *trail-pol* $\mathcal{B}$›
⟨*proof*⟩

lemma *distinct-atoms-rel-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *L* ∈ *distinct-atoms-rel* $\mathcal{A}$ ⟹ *L* ∈ *distinct-atoms-rel* $\mathcal{B}$›
⟨*proof*⟩

lemma *phase-saving-rel-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *phase-saving* $\mathcal{A}$ *heur* ⟹ *phase-saving* $\mathcal{B}$ *heur*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *heuristic-rel-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *heuristic-rel* $\mathcal{A}$ *heur* ⟹ *heuristic-rel* $\mathcal{B}$ *heur*›

⟨*proof*⟩

lemma *vmtf-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ $L \in$ *vmtf* $\mathcal{A}$ M ⟹ $L \in$ *vmtf* $\mathcal{B}$ M›
⟨*proof*⟩

lemma *acids-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ $L \in$ *acids* $\mathcal{A}$ M ⟹ $L \in$ *acids* $\mathcal{B}$ M›
⟨*proof*⟩

lemma *isa-vmtf-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ $L \in$ *bump-heur* $\mathcal{A}$ M ⟹ $L \in$ *bump-heur* $\mathcal{B}$ M›
⟨*proof*⟩

lemma *isa-vmtf-cong'*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *bump-heur* $\mathcal{A}$ = *bump-heur* $\mathcal{B}$›
⟨*proof*⟩

lemma *option-lookup-clause-rel-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ $L \in$ *option-lookup-clause-rel* $\mathcal{A}$ ⟹ $L \in$ *option-lookup-clause-rel* $\mathcal{B}$›
⟨*proof*⟩

lemma D_0*-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ D_0 $\mathcal{A}$ = D_0 $\mathcal{B}$›
⟨*proof*⟩

lemma *phase-saving-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *phase-saving* $\mathcal{A}$ = *phase-saving* $\mathcal{B}$›
⟨*proof*⟩

lemma *cach-refinement-empty-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *cach-refinement-empty* $\mathcal{A}$ = *cach-refinement-empty* $\mathcal{B}$›
⟨*proof*⟩

lemma *vdom-m-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *vdom-m* $\mathcal{A}$ x y = *vdom-m* $\mathcal{B}$ x y›
⟨*proof*⟩

lemma *isasat-input-bounded-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *isasat-input-bounded* $\mathcal{A}$ = *isasat-input-bounded* $\mathcal{B}$›
⟨*proof*⟩

lemma *isasat-input-nempty-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *isasat-input-nempty* $\mathcal{A}$ = *isasat-input-nempty* $\mathcal{B}$›
⟨*proof*⟩

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›
⟨proof⟩

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›
⟨proof⟩

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)))›
⟨proof⟩

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⟩

lemma *arena-lit-pre-le-snat64-max*:
‹length ba ≤ snat64-max ⟹
arena-lit-pre ba a ⟹ a ≤ snat64-max›
⟨proof⟩

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)›
⟨*proof*⟩

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*⟩

lemma *length-watched-le2*:
assumes
prop-inv: ‹*correct-watching-except i j L x1*› **and**
xb-x'a: ‹(*x1a*, *x1*) ∈ *twl-st-heur''* $\mathcal{D}$*1 r lcount*› **and**
x2: ‹*x2* ∈# *all-lits-st x1*› **and** *diff*: ‹*L* ≠ *x2*›
shows ‹*length* (*watched-by x1 x2*) ≤ *r* − *MIN-HEADER-SIZE*›
⟨*proof*⟩

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* ›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*) ∈

$[\lambda\text{-}.\ True]_f\ twl\text{-}st\text{-}heur \times_r Id \to \langle\langle bool\text{-}rel\rangle option\text{-}rel\rangle nres\text{-}rel\rangle$
⟨*proof*⟩

lemma *mop-polarity-st-heur-mop-polarity-wl''*:
‹(*uncurry mop-polarity-st-heur*, *uncurry mop-polarity-wl*) ∈
$[\lambda\text{-}.\ True]_f\ twl\text{-}st\text{-}heur\text{-}up''\ \mathcal{D}\ r\ s\ K\ lcount \times_r Id \to \langle\langle bool\text{-}rel\rangle option\text{-}rel\rangle nres\text{-}rel\rangle$
⟨*proof*⟩

lemma [*simp*,*iff*]: ‹*literals-are-*$\mathcal{L}_{in}$ (*all-atms-st S*) *S* ⟷ *blits-in-*$\mathcal{L}_{in}$ *S*›
⟨*proof*⟩

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* = (λ*S*.
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* = (λ*i 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›
  ⟨proof⟩

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 = (λS. 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 = (λS.
    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›
  ⟨proof⟩

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 = (λS. 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 = (λS. 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 = (λwaste 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*
⟨*proof*⟩

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 $?a + ?b + ?c = ?a + (?b + ?c)$
$?a + ?b = ?b + ?a$
$?b + (?a + ?c) = ?a + (?b + ?c)$
$?a * ?b * ?c = ?a * (?b * ?c)$
$?a * ?b = ?b * ?a$
$?b * (?a * ?c) = ?a * (?b * ?c)$
$((?a \wedge ?b) \wedge ?c) = (?a \wedge ?b \wedge ?c)$
$(?a \wedge ?b) = (?b \wedge ?a)$
$(?b \wedge ?a \wedge ?c) = (?a \wedge ?b \wedge ?c)$
$((?a \vee ?b) \vee ?c) = (?a \vee ?b \vee ?c)$
$(?a \vee ?b) = (?b \vee ?a)$

$(?b \vee ?a \vee ?c) = (?a \vee ?b \vee ?c)$
$inf\ (inf\ ?a\ ?b)\ ?c = inf\ ?a\ (inf\ ?b\ ?c)$
$inf\ ?a\ ?b = inf\ ?b\ ?a$
$inf\ ?b\ (inf\ ?a\ ?c) = inf\ ?a\ (inf\ ?b\ ?c)$
$sup\ (sup\ ?a\ ?b)\ ?c = sup\ ?a\ (sup\ ?b\ ?c)$
$sup\ ?a\ ?b = sup\ ?b\ ?a$
$sup\ ?b\ (sup\ ?a\ ?c) = sup\ ?a\ (sup\ ?b\ ?c)$
$min\ (min\ ?a\ ?b)\ ?c = min\ ?a\ (min\ ?b\ ?c)$
$min\ ?a\ ?b = min\ ?b\ ?a$
$min\ ?b\ (min\ ?a\ ?c) = min\ ?a\ (min\ ?b\ ?c)$
$max\ (max\ ?a\ ?b)\ ?c = max\ ?a\ (max\ ?b\ ?c)$
$max\ ?a\ ?b = max\ ?b\ ?a$
$max\ ?b\ (max\ ?a\ ?c) = max\ ?a\ (max\ ?b\ ?c)$
$coprime\ ?b\ ?a = coprime\ ?a\ ?b$
$(?a\ dvd\ ?c - ?b) = (?a\ dvd\ ?b - ?c)$
$(?a\ @\ ?b)\ @\ ?c = ?a\ @\ ?b\ @\ ?c$
$(?a\ AND\ ?b)\ AND\ ?c = ?a\ AND\ ?b\ AND\ ?c$
$?a\ AND\ ?b = ?b\ AND\ ?a$
$?b\ AND\ ?a\ AND\ ?c = ?a\ AND\ ?b\ AND\ ?c$
$(?a\ OR\ ?b)\ OR\ ?c = ?a\ OR\ ?b\ OR\ ?c$
$?a\ OR\ ?b = ?b\ OR\ ?a$
$?b\ OR\ ?a\ OR\ ?c = ?a\ OR\ ?b\ OR\ ?c$
$(?a\ XOR\ ?b)\ XOR\ ?c = ?a\ XOR\ ?b\ XOR\ ?c$
$?a\ XOR\ ?b = ?b\ XOR\ ?a$
$?b\ XOR\ ?a\ XOR\ ?c = ?a\ XOR\ ?b\ XOR\ ?c$
$gcd\ (gcd\ ?a\ ?b)\ ?c = gcd\ ?a\ (gcd\ ?b\ ?c)$
$gcd\ ?a\ ?b = gcd\ ?b\ ?a$
$gcd\ ?b\ (gcd\ ?a\ ?c) = gcd\ ?a\ (gcd\ ?b\ ?c)$
$lcm\ (lcm\ ?a\ ?b)\ ?c = lcm\ ?a\ (lcm\ ?b\ ?c)$
$lcm\ ?a\ ?b = lcm\ ?b\ ?a$
$lcm\ ?b\ (lcm\ ?a\ ?c) = lcm\ ?a\ (lcm\ ?b\ ?c)$
$signed.min\ (signed.min\ ?a\ ?b)\ ?c = signed.min\ ?a\ (signed.min\ ?b\ ?c)$
$signed.min\ ?a\ ?b = signed.min\ ?b\ ?a$
$signed.min\ ?b\ (signed.min\ ?a\ ?c) = signed.min\ ?a\ (signed.min\ ?b\ ?c)$
$signed.max\ (signed.max\ ?a\ ?b)\ ?c = signed.max\ ?a\ (signed.max\ ?b\ ?c)$
$signed.max\ ?a\ ?b = signed.max\ ?b\ ?a$
$signed.max\ ?b\ (signed.max\ ?a\ ?c) = signed.max\ ?a\ (signed.max\ ?b\ ?c)$
$?a \cap\# ?b \cap\# ?c = ?a \cap\# (?b \cap\# ?c)$
$?a \cap\# ?b = ?b \cap\# ?a$
$?b \cap\# (?a \cap\# ?c) = ?a \cap\# (?b \cap\# ?c)$
$?a \cup\# ?b \cup\# ?c = ?a \cup\# (?b \cup\# ?c)$
$?a \cup\# ?b = ?b \cup\# ?a$

$?b \cup\# (?a \cup\# ?c) = ?a \cup\# (?b \cup\# ?c)$
$((?a \wedge* ?b) \wedge* ?c) = (?a \wedge* ?b \wedge* ?c)$
$(?a \wedge* ?b) = (?b \wedge* ?a)$
$(?b \wedge* ?a \wedge* ?c) = (?a \wedge* ?b \wedge* ?c)$ 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*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*))›
⟨*proof*⟩

lemma *print-trail-st-print-trail-st2-rel*:
‹(*print-trail-st*, *RETURN o print-trail-st2*) ∈ *Id* →$_f$ (⟨*unit-rel*⟩*nres-rel*)›
⟨*proof*⟩

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*›
  ⟨*proof*⟩

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 ‹$(S,T) \in$ *twl-st-heur*› **and** ‹$A \in\#$ *all-atms-st* T›
shows ‹*mop-mark-added-heur-st* A $S \leq$ *SPEC* $(\lambda c.\ (c,\ T) \in \{(U,\ V).\ (U,\ V) \in$ *twl-st-heur* $\wedge$
(*get-clauses-wl-heur* U) = *get-clauses-wl-heur* $S \wedge$
learned-clss-count U = *learned-clss-count* $S\})$›
⟨*proof*⟩

lemma *mark-added-clause-heur2-id*:
assumes ‹$(S,T) \in$ *twl-st-heur*› **and** ‹$C \in\#$ *dom-m* (*get-clauses-wl* T)›
shows ‹*mark-added-clause-heur2* S C
$\leq \Downarrow\{(U,\ V).\ (U,\ V) \in$ *twl-st-heur* $\wedge$ (*get-clauses-wl-heur* U) = *get-clauses-wl-heur* $S \wedge$
learned-clss-count U = *learned-clss-count* $S\}$ (*RETURN* T)› (**is** ‹- $\leq\Downarrow ?R$ -›)
⟨*proof*⟩

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 ‹$(S,T) \in$ *twl-st-heur*› **and** ‹$A \in\#$ *all-atms-st* T›
shows ‹*mop-is-marked-added-heur-st* S $A \leq$ *SPEC*$(\lambda c.\ (c,\ d) \in$ (*UNIV* :: (*bool* $\times$ *bool*) *set*))›
⟨*proof*⟩

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›
⟨*proof*⟩
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›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*]›
⟨*proof*⟩

lemma *aivdom-inv-dec-empty-tvdom*[*intro*!]:
‹*aivdom-inv-dec aivdom d* ⟹ *aivdom-inv-dec* (*empty-tvdom aivdom*) *d*›
⟨*proof*⟩

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*) = []›
⟨*proof*⟩

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⟩

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›
    ⟨proof⟩
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*›
⟨*proof*⟩

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*]) }›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

lemma *eo-list-set-refine*: ‹(*mop-list-set*, *mop-eo-set*) ∈ ⟨*only-some-rel*⟩*list-rel* → *Id* → *Id* → ⟨⟨*only-some-rel*⟩*list-rel*⟩*nres*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*]$_c$
⟨*proof*⟩

lemmas *eo-set-refine-aux* = *set-hnr'*[*FCOMP eo-list-set-refine*]

lemma *pure-entails-empty*: ‹*is-pure A* ⟹ *A a c* ⊢ □›
⟨*proof*⟩

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*]$_c$›
⟨*proof*⟩

end

lemma *id-Some-only-some-rel*: ‹(*id*, *Some*) ∈ *Id* → *only-some-rel*›
⟨*proof*⟩

lemma *map-some-only-some-rel-iff*: ‹(*xs*, *map Some ys*) ∈ ⟨*only-some-rel*⟩*list-rel* ⟷ *xs*=*ys*›
⟨*proof*⟩

lemma *wo-assn-conv*: ‹*wo-assn xs ys* = *eo-assn* (*map Some xs*) *ys*›
⟨*proof*⟩

lemma *to-eo-conv-refine*: ‹(*Mreturn*, *mop-to-eo-conv*) ∈ [λ-. *True*]$_c$ *wo-assn*^{d} → (*eo-assn*) [λ(*ai*) (*r*). *r*=*ai*]$_c$›
⟨*proof*⟩

lemma ‹*None* ∉ *set xs* ⟷ (∃ *ys*. *xs* = *map Some ys*)›
⟨*proof*⟩

lemma *to-wo-conv-refine*: ‹(*Mreturn*, *mop-to-wo-conv*) ∈ [λ-. *True*]$_c$ *eo-assn*^{d} → (*wo-assn*) [λ(*ai*) (*r*). *r*=*ai*]$_c$›
⟨*proof*⟩

lemma *random-access-iterator*: *random-access-iterator wo-assn eo-assn elem-assn*
Mreturn Mreturn
eo-extract-impl
wo-set-impl

⟨*proof*⟩

sublocale *random-access-iterator wo-assn eo-assn elem-assn*
Mreturn Mreturn
eo-extract-impl
wo-set-impl
⟨*proof*⟩

end

lemma *is-pureE-abs*:
assumes *is-pure P*
obtains P' **where** $P = (\lambda x\ x'.\ \uparrow(P'\ x\ x'))$
⟨*proof*⟩

lemma *al-pure-eo*: ‹*is-pure A* ⟹ *pure-eo-adapter A* (*al-assn A*) *arl-nth arl-upd*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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} → *uint32-nat-assn* ×$_a$ *sint64-nat-assn* ×$_a$ *sint64-nat-assn*›
⟨*proof*⟩

sepref-def (**in** −) *clause-score-ordering-code*
is ‹*uncurry* (*RETURN oo clause-score-ordering*)›

:: ‹(*uint32-nat-assn* ×$_a$ *sint64-nat-assn* ×$_a$ *sint64-nat-assn*)^k *$_a$ (*uint32-nat-assn* ×$_a$ *sint64-nat-assn* ×$_a$ *sint64-nat-assn*)^k →$_a$ *bool1-assn*›
⟨*proof*⟩

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} →$_a$ *bool1-assn*›
⟨*proof*⟩

interpretation *LBD*: *weak-ordering-on-lt* **where**
C = ‹*idx-cdom vs*› **and**
less = ‹*clause-score-less vs*›
⟨*proof*⟩

interpretation *LBD*: *parameterized-weak-ordering idx-cdom clause-score-less*
mop-clause-score-less
⟨*proof*⟩

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*

⟨*proof*⟩

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*
⟨*proof*⟩

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*

⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def (**in** −) *clause-size-ordering-code*
is ‹*uncurry* (*RETURN oo* (≤))›

:: ‹(*sint64-nat-assn*)^k $*_a$ (*sint64-nat-assn*)^k $\to_a$ *bool1-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

interpretation *Size-Ordering*: *weak-ordering-on-lt* **where**
C = ‹*idx-clause-cdom vs*› **and**
less = ‹*clause-size-less vs*›
⟨*proof*⟩

interpretation *Size-Ordering*: *parameterized-weak-ordering idx-clause-cdom clause-size-less*
mop-clause-size-less
⟨*proof*⟩

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*

⟨*proof*⟩

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*

⟨*proof*⟩

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*⟩

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*⟩

schematic-goal *mk-free-ghost-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE virtual-vdom-fast-assn ?fr*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *watchlist-put-binaries-first-impl*
is ‹*watchlist-put-binaries-first*›
:: ‹*watchlist-fast-assn*^{d} →$_a$ *watchlist-fast-assn*›
⟨*proof*⟩

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*)›

⟨*proof*⟩

sepref-def *NoMark-impl* [*llvm-inline*]
is [] ‹*uncurry0* (*RETURN 0*)› :: ‹*unit-assn*^{k} →$_a$ *sint-assn′ TYPE*(*3*)›
⟨*proof*⟩

sepref-register *Mark NoMark*

lemma *Mark-rel-aux*: ‹(λ*b*. *if b then 1* :: *int else* −*1*, *Mark*) ∈ *bool-rel* → *mark-rel-aux*›
⟨*proof*⟩

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

lemma *NoMark-rel-aux*: ‹(*0*, *NoMark*) ∈ *mark-rel-aux*›
⟨*proof*⟩

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

lemma *mark-rel-aux-eq*: ‹((=), (=)) ∈ *mark-rel-aux* → *mark-rel-aux* → *bool-rel*›
⟨*proof*⟩

sepref-def *mark-eq-impl* **is**
‹*uncurry* (*RETURN oo* (=))› :: ‹(*sint-assn′ TYPE*(*3*))^d *$_a$ (*sint-assn′ TYPE*(*3*))^d →$_a$ *bool1-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩
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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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›
  ⟨proof⟩

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⟩
  ⟨proof⟩


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⟩
  ⟨proof⟩


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} $\to$ *uint32-nat-assn*›
⟨*proof*⟩

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} $\to$ *uint32-nat-assn*›
⟨*proof*⟩

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} $\to$ *tri-bool-assn*›
⟨*proof*⟩

sepref-def *polarity-pol-fast*
is ‹*uncurry* (*mop-polarity-pol*)›
:: ‹*trail-pol-fast-assn*^{k} $*_a$ *unat-lit-assn*^{k} $\to_a$ *tri-bool-assn*›
⟨*proof*⟩

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} $\to$ *snat-assn*′ *TYPE*(*64*)›
⟨*proof*⟩

sepref-def *mop-isa-length-trail-fast-code*
is ‹*mop-isa-length-trail*›
:: ‹*trail-pol-fast-assn*^{k} $\to_a$ *snat-assn*′ *TYPE*(*64*)›
⟨*proof*⟩

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} $\to_a$ *trail-pol-fast-assn*›
⟨*proof*⟩

sepref-def *tl-trail-tr-fast-code*
is ‹*RETURN o tl-trailt-tr*›
:: ‹[*tl-trailt-tr-pre*]$_a$
trail-pol-fast-assn^d $\to$ *trail-pol-fast-assn*›
⟨*proof*⟩

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 $\to$ *trail-pol-fast-assn*›
⟨*proof*⟩

sepref-def *lit-of-last-trail-fast-code*

is ‹*RETURN o lit-of-last-trail-pol*›
:: ‹$[\lambda(M,\ \text{-}).\ M \neq []]_a$ *trail-pol-fast-assn*$^k \to$ *unat-lit-assn*›
⟨*proof*⟩

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*$^d \to$ *trail-pol-fast-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *get-the-propagation-reason-fast-code*
is ‹*uncurry get-the-propagation-reason-pol*›
:: ‹*trail-pol-fast-assn*$^k *_a$ *unat-lit-assn*$^k \to_a$ *snat-option-assn*$'$ *TYPE*(*64*)›
⟨*proof*⟩

sepref-def *trail-height-before-conflict-impl*
  is ‹*trail-height-before-conflict*›
  :: ‹*trail-pol-fast-assn*^{k} →$_a$ *uint32-nat-assn*›
  ⟨*proof*⟩

sepref-def *trail-zeroed-until-impl*
  is ‹*RETURN o trail-zeroed-until*›
  :: ‹*trail-pol-fast-assn*^{k} →$_a$ *sint64-nat-assn*›
  ⟨*proof*⟩

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} →$_a$ *trail-pol-fast-assn*›
  ⟨*proof*⟩

schematic-goal *mk-free-trail-pol-fast-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE trail-pol-fast-assn ?fr*›
  ⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *stop-profile*
is ‹*RETURN o IsaSAT-Profile.stop*›
:: ‹*word-assn*^{k} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-def *IsaSAT-Profile-PROPAGATE*
is ‹*uncurry0* (*RETURN IsaSAT-Profile.PROPAGATE*)›
:: ‹*unit-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

sepref-def *IsaSAT-Profile-ANALYZE*
is ‹*uncurry0* (*RETURN IsaSAT-Profile.ANALYZE*)›
:: ‹*unit-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

sepref-def *IsaSAT-Profile-GC*
is ‹*uncurry0* (*RETURN IsaSAT-Profile.GC*)›
:: ‹*unit-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

sepref-def *IsaSAT-Profile-REDUCE*
is ‹*uncurry0* (*RETURN IsaSAT-Profile.REDUCE*)›
:: ‹*unit-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

sepref-def *IsaSAT-Profile-MINIMIZATION*
is ‹*uncurry0* (*RETURN IsaSAT-Profile.MINIMIZATION*)›
:: ‹*unit-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

sepref-def *IsaSAT-Profile-INITIALISATION*
is ‹*uncurry0* (*RETURN IsaSAT-Profile.INITIALISATION*)›
:: ‹*unit-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

sepref-def *IsaSAT-Profile-PURE-LITERAL*
is ‹*uncurry0* (*RETURN IsaSAT-Profile.PURE-LITERAL*)›
:: ‹*unit-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

sepref-def *IsaSAT-Profile-BINARY-SIMP*
is ‹*uncurry0* (*RETURN IsaSAT-Profile.BINARY-SIMP*)›

:: ‹*unit-assn*^{k} $\rightarrow_a$ *word-assn*›
⟨*proof*⟩

sepref-def *IsaSAT-Profile-SUBSUMPTION*
is ‹*uncurry0* (*RETURN IsaSAT-Profile.SUBSUMPTION*)›
:: ‹*unit-assn*^{k} $\rightarrow_a$ *word-assn*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *SEEN-FAILED-impl*
is [] ‹*uncurry0* (*RETURN 1*)›
:: ‹*unit-assn*^{k} →$_a$ *minimize-status-ref-assn*›
⟨*proof*⟩

sepref-def *SEEN-UNKNOWN-impl*
is [] ‹*uncurry0* (*RETURN 0*)›
:: ‹*unit-assn*^{k} →$_a$ *minimize-status-ref-assn*›
⟨*proof*⟩

sepref-def *SEEN-REMOVABLE-impl*
is [] ‹*uncurry0* (*RETURN 2*)›
:: ‹*unit-assn*^{k} →$_a$ *minimize-status-ref-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *Some-impl*
is [] ‹*RETURN o* (λ-. *True*)›
:: ‹*bool1-assn*^{k} →$_a$ *bool1-assn*›
⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *Some-impl.refine*[*FCOMP Some-rel*]

lemma *is-Notin-rel*: ‹(λ*x*. ¬*x*, *is-NOTIN*) ∈ *option-bool-rel* → *bool-rel*›
⟨*proof*⟩

sepref-def *is-Notin-impl*
is [] ‹*RETURN o* (λ*x*. ¬*x*)›
:: ‹*bool1-assn*^{k} →$_a$ *bool1-assn*›
⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *is-Notin-impl.refine*[*FCOMP is-Notin-rel*]

lemma *NOTIN-rel*: ‹(*False*, *NOTIN*) ∈ *option-bool-rel*›
⟨*proof*⟩

sepref-def *NOTIN-impl*
is [] ‹*uncurry0* (*RETURN False*)›
:: ‹*unit-assn*^{k} →$_a$ *bool1-assn*›
⟨*proof*⟩

lemmas [*sepref-fr-rules*] = *NOTIN-impl.refine*[*FCOMP NOTIN-rel*]

definition (**in** −) *lookup-clause-rel-assn*
:: ‹*lookup-clause-rel* ⇒ *lookup-clause-assn* ⇒ *assn*›
where
‹*lookup-clause-rel-assn* ≡ (*uint32-nat-assn* ×$_a$ *array-assn option-bool-impl-assn*)›

definition (**in** −)*conflict-option-rel-assn*
:: ‹*conflict-option-rel* ⇒ *option-lookup-clause-assn* ⇒ *assn*›
where
‹*conflict-option-rel-assn* ≡ (*bool1-assn* ×$_a$ *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**

‹*ana-refinement-fast-rel* ≡ *snat-rel′ TYPE*(*64*) ×$_r$ *unat-rel′ TYPE*(*32*) ×$_r$ *bool1-rel*›

abbreviation (**in** −)*ana-refinement-fast-assn* **where**
‹*ana-refinement-fast-assn* ≡ *sint64-nat-assn* ×$_a$ *uint32-nat-assn* ×$_a$ *bool1-assn*›

lemma *ana-refinement-fast-assn-def*:
‹*ana-refinement-fast-assn* = *pure ana-refinement-fast-rel*›
⟨*proof*⟩

abbreviation (**in** −)*analyse-refinement-fast-assn* **where**
‹*analyse-refinement-fast-assn* ≡
arl64-assn ana-refinement-fast-assn›

lemma *lookup-clause-assn-is-None-alt-def*:
‹*RETURN o lookup-clause-assn-is-None* = (λ(*b*, -, -). *RETURN b*)›
⟨*proof*⟩

sepref-def *lookup-clause-assn-is-None-impl*
is ‹*RETURN o lookup-clause-assn-is-None*›
:: ‹*conflict-option-rel-assn*^{k} →$_a$ *bool1-assn*›
⟨*proof*⟩

lemma *size-lookup-conflict-alt-def*:
‹*RETURN o size-lookup-conflict* = (λ(-, *b*, -). *RETURN b*)›
⟨*proof*⟩

sepref-def *size-lookup-conflict-impl*
is ‹*RETURN o size-lookup-conflict*›
:: ‹*conflict-option-rel-assn*^{k} →$_a$ *uint32-nat-assn*›
⟨*proof*⟩

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} → *bool1-assn*›
⟨*proof*⟩

lemma *lookup-clause-assn-is-empty-alt-def*:
‹*lookup-clause-assn-is-empty* = (λ*S*. *size-lookup-conflict S* = *0*)›
⟨*proof*⟩

sepref-def *lookup-clause-assn-is-empty-impl*
is ‹*RETURN o lookup-clause-assn-is-empty*›
:: ‹*conflict-option-rel-assn*^{k} →$_a$ *bool1-assn*›
⟨*proof*⟩

definition *the-lookup-conflict* :: ‹*conflict-option-rel* ⇒ -› **where**
‹*the-lookup-conflict* = *snd*›

lemma *the-lookup-conflict-alt-def*:
‹*RETURN o the-lookup-conflict* = (λ(-, (*n*, *xs*)). *RETURN* (*n*, *xs*))›

⟨*proof*⟩

sepref-def *the-lookup-conflict-impl*
is ‹*RETURN o the-lookup-conflict*›
:: ‹*conflict-option-rel-assn*^{d} →$_a$ *lookup-clause-rel-assn*›
⟨*proof*⟩

definition *Some-lookup-conflict* :: ‹- ⇒ *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*))›
⟨*proof*⟩

sepref-def *Some-lookup-conflict-impl*
is ‹*RETURN o Some-lookup-conflict*›
:: ‹*lookup-clause-rel-assn*^{d} →$_a$ *conflict-option-rel-assn*›
⟨*proof*⟩
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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)]))›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)
})›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *atm-in-conflict-code*
is ‹*uncurry* (*RETURN oo atm-in-conflict-lookup*)›
:: ‹[*uncurry atm-in-conflict-lookup-pre*]$_a$

$\mathit{atom\text{-}assn}^k *_a \mathit{lookup\text{-}clause\text{-}rel\text{-}assn}^k \to \mathit{bool1\text{-}assn}\rangle$
⟨*proof*⟩

sepref-def *conflict-min-cach-l-code*
is ‹*uncurry* (*RETURN oo conflict-min-cach-l*)›
:: ‹$[\mathit{conflict\text{-}min\text{-}cach\text{-}l\text{-}pre}]_a$ $\mathit{cach\text{-}refinement\text{-}l\text{-}assn}^k *_a \mathit{atom\text{-}assn}^k \to \mathit{minimize\text{-}status\text{-}assn}$›
⟨*proof*⟩

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*)
})›
⟨*proof*⟩

lemma *le-unat32-max-div2-le-unat32-max*: ‹*a2'* ≤ *Suc* (*unat32-max div 2*) ⟹ *a2'* < *unat32-max*›
⟨*proof*⟩

sepref-def *conflict-min-cach-set-failed-l-code*
is ‹*uncurry conflict-min-cach-set-failed-l*›
:: ‹$\mathit{cach\text{-}refinement\text{-}l\text{-}assn}^d *_a \mathit{atom\text{-}assn}^k \to_a \mathit{cach\text{-}refinement\text{-}l\text{-}assn}$›
⟨*proof*⟩

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*)
})›
⟨*proof*⟩

sepref-def *conflict-min-cach-set-removable-l-code*
is ‹*uncurry conflict-min-cach-set-removable-l*›
:: ‹$\mathit{cach\text{-}refinement\text{-}l\text{-}assn}^d *_a \mathit{atom\text{-}assn}^k \to_a \mathit{cach\text{-}refinement\text{-}l\text{-}assn}$›
⟨*proof*⟩

lemma *lookup-conflict-size-impl-alt-def*:
‹*RETURN o* (λ(*n*, *xs*). *n*) = (λ(*n*, *xs*). *RETURN n*)›
⟨*proof*⟩

lemma *single-replicate*: ‹[*C*] = *op-list-append* [] *C*›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *isa-mark-failed-lits-stackI*:
assumes
‹*length ba* ≤ *Suc* (*unat32-max div 2*)› **and**
‹*a1'* < *length ba*›
shows ‹*Suc a1'* ≤ *unat32-max*›
⟨*proof*⟩

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›
⟨*proof*⟩

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*›
⟨*proof*⟩

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
→ *sint64-nat-assn* ×$_a$ *sint64-nat-assn* ×$_a$ *sint64-nat-assn* ×$_a$ *sint64-nat-assn*›
⟨*proof*⟩

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} →
ana-refinement-fast-assn›
⟨*proof*⟩

lemma *isa-lit-redundant-rec-wl-lookupI*:

assumes
‹*length ba* ≤ *Suc* (*unat32-max div 2*)›
shows ‹*length ba* < *unat32-max*›
⟨*proof*⟩

lemma *arena-lit-pre-le*: ‹
arena-lit-pre a i ⟹ *length a* ≤ *snat64-max* ⟹ *i* ≤ *snat64-max*›
⟨*proof*⟩

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*
}›
⟨*proof*⟩

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} →
cach-refinement-l-assn ×$_a$ *analyse-refinement-fast-assn* ×$_a$ *bool1-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *unat32-max-ge0*: ‹0 < *unat32-max*› ⟨*proof*⟩

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 ×$_a$ *analyse-refinement-fast-assn* ×$_a$ *bool1-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)
})›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›

⟨*proof*⟩

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*›
⟨*proof*⟩

schematic-goal *mk-free-cach-refinement-l-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE cach-refinement-l-assn ?fr*›
⟨*proof*⟩

schematic-goal *mk-free-trail-pol-fast-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE conflict-option-rel-assn ?fr*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *VMTF-stamp-impl*
is [] ‹*RETURN o* (λ(*a*,*b*,*c*). *a*)›
:: ‹*vmtf-node2-assn*^{k} →$_a$ *uint64-nat-assn*›
⟨*proof*⟩

sepref-def *VMTF-get-prev-impl*
is [] ‹*RETURN o* (λ(*a*,*b*,*c*). *b*)›
:: ‹*vmtf-node2-assn*^{k} →$_a$ *atom.option-assn*›
⟨*proof*⟩

sepref-def *VMTF-get-next-impl*
is [] ‹*RETURN o* (λ(*a*,*b*,*c*). *c*)›
:: ‹*vmtf-node2-assn*^{k} →$_a$ *atom.option-assn*›
⟨*proof*⟩

lemma *workaround-hrcomp-id-norm*[*fcomp-norm-unfold*]: ‹*hr-comp R* (⟨*nat-rel*⟩*option-rel*) = *R*› ⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *length-avdom-aivdom-impl*
is ‹*RETURN o length-avdom-aivdom-int*›
:: ‹*aivdom-int-assn*^{k} →$_a$ *sint64-nat-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *length-ivdom-aivdom-impl*
is ‹*RETURN o length-ivdom-aivdom-int*›
:: ‹*aivdom-int-assn*^{k} →$_a$ *sint64-nat-assn*›
⟨*proof*⟩

sepref-def *length-tvdom-aivdom-impl*
is ‹*RETURN o length-tvdom-aivdom-int*›
:: ‹*aivdom-int-assn*^{k} →$_a$ *sint64-nat-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *empty-aivdom-impl*
is ‹*RETURN o empty-aivdom-int*›
:: ‹*aivdom-int-assn*^{d} →$_a$ *aivdom-int-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *empty-tvdom-impl*
  is ‹*RETURN o empty-tvdom-int*›
  :: ‹*aivdom-int-assn*^{d} →$_a$ *aivdom-int-assn*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

lemma *aivdom-assn-alt-def*:
  ‹*aivdom-assn* = *hr-comp aivdom-int-assn* (⟨*aivdom-int-rel*⟩*code-hider-rel*)›
  ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*))›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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
⟨*proof*⟩
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)]
⟨*proof*⟩

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'*)
⟨*proof*⟩

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*
⟨*proof*⟩

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
}
⟨*proof*⟩

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
}
⟨*proof*⟩

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*)
}
⟨*proof*⟩
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*)
}

⟨*proof*⟩

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')
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *hn-case-tuple4'*[*sepref-comb-rules*]:
assumes *FR*: ‹Γ ⊢ *hn-ctxt* (*tuple4-assn P1 P2 P3 P4*) *p' p* ∗∗ Γ*1*›
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*
∧∗ Γ*1*)
(*f a1 a2 a3 a4*)
(Γ*2 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'*.
Γ*2 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* ∗∗ Γ*1'*›
shows ‹*hn-refine* Γ (*case-tuple4 f p*) (*hn-ctxt* (*tuple4-assn P1' P2' P3' P4'*) *p' p* ∗∗ Γ*1'*)
R (*case-tuple4 CP p*) (*case-tuple4*$(λ$_2$*a1 a2 a3 a4* . *f' a1 a2 a3 a4*)$*p'*)› (**is** ‹*?G* Γ›)
⟨*proof*⟩

apply1 (*rule hn-refine-cons-pre*[*OF FR*])
apply1 (*cases p*; *cases p'*; *simp add*: *tuple4-assn-conv*[*THEN tuple4-assn-ctxt*])
⟨*proof*⟩
applyS (*simp add*: *hn-ctxt-def*)
applyS *simp* ⟨*proof*⟩

lemma *case-tuple4-arity*[*sepref-monadify-arity*]:
‹*case-tuple4* ≡ λ$_2$*fp p*. *SP case-tuple4*$(λ$_2$*a b*. *fp*$*a*$*b*)$*p*›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *ho-tuple4-move*[*sepref-preproc*]: ‹*case-tuple4* (λ*a1 a2 a3 a4* *x*. *f x a1 a2 a3 a4*) =
(λ*p x*. *case-tuple4* (*f x*) *p*)›
⟨*proof*⟩

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*))
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *RETURN-case-tuple4-inverse*: ‹*RETURN*
(*let* - = *M*
in ff) =
(*do* {- ← *mop-free M*;
RETURN (*ff*)})›
⟨*proof*⟩

sepref-definition *update-a-code*
is ‹*uncurry* (*RETURN oo update-a*)›
:: ‹*a-assn*^{d} ∗$_a$ *tuple4-int-assn*^{d} →$_a$ *tuple4-int-assn*›
⟨*proof*⟩

sepref-definition *update-b-code*
is ‹*uncurry* (*RETURN oo update-b*)›
:: ‹*b-assn*^{d} ∗$_a$ *tuple4-int-assn*^{d} →$_a$ *tuple4-int-assn*›
⟨*proof*⟩

sepref-definition *update-c-code*
is ‹*uncurry* (*RETURN oo update-c*)›
:: ‹*c-assn*^{d} ∗$_a$ *tuple4-int-assn*^{d} →$_a$ *tuple4-int-assn*›
⟨*proof*⟩

sepref-definition *update-d-code*
is ‹*uncurry* (*RETURN oo update-d*)›
:: ‹*d-assn*^{d} ∗$_a$ *tuple4-int-assn*^{d} →$_a$ *tuple4-int-assn*›
⟨*proof*⟩

lemma *RETURN-case-tuple4-invers*: ‹(*RETURN* ∘∘ *case-tuple4*)
(λ*x1 x2 x3 x4*.
ff x1 x2 x3 x4) =
case-tuple4
(λ*x1 x2 x3 x4*.

RETURN (ff x1 x2 x3 x4))›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-definition *remove-b-code*
is ‹*RETURN o remove-b*›
:: ‹ *tuple4-int-assn*^{d} →$_a$ *b-assn* ×$_a$ *tuple4-int-assn*›
⟨*proof*⟩

sepref-definition *remove-c-code*
is ‹*RETURN o remove-c*›
:: ‹ *tuple4-int-assn*^{d} →$_a$ *c-assn* ×$_a$ *tuple4-int-assn*›
⟨*proof*⟩

sepref-definition *remove-d-code*
is ‹*RETURN o remove-d*›
:: ‹ *tuple4-int-assn*^{d} →$_a$ *d-assn* ×$_a$ *tuple4-int-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*
})›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *mop-prio-insert-unchanged-impl*
is ‹*uncurry ACIDS.mop-prio-insert-unchanged*›
:: ‹*atom-assn*^{k} ∗$_a$ *acids-assn*^{d} →$_a$ *acids-assn*›
⟨*proof*⟩

sepref-def *acids-tl-impl*
is ‹*uncurry acids-tl*›
:: ‹*atom-assn*^{k} ∗$_a$ *acids-assn2*^{d} →$_a$ *acids-assn2*›
⟨*proof*⟩

sepref-def *acids-pop-min-impl*
is *acids-pop-min*
:: ‹*acids-assn2*^{d} →$_a$ *atom-assn* ×$_a$ *acids-assn2*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *acids-push-literal-impl*
is ‹*uncurry acids-push-literal*›
:: ‹*atom-assn*^{k} ∗$_a$ *acids-assn2*^{d} →$_a$ *acids-assn2*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*⟩

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*›
  ⟨*proof*⟩

schematic-goal *free-acids-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE acids-assn ?a*›
  ⟨*proof*⟩

schematic-goal *free-acids-assn2*[*sepref-frame-free-rules*]: ‹*MK-FREE acids-assn2 ?a*›
  ⟨*proof*⟩

sepref-def *free-acids*
  is ‹*mop-free*›
  :: ‹*acids-assn2*^{d} →$_a$ *unit-assn*›
  ⟨*proof*⟩

lemma *free-acids-assn2'*: ‹*MK-FREE acids-assn2 free-acids*›
  ⟨*proof*⟩

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} →$_a$ *atom-assn*›
⟨*proof*⟩

sepref-def *bottom-vmtf-code*
is ‹*uncurry0* (*RETURN bottom-vmtf*)›
:: ‹*unit-assn*^{k} →$_a$ *vmtf-assn*›
⟨*proof*⟩

schematic-goal *free-vmtf-remove-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE vmtf-assn ?a*›
⟨*proof*⟩

sepref-def *free-vmtf-remove*
is ‹*mop-free*›
:: ‹*vmtf-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-def *bottom-focused*
is ‹*uncurry0* (*RETURN False*)›
:: ‹*unit-assn*^{k} →$_a$ *bool1-assn*›
⟨*proof*⟩
sepref-def *free-focused*
is ‹*mop-free*›
:: ‹*bool1-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩
sepref-def *free-atms-hash-code*
is ‹*mop-free*›
:: ‹*distinct-atoms-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

lemma *free-vmtf-assn-assn2*: ‹*MK-FREE vmtf-assn free-vmtf-remove*›
⟨*proof*⟩

schematic-goal *free-focused-assn*: ‹*MK-FREE bool1-assn ?a*›
⟨*proof*⟩
schematic-goal *free-distinct-atoms-assn*: ‹*MK-FREE distinct-atoms-assn ?a*›
⟨*proof*⟩

lemma *free-focused-assn2*: ‹*MK-FREE bool1-assn free-focused*›
⟨*proof*⟩

lemma *free-distinct-atoms-assn2*: ‹*MK-FREE* (*distinct-atoms-assn*) *free-atms-hash-code*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)})›
⟨*proof*⟩

sepref-def *switch-bump-heur-code*
is ‹*RETURN o switch-bump-heur*›
:: ‹*heuristic-bump-assn*^{d} →$_a$ *heuristic-bump-assn*›
⟨*proof*⟩

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
⟨*proof*⟩
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*)]
⟨*proof*⟩

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*)]
⟨*proof*⟩

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')
⟨*proof*⟩

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*
⟨*proof*⟩

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
}
⟨*proof*⟩

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
}
⟨*proof*⟩

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*)
}
⟨*proof*⟩

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*)
}
⟨*proof*⟩

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'*)
⟨*proof*⟩
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*›
⟨*proof*⟩

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*: $\bigwedge a1\ a2\ a3\ a4\ a5\ a6\ a7\ a8\ a9\ a10\ a11\ a12\ a13\ a14\ a15\ a16\ a17\ a1'\ a2'\ a3'\ a4'\ a5'\ a6'\ a7'\ a8'\ a9'\ a10'\ a11'\ a12'\ a13'\ a14'\ a15'\ a16'\ a17'.$
$[\![p'=Tuple17\ a1'\ a2'\ a3'\ a4'\ a5'\ a6'\ a7'\ a8'\ a9'\ a10'\ a11'\ a12'\ a13'\ a14'\ a15'\ a16'\ a17']\!]$
$\Longrightarrow$ *hn-refine* (*hn-ctxt P1 a1' a1* $\wedge*$ *hn-ctxt P2 a2' a2* $\wedge*$ *hn-ctxt P3 a3' a3* $\wedge*$ *hn-ctxt P4 a4' a4* $\wedge*$
hn-ctxt P5 a5' a5 $\wedge*$ *hn-ctxt P6 a6' a6* $\wedge*$ *hn-ctxt P7 a7' a7* $\wedge*$ *hn-ctxt P8 a8' a8* $\wedge*$
hn-ctxt P9 a9' a9 $\wedge*$ *hn-ctxt P10 a10' a10* $\wedge*$ *hn-ctxt P11 a11' a11* $\wedge*$ *hn-ctxt P12 a12' a12* $\wedge*$
hn-ctxt P13 a13' a13 $\wedge*$ *hn-ctxt P14 a14' a14* $\wedge*$ *hn-ctxt P15 a15' a15* $\wedge*$ *hn-ctxt P16 a16' a16* $\wedge*$ *hn-ctxt P17 a17' a17* $\wedge*$ $\Gamma 1$)
(*f a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15 a16 a17*)
($\Gamma 2$ *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*: ‹$\bigwedge a1\ a2\ a3\ a4\ a5\ a6\ a7\ a8\ a9\ a10\ a11\ a12\ a13\ a14\ a15\ a16\ a17\ a1'\ a2'\ a3'\ a4'\ a5'\ a6'\ a7'\ a8'\ a9'\ a10'\ a11'\ a12'\ a13'\ a14'\ a15'\ a16'\ a17'.$
$\Gamma 2$ *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'* $\vdash$
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 $**$ $\Gamma 1'$›
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* $**$ $\Gamma 1'$)
R (*case-tuple17 CP p*) (*case-tuple17*$($\lambda_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›)
⟨*proof*⟩
apply1 (*rule hn-refine-cons-pre*[*OF FR*])
apply1 (*cases p*; *cases p'*; *simp add*: *tuple17-assn-conv*[*THEN tuple17-assn-ctxt*])
⟨*proof*⟩
applyS (*simp add*: *hn-ctxt-def*)
applyS *simp* ⟨*proof*⟩

lemma *case-tuple17-arity*[*sepref-monadify-arity*]:
‹*case-tuple17* $\equiv \lambda_2 fp\ p.$ *SP case-tuple17*$($\lambda_2 a\ b.$ *fp*$a$$b$)$p›
⟨*proof*⟩

lemma *case-tuple17-comb*[*sepref-monadify-comb*]:
‹$\bigwedge fp\ p.$ *case-tuple17*$fp$$p \equiv$ *Refine-Basic.bind*$(*EVAL*$p$)$($\lambda_2 p.$ (*SP case-tuple17*$fp$$p$))›
⟨*proof*⟩

lemma *case-tuple17-plain-comb*[*sepref-monadify-comb*]:
EVAL$($*case-tuple17*$($\lambda_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$) $\equiv$
Refine-Basic.bind$(*EVAL*$p$)$($\lambda_2 p.$ *case-tuple17*$($\lambda_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$)
⟨*proof*⟩

lemma *ho-tuple17-move*[*sepref-preproc*]: ‹*case-tuple17* ($\lambda a1\ a2\ a3\ a4\ a5\ a6\ a7\ a8\ a9\ a10\ a11\ a12\ a13\ a14\ a15\ a16\ a17\ x.\ f\ x\ a1\ a2\ a3\ a4\ a5\ a6\ a7\ a8\ a9\ a10\ a11\ a12\ a13\ a14\ a15\ a16\ a17$) =
($\lambda p\ x.$ *case-tuple17* (*f x*) *p*)›
⟨*proof*⟩

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*))
⟨*proof*⟩

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-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} $*_a$
q-assn^d
$\to_a$ *isasat-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *RETURN-case-tuple17-inverse*: ‹*RETURN*
(*let* - = *M*
in ff) =
(*do* {- ← *mop-free M*;
RETURN (*ff*)})›
⟨*proof*⟩

sepref-def *update-a-code*
is ‹*uncurry* (*RETURN oo update-a*)›
:: ‹*a-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-b-code*
is ‹*uncurry* (*RETURN oo update-b*)›
:: ‹*b-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-c-code*
is ‹*uncurry* (*RETURN oo update-c*)›
:: ‹*c-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-d-code*
is ‹*uncurry* (*RETURN oo update-d*)›
:: ‹*d-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-e-code*
is ‹*uncurry* (*RETURN oo update-e*)›
:: ‹*e-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-f-code*
is ‹*uncurry* (*RETURN oo update-f*)›
:: ‹*f-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-g-code*
is ‹*uncurry* (*RETURN oo update-g*)›
:: ‹*g-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-h-code*
is ‹*uncurry* (*RETURN oo update-h*)›
:: ‹*h-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-i-code*
is ‹*uncurry* (*RETURN oo update-i*)›
:: ‹*i-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-j-code*
is ‹*uncurry* (*RETURN oo update-j*)›
:: ‹*j-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-k-code*
is ‹*uncurry* (*RETURN oo update-k*)›
:: ‹*k-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-l-code*

is ‹*uncurry* (*RETURN oo update-l*)›
:: ‹*l-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-m-code*
is ‹*uncurry* (*RETURN oo update-m*)›
:: ‹*m-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-n-code*
is ‹*uncurry* (*RETURN oo update-n*)›
:: ‹*n-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-o-code*
is ‹*uncurry* (*RETURN oo update-o*)›
:: ‹*o-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-p-code*
is ‹*uncurry* (*RETURN oo update-p*)›
:: ‹*p-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

sepref-def *update-q-code*
is ‹*uncurry* (*RETURN oo update-q*)›
:: ‹*q-assn*^{d} $*_a$ *isasat-assn*^{d} $\rightarrow_a$ *isasat-assn*›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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-assn*^{d} →$_a$ *a-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-b-code*
is ‹*RETURN o remove-b*›
:: ‹ *isasat-assn*^{d} →$_a$ *b-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-c-code*
is ‹*RETURN o remove-c*›
:: ‹ *isasat-assn*^{d} →$_a$ *c-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-d-code*
is ‹*RETURN o remove-d*›
:: ‹ *isasat-assn*^{d} →$_a$ *d-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-e-code*
is ‹*RETURN o remove-e*›
:: ‹ *isasat-assn*^{d} →$_a$ *e-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-f-code*
is ‹*RETURN o remove-f*›
:: ‹ *isasat-assn*^{d} →$_a$ *f-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-g-code*
is ‹*RETURN o remove-g*›
:: ‹ *isasat-assn*^{d} →$_a$ *g-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-h-code*
is ‹*RETURN o remove-h*›
:: ‹ *isasat-assn*^{d} →$_a$ *h-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-i-code*
is ‹*RETURN o remove-i*›
:: ‹ *isasat-assn*^{d} →$_a$ *i-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-j-code*
is ‹*RETURN o remove-j*›
:: ‹ *isasat-assn*^{d} →$_a$ *j-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-k-code*
is ‹*RETURN o remove-k*›
:: ‹ *isasat-assn*^{d} →$_a$ *k-assn* ×$_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-l-code*

is ‹*RETURN o remove-l*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *l-assn* $\times_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-m-code*
is ‹*RETURN o remove-m*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *m-assn* $\times_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-n-code*
is ‹*RETURN o remove-n*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *n-assn* $\times_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-o-code*
is ‹*RETURN o remove-o*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *o-assn* $\times_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-p-code*
is ‹*RETURN o remove-p*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *p-assn* $\times_a$ *isasat-assn*›
⟨*proof*⟩

sepref-definition *remove-q-code*
is ‹*RETURN o remove-q*›
:: ‹ *isasat-assn*^{d} $\rightarrow_a$ *q-assn* $\times_a$ *isasat-assn*›
⟨*proof*⟩

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)
  }›
  ⟨proof⟩

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)
  }›
  ⟨proof⟩

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*›
⟨*proof*⟩

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-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} *$*_a$ *q-assn*^{k} → *x-assn*›
notes [[*sepref-register-adhoc read-all*]]
begin
sepref-definition *read-all-code-tmp*
is *read-all-st*
:: ‹[*case-tuple17 P*]$_a$ *isasat-assn*^{k} → *x-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

definition *bottom-atom* **where**
‹*bottom-atom* = *0*›
lemma [*sepref-fr-rules*]: ‹(*uncurry0* (*Mreturn 0*), *uncurry0* (*RETURN bottom-atom*)) ∈ *unit-assn*^{k} →$_a$ *atom-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›

⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

schematic-goal *free-trail-pol-fast-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE trail-pol-fast-assn ?a*›
⟨*proof*⟩

sepref-def *free-trail-pol-fast*
is ‹*mop-free*›
:: ‹*trail-pol-fast-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

lemma *free-trail-pol-fast-assn2*: ‹*MK-FREE trail-pol-fast-assn free-trail-pol-fast*›
⟨*proof*⟩

schematic-goal *free-arena-fast-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE arena-fast-assn ?a*›
⟨*proof*⟩

sepref-def *free-arena-fast*
  is ‹*mop-free*›
  :: ‹*arena-fast-assn*^{d} →$_a$ *unit-assn*›
  ⟨*proof*⟩

lemma *free-arena-fast-assn2*: ‹*MK-FREE arena-fast-assn free-arena-fast*›
  ⟨*proof*⟩

schematic-goal *free-conflict-option-rel-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE conflict-option-rel-assn ?a*›
  ⟨*proof*⟩

sepref-def *free-conflict-option-rel*
  is ‹*mop-free*›
  :: ‹*conflict-option-rel-assn*^{d} →$_a$ *unit-assn*›
  ⟨*proof*⟩

lemma *free-conflict-option-rel-assn2*: ‹*MK-FREE conflict-option-rel-assn free-conflict-option-rel*›
  ⟨*proof*⟩

schematic-goal *free-sint64-nat-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE sint64-nat-assn ?a*›
  ⟨*proof*⟩

sepref-def *free-sint64-nat*
  is ‹*mop-free*›
  :: ‹*sint64-nat-assn*^{d} →$_a$ *unit-assn*›
  ⟨*proof*⟩

lemma *free-sint64-nat-assn-assn2*: ‹*MK-FREE sint64-nat-assn free-sint64-nat*›
  ⟨*proof*⟩

schematic-goal *free-watchlist-fast-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE watchlist-fast-assn ?a*›
  ⟨*proof*⟩

sepref-def *free-watchlist-fast*
  is ‹*mop-free*›
  :: ‹*watchlist-fast-assn*^{d} →$_a$ *unit-assn*›
  ⟨*proof*⟩

lemma *free-watchlist-fast-assn2*: ‹*MK-FREE watchlist-fast-assn free-watchlist-fast*›
  ⟨*proof*⟩

schematic-goal *free-heuristic-bump-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE heuristic-bump-assn ?a*›
  ⟨*proof*⟩

sepref-def *free-vmtf-remove*
  is ‹*mop-free*›
  :: ‹*heuristic-bump-assn*^{d} →$_a$ *unit-assn*›
  ⟨*proof*⟩

lemma *free-heuristic-bump-assn2*: ‹*MK-FREE heuristic-bump-assn free-vmtf-remove*›
  ⟨*proof*⟩

schematic-goal *free-uint32-nat-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE uint32-nat-assn ?a*›

⟨*proof*⟩

sepref-def *free-uint32-nat*
is ‹*mop-free*›
:: ‹*uint32-nat-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

lemma *free-uint32-nat-assn2*: ‹*MK-FREE uint32-nat-assn free-uint32-nat*›
⟨*proof*⟩

schematic-goal *free-cach-refinement-l-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE cach-refinement-l-assn ?a*›
⟨*proof*⟩

sepref-def *free-cach-refinement-l*
is ‹*mop-free*›
:: ‹*cach-refinement-l-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

lemma *free-cach-refinement-l-assn2*: ‹*MK-FREE cach-refinement-l-assn free-cach-refinement-l*›
⟨*proof*⟩

schematic-goal *free-lbd-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE lbd-assn ?a*›
⟨*proof*⟩

sepref-def *free-lbd*
is ‹*mop-free*›
:: ‹*lbd-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

lemma *free-lbd-assn2*: ‹*MK-FREE lbd-assn free-lbd*›
⟨*proof*⟩

schematic-goal *free-outl-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE out-learned-assn ?a*›
⟨*proof*⟩

sepref-def *free-outl*
is ‹*mop-free*›
:: ‹*out-learned-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

lemma *free-outl-assn2*: ‹*MK-FREE out-learned-assn free-outl*›
⟨*proof*⟩

schematic-goal *free-heur-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE heuristic-assn ?a*›
⟨*proof*⟩

sepref-def *free-heur*
is ‹*mop-free*›
:: ‹*heuristic-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

lemma *free-heur-assn2*: ‹*MK-FREE heuristic-assn free-heur*›
⟨*proof*⟩

schematic-goal *free-isasat-stats-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE isasat-stats-assn ?a*›

⟨*proof*⟩

sepref-def *free-stats*
is ‹*mop-free*›
:: ‹*isasat-stats-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

lemma *free-isasat-stats-assn2*: ‹*MK-FREE isasat-stats-assn free-stats*›
⟨*proof*⟩

schematic-goal *free-vdom-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE aivdom-assn ?a*›
⟨*proof*⟩

sepref-def *free-vdom*
is ‹*mop-free*›
:: ‹*aivdom-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

lemma *free-vdom-assn2*: ‹*MK-FREE aivdom-assn free-vdom*›
⟨*proof*⟩

schematic-goal *free-lcount-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE lcount-assn ?a*›
⟨*proof*⟩

sepref-def *free-lcount*
is ‹*mop-free*›
:: ‹*lcount-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

lemma *free-lcount-assn2*: ‹*MK-FREE lcount-assn free-lcount*›
⟨*proof*⟩

schematic-goal *free-opts-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE opts-assn ?a*›
⟨*proof*⟩

sepref-def *free-opts*
is ‹*mop-free*›
:: ‹*opts-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

lemma *free-opts-assn2*: ‹*MK-FREE opts-assn free-opts*›
⟨*proof*⟩

schematic-goal *free-old-arena-fast-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE arena-fast-assn ?a*›
⟨*proof*⟩

sepref-def *free-old-arena-fast*
is ‹*mop-free*›
:: ‹*arena-fast-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

lemma *free-old-arena-fast-assn2*: ‹*MK-FREE arena-fast-assn free-old-arena-fast*›
⟨*proof*⟩

sepref-def *free-occs-fast*
is ‹*mop-free*›

:: ‹*occs-assn*^{d} $\rightarrow_a$ *unit-assn*›
⟨*proof*⟩

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} $\rightarrow_a$ *occs-assn*›
⟨*proof*⟩

schematic-goal *free-occs-fast-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE occs-assn ?a*›
⟨*proof*⟩

lemma *free-occs-fast-assn2*: ‹*MK-FREE occs-assn free-occs-fast*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma [*llvm-pre-simp*]:
‹(*Mreturn* $\circ_{17}$ *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*)›
⟨*proof*⟩

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 $*_a$ $A \to b$›
⟨*proof*⟩

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 ‹$(f\ C,\ f'\ C') \in [P\ C']_a\ A \to b$›

⟨*proof*⟩

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 ‹$(uncurry\ f,\ uncurry\ f') \in [uncurry\ P]_a\ A *_a (pure\ R)^k \to b$›
⟨*proof*⟩

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 ‹$(\lambda S.\ f\ S\ C,\ \lambda S.\ f'\ S\ C') \in [\lambda S.\ P\ \ S\ C']_a\ A \to b$›
⟨*proof*⟩

lemma *remove-pure-parameter2'*:
assumes ‹$(uncurry\ (\lambda S\ C.\ f\ C\ S),\ uncurry\ (\lambda S\ C.\ f'\ C\ S)) \in [uncurry\ P]_a\ A *_a (pure\ R)^k \to b$›
‹$(C,\ C') \in R$›
shows ‹$(f\ C,\ f'\ C') \in [\lambda S.\ P\ S\ C']_a\ A \to b$›
⟨*proof*⟩

lemma *remove-pure-parameter2-twoargs*:
assumes ‹$(uncurry2\ f,\ uncurry2\ f') \in [uncurry2\ P]_a\ A *_a (pure\ R)^k *_a (pure\ R')^k \to b$› ‹$(C,\ C') \in R$› ‹$(D,D') \in R'$›
shows ‹$(\lambda S.\ f\ S\ C\ D,\ \lambda S.\ f'\ S\ C'\ D') \in [\lambda S.\ P\ \ S\ C'\ D']_a\ A \to b$›
⟨*proof*⟩

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 ‹$(f,\ f') \in [P]_a\ A \to B$›
shows ‹$(uncurry\ (\lambda M\text{ -}.\ f\ M),\ uncurry\ (\lambda M\text{ -}.\ f'\ M)) \in [uncurry\ (\lambda M\text{ -}.\ P\ M)]_a\ A *_a X^k \to B$›
⟨*proof*⟩
lemma *hn-refine-frame''*: *hn-refine* Γ c Γ' R CP $m \Longrightarrow$ *hn-refine* (F**Γ) c (F**Γ') R CP m
⟨*proof*⟩

lemma *hn-refine-frame-mid''*: *hn-refine* (F**G) c (F'**G') R CP $m \Longrightarrow$ *hn-refine* (F**Γ**G) c (F'**Γ**G') R CP m
⟨*proof*⟩

lemma *remove-component-left*:
assumes ‹$(f,\ f') \in [P]_a\ A \to B$›
shows ‹$(uncurry\ (\lambda\text{-}\ M.\ f\ M),\ uncurry\ (\lambda\text{-}\ M.\ f'\ M)) \in [uncurry\ (\lambda\text{-}\ M.\ P\ M)]_a\ \ X^k *_a A \to B$›
⟨*proof*⟩

lemma *remove-component-middle*:
assumes ‹$(f,\ f') \in [P]_a\ A *_a B \to C$›
shows ‹$(uncurry2\ (\lambda M\text{ -}\ N.\ f\ (M,\ N)),\ uncurry2\ (\lambda M\text{ -}\ N.\ f'\ (M,\ N))) \in [uncurry2\ (\lambda M\text{ -}\ N.\ P$

$(M,\ N))]_a\ \ A *_a X^k *_a B \to C\rangle$
⟨*proof*⟩

lemma (in −**)** *hfref-cong*: $\langle(a,\ b) \in [P]_a\ A \to B \Longrightarrow a = a' \Longrightarrow b = b' \Longrightarrow P = P' \Longrightarrow (a', b') \in [P']_a\ A \to B\rangle$
⟨*proof*⟩

lemma *split-snd-pure-arg*:
assumes $\langle(uncurry\ (\lambda N\ C.\ f\ C\ N),\ uncurry\ (\lambda N\ C'.\ \ f'\ C'\ N))$
$\in [uncurry\ (\lambda S\ C.\ P\ S\ C)]_a\ K^k *_a (pure\ (R \times_f R'))^k \to x\text{-}assn\rangle$
shows $\langle(uncurry2\ (\lambda N\ C\ D.\ f\ (C,D)\ N),\ uncurry2\ (\lambda N\ C'\ D'.\ \ f'\ (C',D')\ N))$
$\in [uncurry2\ (\lambda S\ C\ D.\ P\ S\ (C,\ D))]_a\ K^k *_a (pure(R))^k *_a (pure\ R')^k \to x\text{-}assn\rangle$
⟨*proof*⟩

lemma *add-pure-parameter2-twoargs*:
assumes $\langle\bigwedge C\ C'\ D\ D'.\ (C,\ C') \in R \Longrightarrow\ (D,\ D') \in R' \Longrightarrow (\lambda S.\ f\ S\ C\ D,\ \lambda S.\ f'\ S\ C'\ D') \in [\lambda S.\ P\ S\ C'\ D']_a\ A \to b\rangle$
shows $\langle(uncurry2\ f,\ uncurry2\ f') \in [uncurry2\ P]_a\ A *_a (pure\ R)^k *_a (pure\ R')^k \to b\rangle$
⟨*proof*⟩

lemma *remove-unused-unit-parameter*:
assumes $\langle(uncurry\ (\lambda S\ \text{-}.\ f\ S),\ uncurry\ (\lambda S\ \text{-}.\ f'\ S)) \in [uncurry\ (\lambda S\ \text{-}.\ P\ S)]_a\ A *_a (pure\ unit\text{-}rel)^k \to b\rangle$
shows $\langle(f,\ f') \in [P]_a\ A \to b\rangle$
⟨*proof*⟩

lemma *add-pure-parameter-unit*:
assumes $\langle(\lambda S.\ f\ S\ (),\ \lambda S.\ f'\ S\ ()) \in [\lambda S.\ P\ S]_a\ A \to b\rangle$
shows $\langle(f\ (),\ f'\ ()) \in [P]_a\ A \to b\rangle$
⟨*proof*⟩

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*: $\langle(f,\ f') \in [P]_a\ trail\text{-}pol\text{-}fast\text{-}assn^k \to x\text{-}assn\rangle$
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'*:
$\langle(uncurry16\ (\lambda M$ - - - - - - - - - - - - - - - - -. $f\ M),\ uncurry16\ (\lambda M$ - - - - - - - - - - - - - - - - -. $f'\ M)) \in$
$[uncurry16\ (\lambda M$ - - - - - - - - - - - - - - - - -. $P\ M)]_a$
$trail\text{-}pol\text{-}fast\text{-}assn^k *_a arena\text{-}fast\text{-}assn^k *_a conflict\text{-}option\text{-}rel\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a$
$watchlist\text{-}fast\text{-}assn^k *_a heuristic\text{-}bump\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k *_a cach\text{-}refinement\text{-}l\text{-}assn^k *_a$
$lbd\text{-}assn^k *_a out\text{-}learned\text{-}assn^k *_a isasat\text{-}stats\text{-}assn^k *_a heuristic\text{-}assn^k *_a$
$aivdom\text{-}assn^k *_a lcount\text{-}assn^k *_a opts\text{-}assn^k *_a arena\text{-}fast\text{-}assn^k *_a occs\text{-}assn^k \to x\text{-}assn\rangle$
⟨*proof*⟩

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*›
⟨*proof*⟩

end

locale *read-all-param-adder-ops* =
fixes *f'* :: ‹'*a* ⇒ *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* ⇒ '*e nres*› **and**
P :: ‹'*a* ⇒ *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* ⇒ *bool*›
begin
definition *mop* **where**
‹*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*
}›
end

locale *read-all-param-adder* = *read-all-param-adder-ops f' P*
for *f'* :: ‹'*a* ⇒ *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* ⇒ '*r nres*› **and**
P :: ‹'*a* ⇒ *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* ⇒ *bool*› +
fixes *R* **and** *f* **and** *x-assn* :: ‹'*r* ⇒ '*q* ⇒ *assn*›
assumes *not-deleted-code-refine*:
‹(*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*))
∈ [*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*$\rangle$
begin

context
begin
lemma *not-deleted-code-refine-tmp*:
$\langle\bigwedge C\ C'.\ (C,\ C') \in R \Longrightarrow (uncurry16\ (f\ C),\ uncurry16\ (f'\ C')) \in [uncurry16\ (P\ C')]_a$
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*$\rangle$
⟨*proof*⟩
end

lemma (**in** −) *case-isasat-int-split-getter*: $\langle P$ (*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*$\rangle$
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩
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*)$^k \to$ *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*)$^k \to$ *x-assn*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

lemmas *refine* = *XX.refine*
lemmas *mop-refine* = *XX.mop-refine*
end

locale *read-arena-param-adder-ops* =
  fixes $P :: \langle 'b \Rightarrow arena \Rightarrow bool\rangle$ **and** $f' :: \langle 'b \Rightarrow arena\text{-}el\ list \Rightarrow 'r\ nres\rangle$
begin
definition *mop* **where**
  $\langle mop\ N\ C = do\ \{$
    $ASSERT\ (P\ C\ (get\text{-}clauses\text{-}wl\text{-}heur\ N))$;
    $read\text{-}all\text{-}st\ (\lambda\text{-}\ N\ \text{- - - - - - - - - - - - - - - - -}.\ f'\ C\ N)\ N\quad \}\rangle$

end

locale *read-arena-param-adder* = *read-arena-param-adder-ops P f'*
  for $P :: \langle 'b \Rightarrow arena \Rightarrow bool\rangle$ **and** $f' :: \langle 'b \Rightarrow arena\text{-}el\ list \Rightarrow 'r\ nres\rangle$ +
  fixes $R :: \langle ('a \times 'b)\ set\rangle$ **and** f **and** *x-assn* $:: \langle 'r \Rightarrow 'q \Rightarrow assn\rangle$
  assumes *not-deleted-code-refine*: $\langle(uncurry\ (\lambda S\ C.\ f\ C\ S), uncurry\ (\lambda S\ C'.\ f'\ C'\ S)) \in [uncurry\ (\lambda S\ C.\ P\ C\ S)]_a\ arena\text{-}fast\text{-}assn^k *_a (pure\ R)^k \to x\text{-}assn\rangle$
begin

lemma *not-deleted-code-refine'*:
  $\langle(uncurry17\ (\lambda\text{-}\ M\ \text{- - - - - - - - - - - - - - - - -}\ C.\ f\ C\ M), uncurry17\ (\lambda\text{-}\ M\ \text{- - - - - - - - - - - - - - - - -}\ C'.\ f'\ C'\ M)) \in$
  $[uncurry17\ (\lambda\text{-}\ M\ \text{- - - - - - - - - - - - - - - - -}\ C'.\ P\ C'\ M)]_a$
   $trail\text{-}pol\text{-}fast\text{-}assn^k *_a arena\text{-}fast\text{-}assn^k *_a conflict\text{-}option\text{-}rel\text{-}assn^k *_a sint64\text{-}nat\text{-}assn^k *_a$
  $watchlist\text{-}fast\text{-}assn^k *_a heuristic\text{-}bump\text{-}assn^k *_a uint32\text{-}nat\text{-}assn^k *_a cach\text{-}refinement\text{-}l\text{-}assn^k *_a$
  $lbd\text{-}assn^k *_a out\text{-}learned\text{-}assn^k *_a isasat\text{-}stats\text{-}assn^k *_a heuristic\text{-}assn^k *_a$
  $aivdom\text{-}assn^k *_a lcount\text{-}assn^k *_a opts\text{-}assn^k *_a arena\text{-}fast\text{-}assn^k *_a occs\text{-}assn^k *_a (pure\ R)^k \to x\text{-}assn\rangle$
  ⟨*proof*⟩

sublocale *XX*: *read-all-param-adder* **where**
  $f = \langle(\lambda C\ \text{-}\ M\ \text{- - - - - - - - - - - - - - - - -}.\ f\ C\ M)\rangle$ **and**
  $f' = \langle(\lambda C\ \text{-}\ M\ \text{- - - - - - - - - - - - - - - - -}.\ f'\ C\ M)\rangle$ **and**
  $P = \langle(\lambda C\ \text{-}\ M\ \text{- - - - - - - - - - - - - - - - -}.\ P\ C\ M)\rangle$
  ⟨*proof*⟩
lemmas *refine* = *XX.refine*

lemma *mop-refine*:
  $\langle(uncurry\ (\lambda N\ C.\ read\text{-}all\text{-}st\text{-}code\ (\lambda\text{-}\ M\ \text{- - - - - - - - - - - - - - - - -}.\ f\ C\ M)\ N),$
    $uncurry\ mop)$
  $\in isasat\text{-}bounded\text{-}assn^k\ *_a\ (pure\ R)^k \to_a x\text{-}assn\rangle$
  ⟨*proof*⟩

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*: $\langle(f, f') \in [P]_a\ arena\text{-}fast\text{-}assn^k \to x\text{-}assn\rangle$
begin

sublocale *XX*: *read-arena-param-adder* **where**
  $f = \langle\lambda\text{-}\ N.\ f\ N\rangle$ **and**
  $f' = \langle\lambda\text{-}\ N.\ f'\ N\rangle$ **and**
  $P = \langle\lambda\text{-}.\ P\rangle$ **and**
  $R = \langle unit\text{-}rel\rangle$
  ⟨*proof*⟩

lemmas *refine* = *XX.refine*[*THEN remove-unused-unit-parameter*]

lemmas *mop-refine* = *XX.mop-refine*
end

lemma *merge-second-pure-argument-generalized*:
‹(*uncurry2 f*, *uncurry2 f'*)
∈ [*uncurry2 P*]$_a$ *A* $*_a$ (*pure R*)^k $*_a$ (*pure R'*)^k → *x-assn* ⟹
(*uncurry* (λ*S C*. (*case C of* (*C*, *D*) ⇒ *f S C D*)),
uncurry (λ*S C'*. (*case C' of* (*C*, *D*) ⇒ *f' S C D*)))
∈ [*uncurry*
(λ*S C*. (*case C of* (*C*, *D*) ⇒ *P S C D*))]$_a$ *A* $*_a$ (*pure* (*R* ×$_f$ *R'*))^k → *x-assn*›
⟨*proof*⟩

lemma *merge-second-pure-argument*:
‹(*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)]$_a$ *A* $*_a$ (*pure R*)^k $*_a$ (*pure R'*)^k → *x-assn* ⟹
(*uncurry* (λ*S C*. (*case C of* (*C*, *D*) ⇒ *f C D*) *S*),
uncurry (λ*S C'*. (*case C' of* (*C*, *D*) ⇒ *f' C D*) *S*))
∈ [*uncurry* (λ*S C*. (*case C of* (*C*, *D*) ⇒ *P C D*) *S*)]$_a$ *A* $*_a$ (*pure* (*R* ×$_f$ *R'*))^k → *x-assn*›
⟨*proof*⟩

lemma *merge-third-pure-argument'*:
‹(*uncurry3* (λ*S C D E*. *f C D E S*), *uncurry3* (λ*S C' D' E'*. *f' C' D' E' S*))
∈ [*uncurry3*
(λ*S C' D' E'*.
P C' D' E' S)]$_a$ *A* $*_a$ (*pure R*)^k $*_a$ (*pure R'*)^k $*_a$ (*pure R''*)^k → *x-assn* ⟹
(*uncurry2* (λ*S E C*. (*case C of* (*C*, *D*) ⇒ *f E C D*) *S*),
uncurry2 (λ*S E' C'*. (*case C' of* (*C*, *D*) ⇒ *f' E' C D*) *S*))
∈ [*uncurry2*
(λ*S E C*. (*case C of* (*C*, *D*) ⇒ *P E C D*)
S)]$_a$ *A* $*_a$ (*pure R*)^k $*_a$ (*pure* (*R'* ×$_f$ *R''*))^k → *x-assn*›
⟨*proof*⟩

abbreviation *read-arena-wl-heur-code* :: ‹-› **where**
‹*read-arena-wl-heur-code f* ≡ *read-all-st-code* (λ- *M* - - - - - - - - - - - - - - - -. *f M*)›
abbreviation *read-arena-wl-heur* :: ‹-› **where**
‹*read-arena-wl-heur f* ≡ *read-all-st* (λ- *M* - - - - - - - - - - - - - - - -. *f M*)›

locale *read-arena-param-adder2-twoargs-ops* =
fixes
f' :: ‹'*b* ⇒ '*d* ⇒ *arena* ⇒ '*r nres*› **and**
P :: ‹'*b* ⇒ '*d* ⇒ *arena* ⇒ *bool*›
begin
definition *mop* **where**
‹*mop N C C'* = *do* {
ASSERT (*P C C'* (*get-clauses-wl-heur N*));
read-arena-wl-heur (λ*N*. *f' C C' N*) *N*
}›
end

locale *read-arena-param-adder2-twoargs* =
read-arena-param-adder2-twoargs-ops f' P
for *f'* :: ‹'*b* ⇒ '*d* ⇒ *arena* ⇒ '*r nres*› **and** *P* +

fixes R **and** R' **and** f **and** *x-assn* :: ‹$'r \Rightarrow 'q \Rightarrow$ *assn*›
assumes *not-deleted-code-refine*:
‹(*uncurry2* ($\lambda S\ C\ D.\ f\ C\ D\ S$), *uncurry2* ($\lambda S\ C'\ D'.\ f'\ C'\ D'\ S$)) ∈
[*uncurry2* ($\lambda S\ C'\ D'.\ P\ C'\ D'\ S$)]$_a$ *arena-fast-assn*^{k} $*_a$ (*pure* R)^k $*_a$ (*pure* R')^k → *x-assn*›
begin

sublocale *XX*: *read-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 = ‹$R \times_f R'$›
⟨*proof*⟩

lemma *refine*:
‹(*uncurry2* ($\lambda N\ C\ D.$ *read-arena-wl-heur-code* ($f\ C\ D$) N),
uncurry2 ($\lambda N\ C'\ D.$ *read-arena-wl-heur* ($f'\ C'\ D$) N))
∈ [*uncurry2* ($\lambda S\ C\ D.\ P\ C\ D$ (*get-clauses-wl-heur* S))]$_a$ *isasat-bounded-assn*^{k} $*_a$ (*pure* R)^k $*_a$ (*pure* R')^k → *x-assn*›
⟨*proof*⟩

lemma *mop-refine*:
‹(*uncurry2* ($\lambda N\ C\ D.$ *read-arena-wl-heur-code* ($\lambda N.\ f\ C\ D\ N$) N), *uncurry2 mop*) ∈ *isasat-bounded-assn*^{k} $*_a$ (*pure* R)^k $*_a$ (*pure* R')^k →$_a$ *x-assn*›
⟨*proof*⟩

end

abbreviation *read-conflict-wl-heur-code* :: ‹-› **where**
‹*read-conflict-wl-heur-code* $f \equiv$ *read-all-st-code* (λ- - M - - - - - - - - - - - - - - -. $f\ M$)›
abbreviation *read-conflict-wl-heur* :: ‹-› **where**
‹*read-conflict-wl-heur* $f \equiv$ *read-all-st* (λ- - M - - - - - - - - - - - - - - -. $f\ M$)›

locale *read-conflict-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* ($\lambda S\ C'.\ f'\ C'\ S$)) ∈ [*uncurry* ($\lambda S\ C.\ P\ C\ S$)]$_a$ *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*›
⟨*proof*⟩

sublocale *XX* : *read-all-param-adder* **where**
f = ‹(λC - - M - - - - - - - - - - - - - - -. $f\ C\ M$)› **and**
f' = ‹(λC - - M - - - - - - - - - - - - - - -. $f'\ C\ M$)› **and**
P = ‹(λC - - M - - - - - - - - - - - - - - -. $P\ C\ M$)›
⟨*proof*⟩

lemmas *refine* = *XX.refine*
end

locale *read-conflict-param-adder0* =
  fixes *f* **and** *f'* **and** *x-assn* :: ‹'r ⇒ 'q ⇒ *assn*› **and** *P*
  assumes *not-deleted-code-refine*: ‹(*f*, *f'*) ∈ [*P*]$_a$ *conflict-option-rel-assn*^{k} → *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*
  ⟨*proof*⟩

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* ≡ *read-all-st-code* (λ- - - *M* - - - - - - - - - - - - - -. *f M*)›
abbreviation *read-literals-to-update-wl-heur* :: ‹-› **where**
  ‹*read-literals-to-update-wl-heur f* ≡ *read-all-st* (λ- - - *M* - - - - - - - - - - - - - -. *f M*)›

locale *read-literals-to-update-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*)]$_a$ *sint64-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*›
  ⟨*proof*⟩

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*)›
  ⟨*proof*⟩

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} → *x-assn*›
begin
sublocale *XX*: *read-literals-to-update-param-adder* **where**
  f = ‹λ-. *f*› **and**

$f' = ‹\lambda\text{-}.\ f'›$ **and**
$P = ‹\lambda\text{-}.\ P›$
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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**
$f = ‹\lambda\text{-}.\ f›$ **and**
$f' = ‹\lambda\text{-}.\ f'›$ **and**
$P = ‹\lambda\text{-}.\ P›$
⟨*proof*⟩

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* :: ‹$'r \Rightarrow 'q \Rightarrow$ *assn*› **and** P
assumes *not-deleted-code-refine*:
‹(*uncurry2* ($\lambda S\ C\ D.\ f\ C\ D\ S$), *uncurry2* ($\lambda S\ C'\ D'.\ f'\ C'\ D'\ S$)) ∈
[*uncurry2* ($\lambda S\ C'\ D'.\ P\ C'\ D'\ S$)]$_a$ *watchlist-fast-assn*^{k} $*_a$ (*pure* R)^k $*_a$ (*pure* R')^k → *x-assn*›
begin

sublocale *XX*: *read-watchlist-param-adder* **where**
f = ‹$\lambda(C,D).\ f\ C\ D$› **and**
f' = ‹$\lambda(C,D).\ f'\ C\ D$› **and**
P = ‹$\lambda(C,D).\ P\ C\ D$› **and**
R = ‹$R \times_f R'$›
⟨*proof*⟩

lemma *refine*:
‹(*uncurry2* ($\lambda N\ C\ D.$ *read-watchlist-wl-heur-code* ($f\ C\ D$) N),
uncurry2 ($\lambda N\ C'\ D'.$ *read-watchlist-wl-heur* ($f'\ C'\ D'$) N))
∈ [*uncurry2* ($\lambda S\ C\ D.\ P\ C\ D$ (*get-watched-wl-heur* S))]$_a$
isasat-bounded-assn^k $*_a$ (*pure* R)^k $*_a$ (*pure* R')^k→ *x-assn*›
⟨*proof*⟩

lemmas *mop-refine* = *XX.XX.mop-refine*
end

abbreviation *read-vmtf-wl-heur-code* :: ‹-› **where**
‹*read-vmtf-wl-heur-code* $f \equiv$ *read-all-st-code* (λ- - - - - M - - - - - - - - - - -. $f\ M$)›
abbreviation *read-vmtf-wl-heur* :: ‹-› **where**
‹*read-vmtf-wl-heur* $f \equiv$ *read-all-st* (λ- - - - - M - - - - - - - - - - -. $f\ M$)›

locale *read-vmtf-param-adder* =
fixes f **and** f' **and** *x-assn* :: ‹$'r \Rightarrow 'q \Rightarrow$ *assn*› **and** P **and** R
assumes *not-deleted-code-refine*: ‹(*uncurry* ($\lambda S\ C.\ f\ C\ S$), *uncurry* ($\lambda S\ C'.\ f'\ C'\ S$)) ∈ [*uncurry* ($\lambda S\ C.\ P\ C\ S$)]$_a$ *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*›
⟨*proof*⟩

sublocale *XX* : *read-all-param-adder* **where**
f = ‹(λC - - - - - M - - - - - - - - - - -. $f\ C\ M$)› **and**
f' = ‹(λC - - - - - M - - - - - - - - - - -. $f'\ C\ M$)› **and**
P = ‹(λC - - - - - M - - - - - - - - - - -. $P\ C\ M$)›
⟨*proof*⟩

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*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

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*)›
  ⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sublocale *XX* : *read-all-param-adder* **where**
f = ‹(λ*C* - - - - - - - - - *M* - - - - - - -. *f C M*)› **and**

$f' = ‹(\lambda C$ - - - - - - - - - M - - - - - - -. f' C $M)›$ **and**
$P = ‹(\lambda C$ - - - - - - - - - M - - - - - - -. P C $M)›$
⟨*proof*⟩

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*$^k \rightarrow$ *x-assn*$›$
begin

sublocale *XX*: *read-outl-param-adder* **where**
$f = ‹\lambda$-. $f›$ **and**
$f' = ‹\lambda$-. $f'›$ **and**
$P = ‹\lambda$-. $P›$
⟨*proof*⟩

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* ≡ *read-all-st-code* (λ- - - - - - - - - - M - - - - - -. f M)›
abbreviation *read-stats-wl-heur* :: ‹- $\Rightarrow$ *isasat* $\Rightarrow$ -› **where**
‹*read-stats-wl-heur f* ≡ *read-all-st* (λ- - - - - - - - - - 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* (λS C. f C S), *uncurry* (λS C'. f' C' S)) ∈ [*uncurry* (λS C. P C S)]$_a$ *isasat-stats-assn*^{k} $*_a$ (*pure* R)$^k \rightarrow$ *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)$^k \rightarrow$ *x-assn*›
⟨*proof*⟩

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 $M)›$
⟨*proof*⟩

lemmas *refine* = *XX.refine*
end

locale *read-stats-param-adder0* =
fixes f **and** f' **and** *x-assn* :: $‹'r \Rightarrow 'q \Rightarrow$ *assn*$›$ **and** P

assumes *not-deleted-code-refine*: ‹$(f, f') \in [P]_a$ *isasat-stats-assn*$^k \to$ *x-assn*›
begin
sublocale *XX*: *read-stats-param-adder* **where**
$f = ‹\lambda\text{-}.\ f›$ **and**
$f' = ‹\lambda\text{-}.\ f'›$ **and**
$P = ‹\lambda\text{-}.\ P›$
⟨*proof*⟩

lemmas *refine* = *XX.refine*[*THEN remove-unused-unit-parameter*]
end

abbreviation *read-heur-wl-heur-code* :: ‹- ⇒ *twl-st-wll-trail-fast2* ⇒ -› **where**
‹*read-heur-wl-heur-code f* ≡ *read-all-st-code* (λ- - - - - - - - - - - *M* - - - - -. *f M*)›
abbreviation *read-heur-wl-heur* :: ‹- ⇒ *isasat* ⇒ -› **where**
‹*read-heur-wl-heur f* ≡ *read-all-st* (λ- - - - - - - - - - - *M* - - - - -. *f M*)›

locale *read-heur-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-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*)) ∈
[*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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*$^k \to$ *x-assn*›
begin
sublocale *XX*: *read-heur-param-adder* **where**
$f = ‹\lambda\text{-}.\ f›$ **and**
$f' = ‹\lambda\text{-}.\ f'›$ **and**
$P = ‹\lambda\text{-}.\ P›$
⟨*proof*⟩

lemmas *refine* = *XX.refine*[*THEN remove-unused-unit-parameter*]
end

locale *read-heur-param-adder2* =
fixes f **and** f' **and** *x-assn* :: ‹$'r \Rightarrow 'q \Rightarrow$ *assn*› **and** P **and** R **and** R'
assumes *not-deleted-code-refine*: ‹(*uncurry2* ($\lambda S\ C\ D.\ f\ C\ D\ S$), *uncurry2* ($\lambda S\ C'\ D'.\ f'\ C'\ D'\ S$)) ∈ [*uncurry2* ($\lambda S\ C\ D.\ P\ C\ D\ S$)]$_a$ *heuristic-assn*^{k} $*_a$ (*pure* R)^k$*_a$ (*pure* R')^k → *x-assn*›
begin
sublocale *XX*: *read-heur-param-adder* **where**
f = ‹$\lambda(C,D)\ N.\ f\ C\ D\ N$› **and**
f' = ‹$\lambda(C,D)\ N.\ f'\ C\ D\ N$› **and**
P = ‹$\lambda(C,D)\ N.\ P\ C\ D\ N$› **and**
R = ‹$R \times_f R'$›
⟨*proof*⟩

lemma *refine*:
‹(*uncurry2* ($\lambda N\ C\ D.$ *read-heur-wl-heur-code* ($f\ C\ D$) N),
uncurry2 ($\lambda N\ C'\ D.$ *read-heur-wl-heur* ($f'\ C'\ D$) N))
∈ [*uncurry2* ($\lambda S\ C\ D.\ P\ C\ D$ (*get-heur* S))]$_a$ *isasat-bounded-assn*^{k} $*_a$ (*pure* R)^k $*_a$ (*pure* R')^k → *x-assn*›
⟨*proof*⟩

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* :: ‹$'r \Rightarrow 'q \Rightarrow$ *assn*› **and** P **and** R
assumes *not-deleted-code-refine*: ‹(*uncurry* ($\lambda S\ C.\ f\ C\ S$), *uncurry* ($\lambda S\ C'.\ f'\ C'\ S$)) ∈ [*uncurry* ($\lambda S\ C.\ P\ C\ S$)]$_a$ *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*›
⟨*proof*⟩

sublocale *XX* : *read-all-param-adder* **where**
f = ‹(λC - - - - - - - - - - - - M - - - -. $f\ C\ M$)› **and**
f' = ‹(λC - - - - - - - - - - - - M - - - -. $f'\ C\ M$)› **and**
P = ‹(λC - - - - - - - - - - - - M - - - -. $P\ C\ M$)›
⟨*proof*⟩

lemmas *refine* = *XX.refine*
end

locale *read-vdom-param-adder0* =
fixes f **and** f' **and** *x-assn* :: ‹$'r \Rightarrow 'q \Rightarrow$ *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' = ‹λ-. *f'*› **and**
P = ‹λ-. *P*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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* :: ‹$'r \Rightarrow 'q \Rightarrow$ *assn*› **and** P
assumes *not-deleted-code-refine*: ‹(*uncurry* ($\lambda S\ C.\ f\ C\ S$), *uncurry* ($\lambda S\ C'.\ f'\ C'\ S$)) ∈ [*uncurry* ($\lambda S\ C.\ P\ C\ S$)]$_a$ *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*›
⟨*proof*⟩

sublocale *XX* : *read-all-param-adder* **where**
f = ‹(λC - - - - - - - - - - - - - - - - M -. $f\ C\ M$)› **and**
f' = ‹(λC - - - - - - - - - - - - - - - - M -. $f'\ C\ M$)› **and**
P = ‹(λC - - - - - - - - - - - - - - - - M -. $P\ C\ M$)›
⟨*proof*⟩

lemmas *refine* = *XX.refine*
end

locale *read-old-arena-param-adder0* =
fixes f **and** f' **and** *x-assn* :: ‹$'r \Rightarrow 'q \Rightarrow$ *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›
⟨*proof*⟩

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* :: ‹$'r \Rightarrow 'q \Rightarrow$ *assn*› **and** P **and** R
assumes *not-deleted-code-refine*: ‹(*uncurry* ($\lambda S\ C.\ f\ C\ S$), *uncurry* ($\lambda S\ C'.\ f'\ C'\ S$)) ∈ [*uncurry* ($\lambda S\ C.\ P\ C\ S$)]$_a$ *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- - - - - - - - - - - - - - - - $M\ C'.$

$f'\ C'\ M)) \in$
[*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*›
⟨*proof*⟩

sublocale *XX* : *read-all-param-adder* **where**
$f =$ ‹(λC - - - - - - - - - - - - - - - - - $M.\ f\ C\ M$)› **and**
$f' =$ ‹(λC - - - - - - - - - - - - - - - - - $M.\ f'\ C\ M$)› **and**
$P =$ ‹(λC - - - - - - - - - - - - - - - - - $M.\ P\ C\ M$)›
⟨*proof*⟩

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*$^k \to$ *x-assn*›
begin

sublocale *XX*: *read-occs-param-adder* **where**
$f =$ ‹λ-. *f*› **and**
$f' =$ ‹λ-. *f'*› **and**
$P =$ ‹λ-. *P*›
⟨*proof*⟩

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* ($\lambda S\ C\ D.\ f\ C\ D\ S$), *uncurry2* ($\lambda S\ C'\ D'.\ f'\ C'\ D'\ S$)) ∈
[*uncurry2* ($\lambda S\ C\ D.\ P\ C\ D\ S$)]$_a$ *occs-assn*^{k} $*_a$ (*pure R*)$^k *_a$ (*pure R'*)$^k \to$ *x-assn*›
begin

sublocale *XX*: *read-occs-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**
$R =$ ‹$R \times_f R'$›
⟨*proof*⟩

lemma *refine*:
‹(*uncurry2* ($\lambda N\ C\ D.$ *read-occs-wl-heur-code* (*f C D*) *N*),
uncurry2 ($\lambda N\ C'\ D.$ *read-occs-wl-heur* (*f' C' D*) *N*))
∈ [*uncurry2* ($\lambda S\ C\ D.\ P\ C\ D$ (*get-occs S*))]$_a$ *isasat-bounded-assn*^{k} $*_a$ (*pure R*)$^k *_a$ (*pure R'*)$^k \to$
x-assn›
⟨*proof*⟩

lemma *mop-refine*:
‹(*uncurry2* ($\lambda N\ C\ D.$ *read-occs-wl-heur-code* (*f C D*) *N*), *uncurry2* ($\lambda N\ C\ D.$ *XX.XX.mop N* (*C*,*D*)))
∈
isasat-bounded-assn^k $*_a$ (*pure R*)$^k *_a$ (*pure R'*)$^k \to_a$ *x-assn*›

⟨*proof*⟩

end

locale *read-trail-arena-param-adder-ops* =
fixes $P :: ‹'b \Rightarrow$ *trail-pol* $\Rightarrow$ *arena* $\Rightarrow$ *bool*$›$ **and** $f' :: ‹'b \Rightarrow$ *trail-pol* $\Rightarrow$ *arena-el list* $\Rightarrow 'r$ *nres*$›$
begin

definition *mop* **where**
‹*mop N C* = *do* {
ASSERT (*P C* (*get-trail-wl-heur N*) (*get-clauses-wl-heur N*));
read-all-st (λ*M N - - - - - - - - - - - - - - - -*. *f' C M N*) *N*
}›

end

locale *read-trail-arena-param-adder* = *read-trail-arena-param-adder-ops P f'*
for $P :: ‹'b \Rightarrow$ *trail-pol* $\Rightarrow$ *arena* $\Rightarrow$ *bool*$›$ **and** $f' :: ‹'b \Rightarrow$ *trail-pol* $\Rightarrow$ *arena-el list* $\Rightarrow 'r$ *nres*$›$ +
fixes $R :: ‹('a \times 'b)$ *set*$›$ **and** *f* **and** *x-assn* $:: ‹'r \Rightarrow 'q \Rightarrow$ *assn*$›$
assumes *not-deleted-code-refine*: ‹(*uncurry2* (λ*S T C*. *f C S T*), *uncurry2* (λ*S T C'*. *f' C' S T*)) $\in$ [*uncurry2* (λ*S T C*. *P C S T*)]$_a$ *trail-pol-fast-assn*^{k} $*_a$ *arena-fast-assn*^{k} $*_a$ (*pure R*)$^k \to$ *x-assn*›
begin

lemma *not-deleted-code-refine'*:
‹(*uncurry17* (λ*M N - - - - - - - - - - - - - - - - C*. *f C M N*), *uncurry17* (λ*M N - - - - - - - - - - - - - - - - C'*. *f' C' M N*)) $\in$
[*uncurry17* (λ*M 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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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)
$\in$ *isasat-bounded-assn*^{k} $*_a$ (*pure R*)$^k \to_a$ *x-assn*›
⟨*proof*⟩
end

locale *read-trail-arena-param-adder2-twoargs-ops* =
fixes
$f' :: ‹'b \Rightarrow 'd \Rightarrow$ *trail-pol* $\Rightarrow$ *arena* $\Rightarrow 'r$ *nres*$›$ **and**
$P :: ‹'b \Rightarrow 'd \Rightarrow$ *trail-pol* $\Rightarrow$ *arena* $\Rightarrow$ *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* ⇒ *′q* ⇒ *assn*›
assumes *not-deleted-code-refine*:
‹(*uncurry3* (λ*S T C D. f C D S T*), *uncurry3* (λ*S T C′ D′. f′ C′ D′ S T*)) ∈
[*uncurry3* (λ*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 → *x-assn*›
begin

sublocale *XX*: *read-trail-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′*›
⟨*proof*⟩

lemmas *refine* = *XX.refine*[*unfolded case-isasat-int-split-getter*]

lemma *mop-refine*:
‹(*uncurry2* (λ*N C D. read-all-st-code*
(λ*M N* - - - - - - - - - - - - - - - -. *f C D M N*) *N*), *uncurry2 mop*) ∈ *isasat-bounded-assn*^{k} *∗$_a$* (*pure R*)^k *∗$_a$* (*pure R′*)^k →$_a$ *x-assn*›
⟨*proof*⟩
end

locale *read-trail-arena-param-adder2-threeargs-ops* =
fixes
f′ :: ‹*′b* ⇒ *′d* ⇒ *′e* ⇒ *trail-pol* ⇒ *arena* ⇒ *′r nres*› **and**
P :: ‹*′b* ⇒ *′d* ⇒ *′e* ⇒ *trail-pol* ⇒ *arena* ⇒ *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 (λ*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* → *x-assn*›
shows
‹(*uncurry3 g*, *uncurry3* (λ*N C D E. do* {*ASSERT* (*P N C D E*); *h N C D E*}))
∈ *A ∗$_a$ B ∗$_a$ C ∗$_a$ D* →$_a$ *x-assn*›
⟨*proof*⟩

locale *read-trail-arena-param-adder2-threeargs* =
read-trail-arena-param-adder2-threeargs-ops f′ P
for *f′* :: ‹*′b* ⇒ *′d* ⇒ *′e* ⇒ *trail-pol* ⇒ *arena* ⇒ *′r nres*› **and** *P* +
fixes *R* **and** *R′* **and** *R″* **and** *f* **and** *x-assn* :: ‹*′r* ⇒ *′q* ⇒ *assn*›

assumes *not-deleted-code-refine*:
‹(*uncurry4* (λ*S T C D E*. *f C D E S T*), *uncurry4* (λ*S T C′ D′ E′*. *f′ C′ D′ E′ S T*)) ∈
[*uncurry4* (λ*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 → *x-assn*›
begin
sublocale *XX*: *read-trail-arena-param-adder* **where**
f = ‹λ(*C*,*D*,*E*) *N*. *f C D E N*› **and**
f′ = ‹λ(*C*,*D*,*E*) *N*. *f′ C D E N*› **and**
P = ‹λ(*C*,*D*,*E*) *N*. *P C D E N*› **and**
R = ‹*R* ×$_r$ *R′* ×$_r$ *R″*›
⟨*proof*⟩

It would be better to this without calling auto, but fighting uncurry is just too hard and not really useful.

lemma *refine*:
‹(*uncurry3* (λ*N C D E*. *read-all-st-code* (λ*M N* - - - - - - - - - - - - - - - -. *f C D E M N*) *N*),
uncurry3 (λ*N C D E*. *read-all-st* (λ*M N* - - - - - - - - - - - - - - - -. *f′ C D E M N*) *N*))
∈ [*uncurry3* (λ*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 → *x-assn*›
⟨*proof*⟩
lemma *mop-refine*:
‹(*uncurry3* (λ*N C D E*. *read-all-st-code*
(λ*M N* - - - - - - - - - - - - - - - -. *f C D E M N*) *N*), *uncurry3 mop*) ∈ *isasat-bounded-assn*^{k} *$*_a$ (*pure R*)^k *$*_a$ (*pure R′*)^k *$*_a$ (*pure R″*)^k →$_a$ *x-assn*›
⟨*proof*⟩

end

locale *read-trail-vmtf-param-adder* =
fixes *P* :: ‹′*b* ⇒ - ⇒ *bump-heuristics* ⇒ *bool*› **and** *f′* :: ‹′*b* ⇒ *trail-pol* ⇒ - ⇒ ′*r nres*› **and**
R :: ‹(′*a* × ′*b*) *set*› **and** *f* **and** *x-assn* :: ‹′*r* ⇒ ′*q* ⇒ *assn*›
assumes *not-deleted-code-refine*: ‹(*uncurry2* (λ*S T C*. *f C S T*), *uncurry2* (λ*S T C′*. *f′ C′ S T*)) ∈
[*uncurry2* (λ*S T C*. *P C S T*)]$_a$ *trail-pol-fast-assn*^{k} *$*_a$ *heuristic-bump-assn*^{k} *$*_a$ (*pure R*)^k → *x-assn*›
begin

lemma *not-deleted-code-refine′*:
‹(*uncurry17* (λ*M* - - - - *N* - - - - - - - - - - - *C*. *f C M N*), *uncurry17* (λ*M* - - - - *N* - - - - - - - - - - - *C′*. *f′ C′ M N*)) ∈
[*uncurry17* (λ*M* - - - - *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 → *x-assn*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*
⟨*proof*⟩
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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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} $\to_a$ *bool1-assn*›
⟨*proof*⟩

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} $\to_a$ *bool1-assn*›
⟨*proof*⟩

lemma *get-conflict-wl-is-None-heur-alt-def*: ‹*read-conflict-wl-heur conflict-is-None* = *RETURN* ∘ *get-conflict-wl-is-None*
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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 (λN 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*›
⟨*proof*⟩

lemma *arena-lit2-impl-arena-lit*:
assumes ‹(*C*, *C'*) ∈ *snat-rel*› **and**
‹(*D*, *D'*) ∈ *snat-rel*›
shows ‹(*λS. arena-lit2-impl S C D*, *λS. RETURN (arena-lit S (C' + D'))*)
∈ [*λS. arena-lit-pre S (C' + D') ∧ length S ≤ snat64-max*]$_a$ (*al-assn arena-el-impl-assn*)^k → *unat-lit-assn*›
⟨*proof*⟩

lemma *access-lit-in-clauses-heur-alt-def*:
‹*RETURN ooo access-lit-in-clauses-heur = (λN C' D. read-arena-wl-heur (λN. RETURN (arena-lit N (C' + D))) N)*›
⟨*proof*⟩

lemma *access-lit-in-clauses-heur-pre*:
‹*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)›
⟨*proof*⟩

definition *access-lit-in-clauses-heur-fast-code* :: ‹*twl-st-wll-trail-fast2 ⇒ - ⇒ - ⇒ -*› **where**
‹*access-lit-in-clauses-heur-fast-code = (λN C D. read-arena-wl-heur-code (λN. 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)›

⟨*proof*⟩

lemma *refine-ASSERT-move-to-pre2'*:
‹(*uncurry2 g*, *uncurry2 h*) ∈ [*uncurry2* (λ*a b c*. *P a b c* ∧ *Q a b c*)]$_a$ *A* ∗$_a$ *B* ∗$_a$ *C* → *x-assn* ⟷ (*uncurry2 g*, *uncurry2* (λ*N C D*. *do* {*ASSERT* (*P N C D*); *h N C D*}))
∈ [*uncurry2 Q*]$_a$ *A* ∗$_a$ *B* ∗$_a$ *C* → *x-assn*›
⟨*proof*⟩

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* (λ*N*. *RETURN* (*arena-lit N* (*C* + *C'*))) *N*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

lemma *isasat-bounded-assn-length-arenaD*: ‹*rdomp isasat-bounded-assn a* ⟹ *length* (*get-clauses-wl-heur a*) ≤ *snat64-max*› ⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *rewatch-heur-st-fast-alt-def*:
‹*rewatch-heur-st-fast* = (λ*S*$_0$. *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
})›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

global-interpretation *arena-is-learned*: *read-arena-param-adder* **where**
R = ‹(*snat-rel'* *TYPE*(*64*))› **and**
f' = ‹λ*N C*. (*RETURN oo is-learned*) *C N*› **and**
f = ‹(λ*N C*. *is-learned-impl C N*)› **and**
x-assn = *bool1-assn* **and**
P = ‹λ*C N*. *arena-is-valid-clause-vdom N C*›
rewrites
‹(λ*N C*. *read-arena-wl-heur-code* (λ*Ca. is-learned-impl Ca C*) *N*) = *clause-is-learned-heur-code2*›
and
‹(λ*N C'*. *read-arena-wl-heur* (λ*C*. (*RETURN* ∘∘ *is-learned*) *C C'*) *N*) = *RETURN oo clause-is-learned-heur*›

⟨*proof*⟩

definition *clause-lbd-heur-code2* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*clause-lbd-heur-code2* = (λ*N C*. *read-arena-wl-heur-code* (λ*Ca*. *arena-lbd-impl Ca C*) *N*)›

lemma *clause-lbd-heur-alt-def*:
‹*RETURN* ∘∘ *clause-lbd-heur* = (λ*N C'*. *read-arena-wl-heur* (λ*C*. (*RETURN* ∘∘ *arena-lbd*) *C C'*) *N*)›
⟨*proof*⟩

global-interpretation *arena-get-lbd*: *read-arena-param-adder* **where**
R = ‹(*snat-rel'* *TYPE*(*64*))› **and**
f' = ‹λ*N C*. (*RETURN oo arena-lbd*) *C N*› **and**
f = ‹(λ*N C*. *arena-lbd-impl C N*)› **and**
x-assn = *uint32-nat-assn* **and**
P = ‹λ*C N*. *get-clause-LBD-pre N C*›
rewrites
‹(λ*N C*. *read-arena-wl-heur-code* (λ*Ca. arena-lbd-impl Ca C*) *N*) = *clause-lbd-heur-code2*› **and**
‹(λ*N C'*. *read-arena-wl-heur* (λ*C*. (*RETURN* ∘∘ *arena-lbd*) *C C'*) *N*) = *RETURN oo clause-lbd-heur*›
and
‹*arena-get-lbd.mop* = *mop-arena-lbd-st*›
⟨*proof*⟩

definition *clause-pos-heur-code2* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**

‹*clause-pos-heur-code2* = (λ*N C*. *read-arena-wl-heur-code* (λ*Ca*. *arena-pos-impl Ca C*) *N*)›

definition *mop-arena-pos-st* :: ‹-› **where**
‹*mop-arena-pos-st S* = *mop-arena-pos* (*get-clauses-wl-heur S*)›

global-interpretation *arena-get-pos*: *read-arena-param-adder* **where**
R = ‹(*snat-rel*′ *TYPE*(*64*))› **and**
f′ = ‹λ*N C*. (*RETURN oo arena-pos*) *C N*› **and**
f = ‹(λ*N C*. *arena-pos-impl C N*)› **and**
x-assn = *sint64-nat-assn* **and**
P = ‹λ*C N*. *get-saved-pos-pre N C*›
rewrites
‹(λ*N C*. *read-arena-wl-heur-code* (λ*Ca*. *arena-pos-impl Ca C*) *N*) = *clause-pos-heur-code2*› **and**
‹*arena-get-pos.mop* = *mop-arena-pos-st*›
⟨*proof*⟩

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*:
‹*mop-mark-garbage-heur C i* = (λ*S*$_0$. *do* {
ASSERT(*mark-garbage-pre* (*get-clauses-wl-heur S*$_0$, *C*) ∧ *clss-size-lcount* (*get-learned-count S*$_0$) ≥ *1*
∧ *i* < *length* (*get-avdom 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 (*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*)
})›
⟨*proof*⟩

sepref-def *mop-mark-garbage-heur-impl*
is ‹*uncurry2 mop-mark-garbage-heur*›
:: ‹[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*›
⟨*proof*⟩

lemma *mark-garbage-heur-alt-def*: ‹*RETURN ooo mark-garbage-heur* =

(λC 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)})›
⟨proof⟩

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›
⟨proof⟩

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
})›
⟨proof⟩

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›
⟨proof⟩

lemma *delete-index-vdom-heur-alt-def*: ‹RETURN oo delete-index-vdom-heur = (λi 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)
})›
⟨proof⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *id-lcount-assn*: ‹(*Mreturn*, *RETURN*) ∈ (*lcount-assn*)^k →$_a$ *lcount-assn*›
⟨*proof*⟩

lemma *get-learned-count-alt-def*: ‹*RETURN o get-learned-count* = *read-lcount-wl-heur RETURN*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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**

$P = ‹(\lambda\text{-}.\ True)›$
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*›
⟨*proof*⟩

definition *get-learned-count-number'* :: ‹*isasat* ⇒ *nat*› **where**
‹*get-learned-count-number'* $S \equiv$ *get-learned-count-number* S›

lemma [*def-pat-rules*]: ‹*get-learned-count-number*$S≡*get-learned-count-number'*$S›
⟨*proof*⟩

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*›
:: ‹$[\lambda S.\ learned\text{-}clss\text{-}count\ S \leq unat64\text{-}max]_a$ *isasat-bounded-assn*^{k} → *uint64-nat-assn*›
⟨*proof*⟩

definition *marked-as-used-st-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*marked-as-used-st-fast-code* = ($\lambda N\ C.$ *read-arena-wl-heur-code* ($\lambda N.$ *marked-as-used-impl N C*) *N*)›
global-interpretation *marked-used*: *read-arena-param-adder* **where**
$R = ‹snat\text{-}rel'\ TYPE(64)›$ **and**
$f' = ‹\lambda C\ N.\ RETURN\ (marked\text{-}as\text{-}used\ N\ C)›$ **and**
$f = ‹(\lambda C'\ N.\ marked\text{-}as\text{-}used\text{-}impl\ N\ C')›$ **and**
x-assn = ‹*unat-assn'* *TYPE*(*2*)› **and**
$P = ‹(\lambda C\ S.\ marked\text{-}as\text{-}used\text{-}pre\ S\ C)›$
rewrites
‹($\lambda N\ C'.$ *read-arena-wl-heur* ($\lambda N.$ *RETURN* (*marked-as-used N C'*)) *N*) = *RETURN oo marked-as-used-st*›
and
‹($\lambda N\ C.$ *read-arena-wl-heur-code* ($\lambda N.$ *marked-as-used-impl N C*) *N*) = *marked-as-used-st-fast-code*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*))})›
⟨*proof*⟩

lemma *isa-vmtf-enqueue*:
‹(*uncurry2 isa-vmtf-enqueue*, *uncurry2* (*RETURN ooo vmtf-enqueue*)) ∈
[λ((*M*, *L*), -). *L* ∈# *A*]$_f$ (*trail-pol A*) ×$_f$ *nat-rel* ×$_f$ *Id* → ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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* ∈# *A*]$_f$ (*trail-pol A*) ×$_f$ *nat-rel* ×$_f$ *Id* → ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

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*)›

⟨*proof*⟩

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*))›
⟨*proof*⟩

lemma *insert-sort-reorder-list*:
assumes *trans*: ‹⋀ *x y z*. ⟦*R* (*h x*) (*h y*); *R* (*h y*) (*h z*)⟧ ⟹ *R* (*h x*) (*h z*)› **and** *lin*: ‹⋀*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*⟩

lemma *quicksort-vmtf-nth-reorder*:
‹(*uncurry quicksort-vmtf-nth*, *uncurry reorder-list*) ∈
Id ×*r* ⟨*Id*⟩*list-rel* →*f* ⟨*Id*⟩ *nres-rel*›
⟨*proof*⟩

lemma *atoms-hash-del-op-set-delete*:
‹(*uncurry* (*RETURN oo atoms-hash-del*),
uncurry (*RETURN oo Set.remove*)) ∈
nat-rel ×*r* *atoms-hash-rel* *A* →*f* ⟨*atoms-hash-rel* *A*⟩*nres-rel*›
⟨*proof*⟩

definition *current-stamp* **where**
‹*current-stamp vm* = *fst* (*snd vm*)›

lemma *current-stamp-alt-def*:
‹*current-stamp* = (λ(-, *m*, -). *m*)›
⟨*proof*⟩

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*))
})›
⟨*proof*⟩

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* *A*)) ∈ *trail-pol* (*A*::*nat multiset*) ×$_f$ *Id* ×$_f$ *Id*
→$_f$ ⟨*Id*×$_f$ *Id*⟩*nres-rel*›
⟨*proof*⟩

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*⟩

lemma *atms-hash-insert-pre*:
assumes ‹*L* ∈# *A*› **and** ‹(*x*, *x*′) ∈ *distinct-atoms-rel A*› **and** ‹*isasat-input-bounded A*›
shows ‹*atms-hash-insert-pre L x*›
⟨*proof*⟩

lemma *atoms-hash-del-op-set-insert*:
‹(*uncurry* (*RETURN oo atoms-hash-insert*), *uncurry* (*RETURN oo insert*)) ∈
[λ(*i*, *xs*). *i* ∈# *A*$_{in}$ ∧ *isasat-input-bounded A*]$_f$
nat-rel ×$_r$ *distinct-atoms-rel A*$_{in}$ → ⟨*distinct-atoms-rel A*$_{in}$⟩*nres-rel*›
⟨*proof*⟩

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* = (λ*L* ((*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*) ∈ [λ*D*. *D* ≠ *None*]$_f$ *option-lookup-clause-rel* $\mathcal{A}$ →
⟨*nat-rel*⟩*nres-rel*›
⟨*proof*⟩

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 ×$_f$ (*Id*) →$_f$
⟨(*Id*)⟩*nres-rel*›
⟨*proof*⟩

lemma *isa-vmtf-unset-isa-vmtf*:
assumes ‹*vm* ∈ *vmtf* $\mathcal{A}$ *M*› **and** ‹*L* ∈# $\mathcal{A}$›
shows ‹*isa-vmtf-unset L vm* ∈ *vmtf* $\mathcal{A}$ *M*›
⟨*proof*⟩

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*⟩

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* ›
⟨*proof*⟩

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*⟩

definition *vmtf-rescore-body*
:: ‹*nat multiset* ⇒ *nat clause-l* ⇒ (*nat*,*nat*) *ann-lits* ⇒ *bump-heuristics* ⇒
(*nat* × *bump-heuristics*) *nres*›
where
‹*vmtf-rescore-body* $\mathcal{A}_{in}$ *C - vm* = *do* {
WHILE$_T$$^{\lambda(i,\ vm).\ i \le length\ C}$
(λ(*i*, *vm*). *i* < *length C*)
(λ(*i*, *vm*). *do* {
ASSERT(*i* < *length C*);
ASSERT(*atm-of* (*C*!*i*) ∈# $\mathcal{A}_{in}$);
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 𝒜 C M vm = SPEC (λ(vm'). vm' ∈ bump-heur 𝒜 M)›

lemma isa-bump-mark-to-rescore:
  assumes
    ‹L ∈# 𝒜› ‹b ∈ bump-heur 𝒜 M› ‹length (fst (get-bumped-variables b)) < Suc (Suc (unat32-max div
2))›
  shows ‹
     isa-bump-mark-to-rescore L b ≤ SPEC (λvm'. vm' ∈ bump-heur 𝒜 M)›
⟨proof⟩

lemma length-get-bumped-variables-le:
  assumes ‹vm ∈ bump-heur 𝒜 M› ‹isasat-input-bounded 𝒜›
  shows ‹length (fst (get-bumped-variables vm)) < Suc (Suc (unat32-max div 2))›
  ⟨proof⟩

lemma vmtf-rescore-score-clause:
  ‹(uncurry2 (vmtf-rescore 𝒜), uncurry2 (isa-rescore-clause 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⟩

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⟩

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⟩

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)
(λi 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)
(λi 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›
⟨proof⟩

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*)›
⟨*proof*⟩

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*)
(λ*i 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* $\mathcal{A}$)) ∈
[λ-. *isasat-input-bounded* $\mathcal{A}$]$_f$
trail-pol $\mathcal{A}$ ×$_f$ *Id* ×$_f$ *Id* ×$_f$ *Id* ×$_f$ *Id* → ⟨*Id*⟩*nres-rel*›

⟨*proof*⟩

lemma *vmtf-mark-to-rescore-also-reasons-spec*:
‹*vm* ∈ *bump-heur* $\mathcal{A}$ *M* ⟹ *valid-arena arena N vdom* ⟹ *length outl* ≤ *unat32-max* ⟹ *isasat-input-bounded* $\mathcal{A}$ ⟹
(∀ *L* ∈ *set outl*. *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$) ⟹
(∀ *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* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$)))) ⟹
vmtf-mark-to-rescore-also-reasons $\mathcal{A}$ *M arena outl L vm* ≤ *RES* (*bump-heur* $\mathcal{A}$ *M*)›
⟨*proof*⟩

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* $\mathcal{A}$ *M arena C L vm* = *do* {
ASSERT(*arena-is-valid-clause-idx arena C*);
nfoldli
([*0*..<*arena-length arena C*])
(λ-. *True*)
(λ*i vm*. *do* {
K ← *mop-arena-lit2 arena C i*;
ASSERT(−*K* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$);
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* $\mathcal{A}$ *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*)
(λ*i 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* $\mathcal{A}$)) ∈
[λ-. *isasat-input-bounded* $\mathcal{A}$]$_f$
trail-pol $\mathcal{A}$ ×$_f$ *Id* ×$_f$ *Id* ×$_f$ *Id* ×$_f$ (*Id*) → ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

lemma *arena-lifting-list*:
‹*valid-arena arena N vdom* ⟹ *C* ∈# *dom-m N* ⟹
N ∝ *C* = *map* (λ*i*. *arena-lit arena* (*C*+*i*)) [*0*..<*arena-length arena C*]›
⟨*proof*⟩

lemma *vmtf-mark-to-rescore-also-reasons-cl-spec*:
‹*vm* ∈ *bump-heur* $\mathcal{A}$ *M* ⟹ *valid-arena arena N vdom* ⟹ *C* ∈# *dom-m N* ⟹ *isasat-input-bounded* $\mathcal{A}$ ⟹
(∀ *L* ∈ *set* (*N* ∝ *C*). *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$) ⟹
(∀ *L* ∈ *set* (*N* ∝ *C*). ∀ *C*. (*Propagated* (−*L*) *C* ∈ *set M* ⟶ *C* ≠ *0* ⟶ (*C* ∈# *dom-m N* ∧
(∀ *C* ∈ *set* [*C*..<*C* + *arena-length arena C*]. *arena-lit arena C* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$)))) ⟹
vmtf-mark-to-rescore-also-reasons-cl $\mathcal{A}$ *M arena C L vm* ≤ *RES* (*bump-heur* $\mathcal{A}$ *M*)›
⟨*proof*⟩

13.3 Backtrack level for Restarts

hide-const (**open**) *find-decomp-wl-imp*

lemma *isa-bump-unset-pre*:
assumes
‹*x* ∈ *bump-heur* $\mathcal{A}$ *M*› **and**
‹*L* ∈# $\mathcal{A}$›
shows ‹*isa-bump-unset-pre L x*›
⟨*proof*⟩

definition *isa-acids-flush-int* :: ‹*trail-pol* ⇒ (*nat*, *nat*)*acids* ⇒ - ⇒ ((*nat*, *nat*)*acids* × -) *nres*› **where**
‹*isa-acids-flush-int* = (λ*M vm* (*to-remove*, *h*). *do* {
ASSERT(*length to-remove* ≤ *unat32-max*);
(-, *vm*, *h*) ← *WHILE*$_T$λ(*i*, *vm*′, *h*). *i* ≤ *length to-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* $\mathcal{A}$)) ∈ *trail-pol* ($\mathcal{A}$::*nat multiset*) ×$_f$ *Id* ×$_f$ *Id* →$_f$ ⟨*Id*×$_f$ *Id*⟩*nres-rel*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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* 𝒜$_{in}$ = (λ*M vm*. *SPEC* (λ*x*. *x* ∈ *bump-heur* 𝒜$_{in}$ *M*))›

lemma *in-distinct-atoms-rel-in-atmsD*: ‹(*ba*, *y*) ∈ *distinct-atoms-rel* 𝒜 ⟹
xa ∈ *set* (*fst ba*) ⟹ *xa* ∈# 𝒜› **and**
distinct-atoms-rel-emptiedI: ‹((*ae*, *baa*), {}) ∈ *distinct-atoms-rel* 𝒜 ⟹
((*ae*, *baa*), *set ae*) ∈ *distinct-atoms-rel* 𝒜›
⟨*proof*⟩

lemma *acids-change-to-remove-order′*:
‹(*uncurry2* (*acids-flush-int* 𝒜$_{in}$), *uncurry2* (*acids-flush* 𝒜$_{in}$)) ∈
[λ((*M*, *vm*), *to-r*). *vm* ∈ *acids* 𝒜$_{in}$ *M* ∧ *isasat-input-bounded* 𝒜$_{in}$ ∧ *isasat-input-nempty* 𝒜$_{in}$ ∧ *to-r* ⊆ *set-mset* 𝒜$_{in}$]$_f$
Id ×$_f$ *Id* ×$_f$ *distinct-atoms-rel* 𝒜$_{in}$ → ⟨(*Id* ×$_r$ *distinct-atoms-rel* 𝒜$_{in}$)⟩ *nres-rel*›
⟨*proof*⟩

lemma *isa-bump-heur-flush-isa-bump-flush*:
‹(*uncurry* (*isa-bump-heur-flush*), *uncurry* (*isa-bump-flush* 𝒜))
∈ [λ(*M*, *vm*). *vm* ∈ *bump-heur* 𝒜 *M* ∧ *isasat-input-bounded* 𝒜 ∧
isasat-input-nempty 𝒜]$_f$ *trail-pol* 𝒜 ×$_f$ *Id* → ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

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* 𝒜 = (λ((*M*, *highest*), *vm*).
no-dup M ∧
highest < *count-decided M* ∧
isasat-input-bounded 𝒜 ∧
*literals-are-in-*ℒ$_{in}$*-trail* 𝒜 *M* ∧
vm ∈ *bump-heur* 𝒜 *M*)›

definition *find-decomp-wl-imp*
:: ‹*nat multiset* ⇒ (*nat*, *nat*) *ann-lits* ⇒ *nat* ⇒ *bump-heuristics* ⇒
(*nat*, *nat*) *ann-lits* × *bump-heuristics*) *nres*›
where

```
‹find-decomp-wl-imp 𝒜 = (λM₀ lev vm. do {
  let k = count-decided M₀;
  let M₀ = trail-conv-to-no-CS M₀;
  let n = length M₀;
  pos ← get-pos-of-level-in-trail M₀ lev;
  ASSERT((n − pos) ≤ unat32-max);
  ASSERT(n ≥ pos);
  let target = n − pos;
  (-, M, vm′) ←
   WHILE_T^λ(j, M, vm′). j ≤ target ∧          M = drop j M₀ ∧ target ≤ length M₀ ∧          vm′ ∈ bump-heur 𝒜 M ∧ 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 ∈# 𝒜);
        vm ←isa-bump-unset L vm;
        RETURN (j + 1, tl M, vm)
      })
      (0, M₀, vm);
  ASSERT(lev = count-decided M);
  let M = trail-conv-back lev M;
  RETURN (M, vm′)
})›
```

definition *isa-find-decomp-wl-imp*
:: ‹*trail-pol* ⇒ *nat* ⇒ *bump-heuristics* ⇒ (*trail-pol* × *bump-heuristics*) *nres*›
where

```
‹isa-find-decomp-wl-imp = (λM₀ lev vm. do {
  let k = count-decided-pol M₀;
  let M₀ = trail-pol-conv-to-no-CS M₀;
  ASSERT(isa-length-trail-pre M₀);
  let n = isa-length-trail M₀;
  pos ← get-pos-of-level-in-trail-imp M₀ lev;
  ASSERT((n − pos) ≤ unat32-max);
  ASSERT(n ≥ pos);
  let target = n − pos;
  (-, M, vm′) ←
    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, M₀, vm);
  M ← trail-conv-back-imp lev M;
  RETURN (M, vm′)
})›
```

abbreviation *find-decomp-w-ns-prop* **where**
‹*find-decomp-w-ns-prop* $\mathcal{A} \equiv$
(λ(*M*::(*nat*, *nat*) *ann-lits*) *highest* -.
(λ(*M1*, *vm*). $\exists$ *K M2*. (*Decided K* # *M1*, *M2*) $\in$ *set* (*get-all-ann-decomposition M*) $\wedge$
get-level M K = *Suc highest* $\wedge$ *vm* $\in$ *bump-heur* $\mathcal{A}$ *M1*))›

definition *find-decomp-w-ns* **where**
‹*find-decomp-w-ns* $\mathcal{A}$ =
(λ(*M*::(*nat*, *nat*) *ann-lits*) *highest vm*.
SPEC(*find-decomp-w-ns-prop* $\mathcal{A}$ *M highest vm*))›

lemma *isa-find-decomp-wl-imp-find-decomp-wl-imp*:
‹(*uncurry2 isa-find-decomp-wl-imp*, *uncurry2* (*find-decomp-wl-imp* $\mathcal{A}$)) $\in$
[λ((*M*, *lev*), *vm*). *lev* < *count-decided M*]$_f$ *trail-pol* $\mathcal{A}$ $\times_f$ *nat-rel* $\times_f$ *Id* $\to$
⟨*trail-pol* $\mathcal{A}$ $\times_r$ (*Id*)⟩*nres-rel*›
⟨*proof*⟩

definition (**in** −) *find-decomp-wl-st* :: ‹*nat literal* ⇒ *nat twl-st-wl* ⇒ *nat twl-st-wl nres*› **where**
‹*find-decomp-wl-st* = (λ*L* (*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* = (λ*highest 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* $\in$ *bump-heur* $\mathcal{A}$ M_0› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ M_0› **and**
target: ‹*highest* < *count-decided* M_0› **and**
n-d: ‹*no-dup* M_0› **and**
bounded: ‹*isasat-input-bounded* $\mathcal{A}$› **and**
count:‹*count-decided* M_0 > *0*›
shows
find-decomp-wl-imp-le-find-decomp-wl':
‹*find-decomp-wl-imp* $\mathcal{A}$ M_0 *highest vm* ≤ *find-decomp-w-ns* $\mathcal{A}$ M_0 *highest vm*›
(**is** *?decomp*)
⟨*proof*⟩

lemma *find-decomp-wl-imp-find-decomp-wl'*:
‹(*uncurry2* (*find-decomp-wl-imp* $\mathcal{A}$), *uncurry2* (*find-decomp-w-ns* $\mathcal{A}$)) $\in$
[*find-decomp-w-ns-pre* $\mathcal{A}$]$_f$ *Id* $\times_f$ *Id* $\times_f$ *Id* $\to$ ⟨*Id* $\times_f$ *Id*⟩*nres-rel*›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*›
⟨*proof*⟩

interpretation *VMTF*: *weak-ordering-on-lt* **where**
C = ‹*valid-atoms vs*› **and**
less = ‹*VMTF-score-less vs*›
⟨*proof*⟩

interpretation *VMTF*: *parameterized-weak-ordering valid-atoms VMTF-score-less*
mop-VMTF-score-less
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*

⟨*proof*⟩

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*›
⟨*proof*⟩

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} →$_a$ *VMTF-it.arr-assn*›
⟨*proof*⟩

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*›
:: ‹$(\textit{vmtf-assn})^k *_a$ *VMTF-it.arr-assn*$^d \to_a$ *VMTF-it.arr-assn*›
⟨*proof*⟩

sepref-def *atoms-hash-del-code*
is ‹*uncurry* (*RETURN oo atoms-hash-del*)›
:: ‹$[\textit{uncurry atoms-hash-del-pre}]_a$ *atom-assn*$^k *_a$ (*atoms-hash-assn*)$^d \to$ *atoms-hash-assn*›
⟨*proof*⟩

sepref-def *atoms-hash-insert-code*
is ‹*uncurry* (*RETURN oo atoms-hash-insert*)›
:: ‹$[\textit{uncurry atms-hash-insert-pre}]_a$
atom-assn$^k *_a$ (*distinct-atoms-assn*)$^d \to$ *distinct-atoms-assn*›
⟨*proof*⟩

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*)›
:: ‹$(\textit{atom.option-assn})^k *_a$ *vmtf-assn*$^d \to_a$ *vmtf-assn*›
⟨*proof*⟩

lemma *case-option-split*:
‹(*case a of None* ⇒ *x* | *Some y* ⇒ *f y*) =
(*if is-None a then x else let y* = *the a in f y*)›
⟨*proof*⟩

sepref-def *ns-vmtf-dequeue-code*
is ‹*uncurry* (*RETURN oo ns-vmtf-dequeue*)›
:: ‹$[\textit{vmtf-dequeue-pre}]_a$
atom-assn$^k *_a$ (*array-assn vmtf-node-assn*)$^d \to$ *array-assn vmtf-node-assn*›
⟨*proof*⟩

sepref-register *get-next get-prev stamp*
lemma *eq-Some-iff*: ‹$x = \textit{Some } b \longleftrightarrow (\neg \textit{is-None } x \wedge \textit{the } x = b)$›
⟨*proof*⟩

lemma *hfref-refine-with-pre*:
assumes ‹$\bigwedge x.\ P\ x \Longrightarrow g'\ x \leq g\ x$›
assumes ‹$(f,g') \in [P]_a\ A \to R$›
shows ‹$(f,g) \in [P]_a\ A \to R$›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *isa-vmtf-en-dequeue-alt-def2*:
‹*isa-vmtf-en-dequeue-pre x* ⟹ *uncurry2* (λ*M 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*
›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›

⟨*proof*⟩

sepref-register *vmtf-rescale*
sepref-def *vmtf-rescale-code*
is ‹*vmtf-rescale*›
:: ‹*vmtf-assn*^{d} →$_a$ *vmtf-assn*›
⟨*proof*⟩

sepref-register *partition-between-ref*

sepref-register *isa-vmtf-enqueue*

lemma *emptied-list-alt-def*: ‹*emptied-list xs* = *take 0 xs*›
⟨*proof*⟩

sepref-def *current-stamp-impl*
is ‹*RETURN o current-stamp*›
:: ‹*vmtf-assn*^{k} →$_a$ *uint64-nat-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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›
⟨*proof*⟩

lemma *isa-vmtf-bump-to-rescore-also-reasons-clD*:
‹*arena-is-valid-clause-idx arena C* ⟹ *C* + *arena-length arena C* ≤ *length arena*›
⟨*proof*⟩

schematic-goal *mk-free-vmtf-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE vmtf-assn ?fr*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *isa-acids-flush-int-alt-def*:
‹*isa-acids-flush-int* = (λ*M vm* (*to-remove*, *h*). *do* {
ASSERT(*length to-remove* ≤ *unat32-max*);
let n = *length to-remove*;
(-, *vm*, *h*) ← *WHILE*$_T$$^{\lambda(i,\ vm',\ h).\ i \le 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*))

})›
⟨proof⟩

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} $\rightarrow_a$ *acids-assn2* $\times_a$ *distinct-atoms-assn*
›
⟨proof⟩

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⟩

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*)›
⟨proof⟩

sepref-def *hp-acids-empty-code*
is ‹*RETURN o hp-acids-empty*›
:: ‹*hp-assn*^{k} $\rightarrow_a$ *bool1-assn*›
⟨proof⟩

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} $\rightarrow_a$ *bool1-assn*›
⟨proof⟩

sepref-def *acids-find-next-undef-impl*
is ‹*uncurry isa-acids-find-next-undef*›
:: ‹*acids-assn2*^{d} $*_a$ *trail-pol-fast-assn*^{k} $\rightarrow_a$ *atom.option-assn* $\times_a$ *acids-assn2*›
⟨proof⟩

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)
})›
⟨*proof*⟩

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} → *heuristic-bump-assn*›
⟨*proof*⟩

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} →$_a$
trail-pol-fast-assn ×$_a$ *heuristic-bump-assn*›
⟨*proof*⟩

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} →$_a$ *heuristic-bump-assn*›
⟨*proof*⟩

sepref-def *isa-bump-mark-to-rescore-clause-fast-code*
is ‹*uncurry2 (isa-bump-mark-to-rescore-clause)*›
:: ‹[*λ((N, -), -). length N ≤ snat64-max*]$_a$
arena-fast-assn^k $*_a$ *sint64-nat-assn*^{k} $*_a$ *heuristic-bump-assn*^{d} → *heuristic-bump-assn*›
⟨*proof*⟩

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} →$_a$
heuristic-bump-assn›
⟨*proof*⟩

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›
⟨*proof*⟩

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›
⟨*proof*⟩

sepref-def *isa-bump-heur-flush-impl*
is ‹*uncurry isa-bump-heur-flush*›
:: ‹*trail-pol-fast-assn*^{k} $*_a$ *heuristic-bump-assn*^{d} $\rightarrow_a$ *heuristic-bump-assn*›
⟨*proof*⟩

sepref-def *isa-vmtf-heur-fst-code*
is ‹*isa-vmtf-heur-fst*›
:: ‹*heuristic-bump-assn*^{k} $\rightarrow_a$ *atom-assn*›
⟨*proof*⟩

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*))›
⟨*proof*⟩

sepref-register *is-focused-heuristics vmtf-heur-array-nth*
sepref-def *isa-vmtf-heur-array-nth-code*
is ‹*uncurry isa-vmtf-heur-array-nth*›
:: ‹[λ(*vm*, *i*). *i* < *length* (*fst* (*bump-get-heuristics vm*))]$_a$ *heuristic-bump-assn*^{k} $*_a$ *atom-assn*^{k} → *vmtf-node-assn*›
⟨*proof*⟩

definition *vmtf-array-fst* :: ‹*vmtf* ⇒ *nat*› **where**
‹*vmtf-array-fst* = (λ(*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* ⇒
(*vmtf-array-fst b*))›
⟨*proof*⟩

sepref-def *vmtf-array-fst-code*
is ‹*RETURN o vmtf-array-fst*›
:: ‹*vmtf-assn*^{k} $\rightarrow_a$ *atom-assn*›
⟨*proof*⟩

sepref-def *bumped-vmtf-array-fst-code*
is ‹*RETURN o bumped-vmtf-array-fst*›
:: ‹*heuristic-bump-assn*^{k} $\rightarrow_a$ *atom-assn*›
⟨*proof*⟩

sepref-register *access-focused-vmtf-array*

definition *access-vmtf-array* :: ‹*vmtf* ⇒ *nat* ⇒ - *nres*› **where**
‹*access-vmtf-array* = (λ(*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* ⇒ *do* {
access-vmtf-array b i
})›
⟨*proof*⟩

sepref-def *access-vmtf-array-code*
is ‹*uncurry access-vmtf-array*›
:: ‹*vmtf-assn*^{k} $*_a$ *atom-assn*^{k} $\rightarrow_a$ *vmtf-node-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*})›
⟨*proof*⟩

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} *→*$_a$ *sint64-nat-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*
)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)
}›
⟨*proof*⟩

sepref-def *mop-isasat-length-trail-st-code*
is ‹*mop-isasat-length-trail-st*›
:: ‹*isasat-bounded-assn*^{k} →$_a$ *sint64-nat-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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} *$\to_a$ status-impl-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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} *$\to_a$ sint64-nat-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*
})›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *id-clvls-assn*: ‹(*Mreturn*, *RETURN*) ∈ (*uint32-nat-assn*)^k *→*$_a$ *uint32-nat-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*
)›
⟨*proof*⟩

sepref-def *clss-size-resetUS0-st*
is ‹*RETURN o clss-size-resetUS0-st*›
:: ‹*isasat-bounded-assn*^{d} *→*$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

lemma *length-ll*[*def-pat-rules*]: ‹*length-ll*$*xs*$*i* ≡ *op-list-list-llen*$*xs*$*i*›
⟨*proof*⟩

sepref-def *length-watchlist-impl*
is ‹*uncurry* (*RETURN oo length-watchlist*)›

:: ‹[uncurry (λS L. nat-of-lit L < length S)]$_a$ watchlist-fast-assn^k *$_a$ unat-lit-assn^k → sint64-nat-assn›
⟨proof⟩

definition *length-ll-fs-heur-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*length-ll-fs-heur-fast-code* = (λN C. *read-watchlist-wl-heur-code* (λN. *length-watchlist-impl N C*) N)›
global-interpretation *watched-by-app*: *read-watchlist-param-adder* **where**
R = ‹*unat-lit-rel*› **and**
f = ‹λC N. *length-watchlist-impl N C*› **and**
f' = ‹λC N. (*RETURN oo length-watchlist*) N C› **and**
x-assn = *sint64-nat-assn* **and**
P = ‹(λL S. *nat-of-lit L < length* (*S*))›
rewrites
‹(λN C'. *read-watchlist-wl-heur* (λN. (*RETURN* ∘∘ *length-watchlist*) N C') N) = *RETURN oo length-ll-fs-heur*› **and**
‹(λN C. *read-watchlist-wl-heur-code* (λN. *length-watchlist-impl N C*) N) = *length-ll-fs-heur-fast-code*› **and**
‹*watched-by-app.XX.mop* = *mop-length-watched-by-int*›
⟨proof⟩

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*›
⟨proof⟩

definition *get-bump-heur-array-nth-impl* **where**
‹*get-bump-heur-array-nth-impl N C'* = *read-vmtf-wl-heur-code* (λM. *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*))›
⟨proof⟩

global-interpretation *vmtf-array-nth*: *read-vmtf-param-adder* **where**
f' = ‹λa b. *isa-vmtf-heur-array-nth b a*› **and**
f = ‹λa b. *isa-vmtf-heur-array-nth-code b a*› **and**
x-assn = *vmtf-node-assn* **and**
P = ‹(λn 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*›
⟨proof⟩

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*)
}›
⟨*proof*⟩

sepref-def *mop-mark-added-heur-iml*
is ‹*uncurry2 mop-mark-added-heur*›
:: ‹*atom-assn*^{k} *∗*$_a$ *bool1-assn*^{k} *∗*$_a$ *heuristic-assn*^{d} *→*$_a$ *heuristic-assn*›
⟨*proof*⟩

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} *→*$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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} $\rightarrow_a$ *unat-lit-assn*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

lemma *trail-nth-precond-simp*: ‹(λ*M*. *fst M* ≠ []) = (λ(*M*,-). *M* ≠ [])›
  ⟨*proof*⟩
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*›
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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)›
⟨*proof*⟩

sepref-def *heuristic-reluctant-untrigger-st-impl*
is ‹*RETURN o heuristic-reluctant-untrigger-st*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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
})›
⟨*proof*⟩

sepref-def *incr-restart-stat-fast-code*
is ‹*incr-restart-stat*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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
})›
⟨*proof*⟩

sepref-def *incr-reduction-stat-fast-code*
is ‹*incr-reduction-stat*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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})›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*
)›
⟨*proof*⟩

sepref-def *empty-US-heur-code*
is ‹*RETURN o empty-US-heur*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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*))})›
⟨*proof*⟩

lemma *mark-garbage-preD*:
‹*mark-garbage-pre* (*N*, *C*) ⟹ *arena-is-valid-clause-vdom N C*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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

}›
⟨proof⟩

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*›
⟨proof⟩

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*›
⟨proof⟩

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**
‹(λ*N C*. *read-vdom-wl-heur-code* (λ*N*. *avdom-aivdom-at-impl N C*) *N*) = *access-avdom-at-fast-code*› **and**
‹(λ*S C*. *C* < *length* (*get-avdom-aivdom* (*get-aivdom S*))) = *access-avdom-at-pre*›
⟨proof⟩

definition *access-ivdom-at-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*access-ivdom-at-fast-code* = (λ*N C*. *read-vdom-wl-heur-code* (λ*N*. *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′ = ‹λ*C N*. (*RETURN oo ivdom-aivdom-at*) *N C*› **and**
f = ‹λ*C N*. *ivdom-aivdom-at-impl N C*› **and**
P = ‹λ*C N*. *C* < *length* (*get-ivdom-aivdom N*)›
rewrites
‹(λ*N C*′. *read-vdom-wl-heur* (λ*N*. (*RETURN* ∘∘ *ivdom-aivdom-at*) *N C*′) *N*) = *RETURN oo access-ivdom-at*› **and**
‹(λ*N C*. *read-vdom-wl-heur-code* (λ*N*. *ivdom-aivdom-at-impl N C*) *N*) = *access-ivdom-at-fast-code*› **and**
‹(λ*S C*. *C* < *length* (*get-ivdom-aivdom* (*get-aivdom S*))) = *access-ivdom-at-pre*›
⟨proof⟩

definition *access-tvdom-at-fast-code* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*access-tvdom-at-fast-code* = (λ*N C*. *read-vdom-wl-heur-code* (λ*N*. *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' = ‹λ*C N*. (*RETURN oo tvdom-aivdom-at*) *N C*› **and**
f = ‹λ*C N*. *tvdom-aivdom-at-impl N C*› **and**
P = ‹λ*C N*. *C* < *length* (*get-tvdom-aivdom N*)›
rewrites
‹(λ*N C'*. *read-vdom-wl-heur* (λ*N*. (*RETURN* ∘∘ *tvdom-aivdom-at*) *N C'*) *N*) = *RETURN oo access-tvdom-at*› **and**
‹(λ*N C*. *read-vdom-wl-heur-code* (λ*N*. *tvdom-aivdom-at-impl N C*) *N*) = *access-tvdom-at-fast-code*› **and**
‹(λ*S C*. *C* < *length* (*get-tvdom-aivdom* (*get-aivdom S*))) = *access-tvdom-at-pre*›
⟨*proof*⟩

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} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-def *print-encoded-lit-end-code*
is *print-literal-of-trail*
:: ‹*uint32-nat-assn*^{k} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-def *print-trail-code*
is ‹*print-trail*›
:: ‹*trail-pol-fast-assn*^{k} →$_a$ *unit-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*›

⟨*proof*⟩

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′*))›
⟨*proof*⟩
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*›
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *mop-is-marked-added-heur-st-alt-def*:
‹*is-marked-added.XX.mop* = *mop-is-marked-added-heur-st*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
  ⟨*proof*⟩

sepref-def *get-fast-ema-heur-full-impl*
  is ‹*RETURN o get-fast-ema-heur-full*›
  :: ‹*heuristic-assn*^{k} →$_a$ *word64-assn*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

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*›
⟨*proof*⟩
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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *length-occs-at-alt-def*:
‹*length-occs-at = length-occs-at.XX.mop*›

⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *get-occs-list-at-alt-def*: ‹*get-occs-list-at* = (λ*N C D*. *occs-at-at.XX.XX.mop N* (*C*, *D*))›
⟨*proof*⟩

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*))
}›
⟨*proof*⟩

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*›

⟨*proof*⟩

sepref-def *get-lsize-limit-stats-impl*
is ‹*RETURN o get-lsize-limit-stats*›
:: ‹*isasat-stats-assn*^{k} →$_a$ *lbd-size-limit-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)
)›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›

⟨*proof*⟩

sepref-def *stats-forward-rounds-st-impl*
is ‹*RETURN o stats-forward-rounds-st*›
:: ‹*isasat-bounded-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

sepref-def *incr-purelit-rounds-st-impl*
is ‹*RETURN o incr-purelit-rounds-st*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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} →$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

sepref-def *maybe-mark-added-clause-heur2-impl*
is ‹*uncurry maybe-mark-added-clause-heur2*›
:: ‹*isasat-bounded-assn*^{d} ×$_a$ *sint64-nat-assn*^{k} →$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

experiment begin
lemma *from-bool1*: *from-bool True* = *1*
⟨*proof*⟩
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))›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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))›
  ⟨proof⟩


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)›
  ⟨proof⟩

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›
  ⟨proof⟩

sepref-def print-uint64-impl
  is ‹RETURN o print-uint64›
  :: ‹word-assn^k →_a unit-assn›
  ⟨proof⟩

sepref-def print-open-colour-impl
  is ‹RETURN o print-open-colour›
  :: ‹word-assn^k →_a unit-assn›
  ⟨proof⟩
```

sepref-def *print-close-colour-impl*
  is ‹*RETURN o print-close-colour*›
  :: ‹*word-assn*^{k} →$_a$ *unit-assn*›
  ⟨*proof*⟩

sepref-def *print-char-impl*
  is ‹*RETURN o print-char*›
  :: ‹*word-assn*^{k} →$_a$ *unit-assn*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

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*)))))›
  ⟨*proof*⟩

sepref-def *isasat-current-status-fast-code*
  is ‹*isasat-current-status*›
  :: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
  ⟨*proof*⟩

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 ())›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

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} $\rightarrow_a$ *heuristic-int-assn*›
⟨*proof*⟩

sepref-register *save-rephase-heur-stats*
sepref-def *save-phase-heur-impl*
is ‹*uncurry save-rephase-heur*›
:: ‹*word64-assn*^{k} $*_a$ *heuristic-assn*^{d} $\rightarrow_a$ *heuristic-assn*›
⟨*proof*⟩

lemma *save-phase-st-alt-def*:
‹*save-phase-st* = (λ*S*. *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*))
})›
⟨*proof*⟩

sepref-def *save-phase-heur-st*
is *save-phase-st*
:: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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)))
  })›
 ⟨proof⟩

sepref-register rephase-heur
sepref-def rephase-heur-st-impl
 is rephase-heur-st
 :: ‹isasat-bounded-assn^d →_a isasat-bounded-assn›
 ⟨proof⟩


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)
  (λi 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›
 ⟨proof⟩

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-L_in A (mset (N' ∝ C))›
 shows ‹mark-lbd-from-clause-heur M N C lbd ≤ ⇓ Id (SPEC(λ-::bool list. True))›
 ⟨proof⟩

definition calculate-LBD-st :: ‹(nat, nat) ann-lits ⇒ nat clauses-l ⇒ nat ⇒ nat clauses-l nres› where
 ‹calculate-LBD-st = (λM 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*›)
⟨*proof*⟩

lemma *RF-COME-ON*: ‹(*x*, *y*) ∈ *Id* ⟹ *f x* ≤ ⇓ *Id* (*f y*)›
⟨*proof*⟩

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*})›
⟨*proof*⟩

lemma *mop-arena-mark-used-valid*:
‹$C \in\# \mathit{dom\text{-}m}\ N \Longrightarrow \mathit{valid\text{-}arena}\ \mathit{arena}\ N\ \mathit{vdom} \Longrightarrow$
$\mathit{mop\text{-}arena\text{-}mark\text{-}used}\ \mathit{arena}\ C \leq \mathit{SPEC}(\lambda c.\ (c,\ N) \in \{(c,\ N').\ N'{=}N \wedge \mathit{valid\text{-}arena}\ c\ N\ \mathit{vdom} \wedge$
$\mathit{length}\ c = \mathit{length}\ \mathit{arena}\})$›
⟨*proof*⟩

lemma *mop-arena-mark-used2-valid*:
‹$C \in\# \mathit{dom\text{-}m}\ N \Longrightarrow \mathit{valid\text{-}arena}\ \mathit{arena}\ N\ \mathit{vdom} \Longrightarrow$
$\mathit{mop\text{-}arena\text{-}mark\text{-}used2}\ \mathit{arena}\ C \leq \mathit{SPEC}(\lambda c.\ (c,\ N) \in \{(c,\ N').\ N'{=}N \wedge \mathit{valid\text{-}arena}\ c\ N\ \mathit{vdom} \wedge$
$\mathit{length}\ c = \mathit{length}\ \mathit{arena}\})$›
⟨*proof*⟩

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* $\mathcal{A}$›
‹*C* ∈# *dom-m N*›
‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*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*⟩

definition *mark-lbd-from-list* :: ‹-› **where**
‹*mark-lbd-from-list M C lbd* = *do* {
nfoldli (*drop 1 C*) (λ-. *True*)
(λ*L 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*)
(λ*i 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* = (λ*S*. *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 ‹$(M,\ M') \in$ *trail-pol* $\mathcal{A}$› **and** ‹*literals-are-in-*$\mathcal{L}_{in}$ $\mathcal{A}$ (*mset* (*tl* C))›
shows ‹*mark-lbd-from-list-heur* M C *lbd* $\leq \Downarrow$ *Id* (*SPEC*(λ-::*bool list*. *True*))›
⟨*proof*⟩

definition *mark-LBD-st* :: ‹$'v$ *twl-st-wl* ⇒ ($'v$ *twl-st-wl*) *nres*› **where**
‹*mark-LBD-st* = (λS. *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). $S = T$)}›
⟨*proof*⟩

lemma *mark-lbd-from-conflict-mark-LBD-st*:
‹(*mark-lbd-from-conflict*, *mark-LBD-st*) ∈
[λS. *get-conflict-wl* $S \neq$ *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*›
⟨*proof*⟩

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›
⟨*proof*⟩

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*:
‹$C \in\#$ *dom-m* $N \Longrightarrow$ *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*})›
⟨*proof*⟩

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* ⇒ *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 = (λS i. do {
     isa-find-unwatched (λL. 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*
(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
})
(*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 S*$_0$) ∧
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 S*$_0$ = *do* {
let j1 = *isa-length-trail* (*get-trail-wl-heur S*$_0$);
- ← *RETURN* (*IsaSAT-Profile.start-propagate*);
S ← *WHILE*$_T$$^{\textit{unit-propagation-outer-loop-wl-D-heur-inv}}$ *S*$_0$
(λ*S*. *literals-to-update-wl-heur S* < *isa-length-trail* (*get-trail-wl-heur S*))
(λ*S*. *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$_0$;
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* (λ*L*. *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*}›
  ⟨*proof*⟩

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*⟩

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*⟩

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
}
}
›
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

14.2 Updates

lemma *isa-vmtf-bump-to-rescore-also-reasons-cl-isa-vmtf*:
assumes ‹(*M*,*M'*)∈*trail-pol* $\mathcal{A}$› ‹*isasat-input-bounded* $\mathcal{A}$› **and** *vm*: ‹*vm* ∈ *bump-heur* $\mathcal{A}$ *M'*› **and**
valid: ‹*valid-arena N N' vd*› **and**
C: ‹*C* ∈# *dom-m N'*› **and**
H:‹ ∀ *L*∈*set* (*N'* ∝ *C*). *L* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$›
‹∀ *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* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$)›
and
bound: ‹*isasat-input-bounded* $\mathcal{A}$›
shows
‹*isa-vmtf-bump-to-rescore-also-reasons-cl M N C L vm* ≤ *RES* (*bump-heur* $\mathcal{A}$ *M'*)›
⟨*proof*⟩

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*(λ*S'*. (*S'*, *T*)∈ *twl-st-heur-up''* $\mathcal{D}$ *r s K lcount*)›
⟨*proof*⟩

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*)›

⟨*proof*⟩

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*›
⟨*proof*⟩

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'*)›
⟨*proof*⟩

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*)]))))
})›
⟨*proof*⟩

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*›

⟨*proof*⟩

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 ×$_f$ *Id* ×$_f$ *nat-rel* ×$_f$ *bool-rel* ×$_f$ *nat-rel* ×$_f$ *nat-rel* ×$_f$ *nat-rel* ×$_f$ *nat-rel* ×$_f$ *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*›
⟨*proof*⟩

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''* $\mathcal{D}$ *r s K lcount* → ⟨*twl-st-heur-up''* $\mathcal{D}$ *r s K lcount*⟩*nres-rel*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)
}›
⟨*proof*⟩

lemma *pos-of-watched-heur*:
‹(*S*, *S'*) ∈ {(*T*, *T'*). *get-vdom T* = *get-vdom x2e* ∧ (*T*, *T'*) ∈ *twl-st-heur-up''* $\mathcal{D}$ *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'*)›
⟨*proof*⟩

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-wl-heur S L C i ≤ ⇓ Id (other-watched-wl S′ L′ C′ i′)›
⟨proof⟩

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*) ((λx. *mset* (*fst x*)) ‘# *ran-m* (*get-clauses-wl S*))›
⟨*proof*⟩

lemma *set-conflict-wl-alt-def*:
‹*set-conflict-wl* = (λC 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*)
})›
⟨*proof*⟩

lemma *set-conflict-wl-pre-set-conflict-wl′-pre*:
assumes ‹*set-conflict-wl-pre C S*›
shows ‹*set-conflict-wl′-pre C S*›
⟨*proof*⟩

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′′ 𝒟 r s K lcount* → ⟨*twl-st-heur-up′′ 𝒟 r s K lcount*⟩*nres-rel*›
⟨*proof*⟩

lemma *in-Id-in-Id-option-rel*[*refine*]:
‹(*f*, *f′*) ∈ *Id* ⟹ (*f*, *f′*) ∈ ⟨*Id*⟩ *option-rel*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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'*)›
⟨*proof*⟩

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*⟩

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
            }
        }
      }
    }
  }›
  ⟨proof⟩

lemma fref-to-Down-curry8:
  ‹(uncurry8 ff, uncurry8 g) ∈ [P]_f A → ⟨B⟩nres-rel ⟹
    (⋀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') ⟹
      ((((((((x, y), z), a), b), c), d), e), f), ((((((((x', y'), z'), a'), b'), c'), d'), e'), f')) ∈ A ⟹
        ff x y z a b c d e f ≤ ⇓ B (g x' y' z' a' b' c' d' e' f'))›
  ⟨proof⟩


lemma unit-propagation-inner-loop-body-wl-heur-unit-propagation-inner-loop-body-wl-D:
```

‹(*uncurry3 unit-propagation-inner-loop-body-wl-heur*,
uncurry3 unit-propagation-inner-loop-body-wl)
∈ [λ(((*L*, *i*), *j*), *S*). *length* (*watched-by S L*) ≤ *r* − *MIN-HEADER-SIZE* ∧ *L* = *K* ∧
length (*watched-by S L*) = *s*]$_f$
nat-lit-lit-rel ×$_f$ *nat-rel* ×$_f$ *nat-rel* ×$_f$ *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*›
⟨*proof*⟩

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*'' $\mathcal{D}$ *r s K lcount* →
⟨*nat-rel* ×$_r$ *nat-rel* ×$_r$ *twl-st-heur-up*'' $\mathcal{D}$ *r s K lcount*⟩*nres-rel*›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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 ×$_f$ *nat-rel* ×$_f$ *nat-lit-lit-rel* ×$_f$ *twl-st-heur*'' $\mathcal{D}$ *r lcount* → ⟨*twl-st-heur*'' $\mathcal{D}$ *r lcount*⟩*nres-rel*›
⟨*proof*⟩

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 ×$_f$ *twl-st-heur*'' $\mathcal{D}$ *r lcount* → ⟨*twl-st-heur*'' $\mathcal{D}$ *r lcount*⟩ *nres-rel*›
⟨*proof*⟩

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*'' $\mathcal{D}$*1 r1 lcount* ⟹
select-and-remove-from-literals-to-update-wl-heur x
≤ ⇓{((*S*, *L*), (*S*′, *L*′)). ((*S*, *L*), (*S*′, *L*′)) ∈ *twl-st-heur*'' $\mathcal{D}$*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*)›
⟨*proof*⟩

lemma *outer-loop-length-watched-le-length-arena*:
assumes
xa-x': ‹$(xa, x') \in$ *twl-st-heur''* $\mathcal{D}$ *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: ‹$(xb, x'a) \in \{((S, L), (S', L')).$ $((S, L), (S', L')) \in$ *twl-st-heur''* $\mathcal{D}1$ *r lcount* $\times_f$ *nat-lit-lit-rel*
$\wedge$
$S' =$ *set-literals-to-update-wl* (*literals-to-update-wl x'* $-$ $\{\#L\#\})$ *x'* $\wedge$
get-clauses-wl-heur S = get-clauses-wl-heur xa$\}$› **and**
st: ‹*x'a* $= (x1, x2)$›
‹*xb* $= (x1a, x2a)$› **and**
x2: ‹*x2* $\in\#$ *all-lits-st x1*› **and**
st': ‹$(x2, x1) = (x1b, x2b)$›
shows ‹*length* (*watched-by x2b x1b*) $\leq$ *r*$-$*MIN-HEADER-SIZE*›
⟨*proof*⟩

lemma *unit-propagation-outer-loop-wl-D-heur-alt-def*:
‹*unit-propagation-outer-loop-wl-D-heur* $S_0 =$ *do* {
$S \leftarrow$ *WHILE*$_T$*unit-propagation-outer-loop-wl-D-heur-inv* S_0
(λ*S*. *literals-to-update-wl-heur S < isa-length-trail* (*get-trail-wl-heur S*))
(λ*S*. *do* {
ASSERT(*literals-to-update-wl-heur S < isa-length-trail* (*get-trail-wl-heur S*));
$(S', L) \leftarrow$ *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*
} ›
⟨*proof*⟩

lemma *unit-propagation-outer-loop-wl-alt-def*:
‹*unit-propagation-outer-loop-wl* $S_0 =$ *do* {
$S \leftarrow$ *WHILE*$_T$*unit-propagation-outer-loop-wl-inv*
(λ*S*. *literals-to-update-wl S* $\neq \{\#\}$)
(λ*S*. *do* {
ASSERT(*literals-to-update-wl S* $\neq \{\#\}$);
$(S', L) \leftarrow$ *select-and-remove-from-literals-to-update-wl S*;
ASSERT($L \in\#$ *all-lits-st S'*);
unit-propagation-inner-loop-wl L S'
})
(S_0 :: *'v twl-st-wl*);
RETURN S}
›
⟨*proof*⟩

lemma *unit-propagation-update-statistics-twl-st-heur''*:
‹$(S, x') \in$ *twl-st-heur''* $\mathcal{D}$ *r lcount* $\Longrightarrow$
unit-propagation-update-statistics a b S $\leq \Downarrow$(*twl-st-heur''* $\mathcal{D}$ *r lcount*) (*RETURN x'*)›
⟨*proof*⟩

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*) $\in$

twl-st-heur'' $\mathcal{D}$ r lcount $\rightarrow_f$ $\langle$*twl-st-heur'' $\mathcal{D}$ r lcount*$\rangle$ *nres-rel*$\rangle$
⟨*proof*⟩

lemma *twl-st-heur'D-twl-st-heurD*:
assumes *H*: ‹($\bigwedge\mathcal{D}$. *f* ∈ *twl-st-heur' $\mathcal{D}$* $\rightarrow_f$ ⟨*twl-st-heur' $\mathcal{D}$*⟩ *nres-rel*)›
shows ‹*f* ∈ *twl-st-heur* $\rightarrow_f$ ⟨*twl-st-heur*⟩ *nres-rel*› (**is** ‹- ∈ *?A B*›)
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)))
})›

⟨*proof*⟩

sepref-def *mark-lbd-from-conflict-impl*
is ‹*mark-lbd-from-conflict*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

lemma *update-lbd-pre-arena-act-preD*:
‹*update-lbd-pre* ((*a*, *ba*), *b*) ⟹
arena-act-pre (*update-lbd a ba b*) *a*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*))
}›
⟨*proof*⟩

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} →$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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
})
›
⟨*proof*⟩

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*›
⟨*proof*⟩

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* $N \leq$ *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}
$\to$
snat-option-assn' *TYPE*(*64*)›
⟨*proof*⟩

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* = (λN C D E.
read-all-st-code
(λM 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 = ‹λC C' C'' M N. *isa-find-unwatched-between-fast-code* M N C C' C''› **and**
f' = ‹λC C' C'' M N. *isa-find-unset-lit* M N C C' C''› **and**
x-assn = ‹*snat-option-assn'* *TYPE*(*64*)› **and**
P = ‹λC C' C'' M N. *length* $N \leq$ *snat64-max*›
rewrites
‹(λN C D E.
read-all-st (λM N - - - - - - - - - - - - - - - -. *isa-find-unset-lit* M N C D E) N) = *isa-find-unset-lit-st*›
and
‹(λN 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*›
⟨*proof*⟩

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} $\rightarrow_a$ *arena-fast-assn*›
⟨*proof*⟩

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)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

lemma *other-watched-wl-heur-alt-def*:
‹*other-watched-wl-heur* = (λ*S*. *arena-other-watched* (*get-clauses-wl-heur S*))›
⟨*proof*⟩

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} $\rightarrow_a$
unat-lit-assn›
⟨*proof*⟩

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* ›
⟨*proof*⟩

lemma *snat64-max-le-max-snat64*: ‹*a* < *snat64-max* ⟹ *Suc a* < *max-snat 64* ›
⟨*proof*⟩

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*)
})›
⟨*proof*⟩

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*›
⟨*proof*⟩

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$_0$)›
⟨*proof*⟩

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*›
⟨*proof*⟩
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$_0$)›
⟨*proof*⟩

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
})›
⟨*proof*⟩

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›
⟨*proof*⟩

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*)
})›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)]›
⟨*proof*⟩

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*)
})›
⟨*proof*⟩

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} →$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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 }›
⟨*proof*⟩

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} →$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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})›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*$_T$ (λ*j*. *j* < *i*)
(λ*j*. *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⟩

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⟩


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›
  ⟨proof⟩

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›
  ⟨proof⟩

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 (λc. (c, log-clause T C') ∈ unit-rel)›
    ⟨proof⟩

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*⟩

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 ∈# $\mathcal{L}_{all}$ $\mathcal{A}$);
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-$\mathcal{L}_{in}$ $\mathcal{A}$ (mset outl)› **and**
DD': ‹(D', D) ∈ lookup-clause-rel $\mathcal{A}$› **and**
consistent: ‹¬ tautology (mset outl)› **and**
bounded: ‹isasat-input-bounded $\mathcal{A}$›
shows
‹empty-conflict-and-extract-clause-heur $\mathcal{A}$ M D' outl ≤ ⇓ (option-lookup-clause-rel $\mathcal{A}$ ×$_r$ Id ×$_r$ Id)
(empty-conflict-and-extract-clause M D outl)›
⟨proof⟩

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 $\mathcal{A}$)) ∈
trail-pol $\mathcal{A}$ ×$_f$ Id ×$_f$ Id →$_f$ ⟨Id⟩nres-rel ›
⟨proof⟩

definition *extract-shorter-conflict-wl-nlit* **where**
‹extract-shorter-conflict-wl-nlit K M NU D NE UE =
SPEC(λD'. 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 = (λM 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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *heuristic-rel-stats-update-heuristics-stats*[*intro*!]:
‹*heuristic-rel-stats* $\mathcal{A}$ *heur* ⟹ *heuristic-rel-stats* $\mathcal{A}$ (*update-propagation-heuristics-stats glue heur*)›
⟨*proof*⟩

lemma *heuristic-rel-update-heuristics*[*intro*!]:
‹*heuristic-rel* $\mathcal{A}$ *heur* ⟹ *heuristic-rel* $\mathcal{A}$ (*update-propagation-heuristics glue heur*)›
⟨*proof*⟩

lemma *valid-arena-update-lbd-and-mark-used*:
assumes *arena*: ‹*valid-arena arena N vdom*› **and** *i*: ‹*i* ∈# *dom-m N*›
shows ‹*valid-arena* (*update-lbd-and-mark-used i lbd arena*) *N vdom*›
⟨*proof*⟩

definition *remove-last*
:: ‹*nat literal* ⇒ *nat clause option* ⇒ *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*) ∈ *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* ∈# *remove1-mset* (− *lit-of* (*hd M'*)) *y*›
shows ‹*get-level a L* = *get-level M' L*›
⟨*proof*⟩

definition *del-conflict-wl* :: ‹'*v twl-st-wl* ⇒ '*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*›
⟨*proof*⟩

lemma *lcount-add-clause*[*simp*]: ‹*i* ∉# *dom-m N* ⟹
size (*learned-clss-l* (*fmupd i* (*C*, *False*) *N*)) = *Suc* (*size* (*learned-clss-l N*))›
⟨*proof*⟩

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*⟩

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*⟩

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* = (λ*highest 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
})›
⟨*proof*⟩

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›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *log-start-new-clause-impl*
is ‹*RETURN o log-start-new-clause*›
:: ‹*unat64-assn*^{k} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-def *log-start-del-clause-impl*
is ‹*RETURN o log-start-del-clause*›
:: ‹*unat64-assn*^{k} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-def *log-end-clause-impl*
is ‹*RETURN o log-end-clause*›
:: ‹*unat64-assn*^{k} →$_a$ *unit-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *log-new-clause-heur-impl*
is ‹*uncurry log-new-clause-heur*›
:: ‹*isasat-bounded-assn*^{k} *$_a$ *snat64-assn*^{k} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-def *log-del-clause-heur-impl*
is ‹*uncurry log-del-clause-heur*›
:: ‹*isasat-bounded-assn*^{k} *$_a$ *snat64-assn*^{k} →$_a$ *unit-assn*›
⟨*proof*⟩

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} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-def *log-del-binary-clause-impl*
  is ‹*uncurry* (*RETURN oo log-del-binary-clause*)›
  :: ‹*unat-lit-assn*^{k} *$*_a$ *unat-lit-assn*^{k} *$\to_a$ *unit-assn*›
  ⟨*proof*⟩

sepref-def *mark-literal-for-unit-deletion-impl*
  is ‹*RETURN o mark-literal-for-unit-deletion*›
  :: ‹*unat-lit-assn*^{k} *$\to_a$ *unit-assn*›
  ⟨*proof*⟩

sepref-def *mark-clause-for-unit-as-unchanged-impl*
  is ‹*RETURN o mark-clause-for-unit-as-unchanged*›
  :: ‹*unat64-assn*^{k} *$\to_a$ *unit-assn*›
  ⟨*proof*⟩

sepref-def *mark-clause-for-unit-as-changed-impl*
  is ‹*RETURN o mark-clause-for-unit-as-changed*›
  :: ‹*unat64-assn*^{k} *$\to_a$ *unit-assn*›
  ⟨*proof*⟩

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*))
   }›
   ⟨*proof*⟩

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*›
  ⟨*proof*⟩

lemma *emptied-list-alt-def*: ‹*emptied-list xs* = *take 0 xs*›
  ⟨*proof*⟩

sepref-def *empty-cach-code*
  is ‹*empty-cach-ref-set*›
  :: ‹*cach-refinement-l-assn*^{d} →$_a$ *cach-refinement-l-assn*›
  ⟨*proof*⟩

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*⟩

sepref-register *fm-add-new-fast*

lemma *isasat-fast-length-leD*: ‹*isasat-fast S* ⟹ *Suc* (*length* (*get-clauses-wl-heur S*)) < *max-snat 64*›
  ⟨*proof*⟩

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} →$_a$ *heuristic-int-assn*›
  ⟨*proof*⟩

sepref-def *update-heuristics-impl*
  is ‹*uncurry* (*RETURN oo update-propagation-heuristics*)›
  :: ‹*uint32-nat-assn*^{k} *$*_a$ *heuristic-assn*^{d} →$_a$ *heuristic-assn*›
  ⟨*proof*⟩

lemma *isasat-fast-countD-tmp*:
  ‹*isasat-fast S* ⟹ *clss-size-lcountUEk* (*get-learned-count S*) < *unat64-max*›
  ⟨*proof*⟩

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})›
⟨*proof*⟩

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} → *isasat-bounded-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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} $\to_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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 S*$_0$ ⟶ *append-and-length-fast-code-pre* ((*b*, *C*), *N0*));
ASSERT(*isasat-fast S*$_0$ ⟶ *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*$_0$ ⟶ *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 S*$_0$ ⟶ *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*)
})›
⟨*proof*⟩

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*›
⟨*proof*⟩

16.2 Backtrack with direct extraction of literal if highest level

lemma *le-unat32-max-div-2-le-unat32-max*: ‹*a* ≤ *unat32-max div 2* + *1* ⟹ *a* ≤ *unat32-max*›
⟨*proof*⟩

lemma *propagate-bt-wl-D-fast-code-isasat-fastI2*: ‹*isasat-fast b* ⟹
a < *length* (*get-clauses-wl-heur b*) ⟹ *a* ≤ *snat64-max*›
⟨*proof*⟩

lemma *propagate-bt-wl-D-fast-code-isasat-fastI3*: ‹*isasat-fast b* ⟹
a ≤ *length* (*get-clauses-wl-heur b*) ⟹ *a* < *snat64-max*›
⟨*proof*⟩

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*›
:: ‹[λ((*L*, *C*), *S*). *isasat-fast S*]$_a$
unat-lit-assn^k *$_a$ *clause-ll-assn*^{k} *$_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
⟨*proof*⟩

lemma *extract-shorter-conflict-list-heur-st-alt-def*:
‹*extract-shorter-conflict-list-heur-st* = (λ*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 (*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 ← *mark-lbd-from-list-heur M outl lbd*;
let D = *the-lookup-conflict bD*;
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*) *vm0*;
(*D*, *ccach*, *outl*) ← *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* ≠ [] ∧ *length outl* ≤ *unat32-max*);
(*D*, *C*, *n*) ← *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)
 })›
 ⟨proof⟩

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›
 ⟨proof⟩

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›
 ⟨proof⟩

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›
 ⟨proof⟩

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›
 ⟨proof⟩


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*: ‹(λS. True) ∘ f = (λ-. True)› ‹uncurry (λa b. True) = (λ-. True)› ‹uncurry2 (λa b c. True) = (λ-. True)›
‹case-tuple15 (λM - - - - - - - - - - - - - - -. True) = (λ-. True)›
⟨proof⟩

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›
⟨proof⟩

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›
⟨proof⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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* $\mathcal{A}$ *n* = *do* {
ASSERT ($\mathcal{A} \neq \{\#\} \longrightarrow n >$ *Max* (*insert 0* (*set-mset* $\mathcal{A}$)));
RETURN (($\mathcal{A}$, {#}, 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* $\mathcal{A}$, *init-ACIDS0* $\mathcal{A}$) ∈
Id $\rightarrow_f$ ⟨((⟨⟨*nat-rel*⟩*option-rel*, ⟨*nat-rel*⟩*option-rel*⟩*pairing-heaps-rel*)) *O*
acids-encoded-hmrel⟩*nres-rel*›
⟨*proof*⟩

definition *init-ACIDS* **where**
‹*init-ACIDS* $\mathcal{A}$ *n* = *do* {
ac ← *init-ACIDS0* $\mathcal{A}$ *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*) ← $WHILE_T$ λ-. *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*›
⟨*proof*⟩

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*⟩

lemma *initialise-ACIDS*:
shows ‹(*uncurry initialise-ACIDS*, *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* $\mathcal{A}$]$_f$
(⟨*nat-rel*⟩*list-rel-mset-rel*) ×$_f$ *nat-rel* → ⟨*Id*⟩*nres-rel*›
(**is** ‹(*?init*, *?R*) ∈ -›)
⟨*proof*⟩

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* $\mathcal{A}$]$_f$
(⟨*nat-rel*⟩*list-rel-mset-rel*) ×$_f$ *nat-rel* → ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

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* ≤ *RES* Φ
assumes ⋀*x*. *x* ∈ Φ ⟹ *f x* ≤ *M*
shows *do* { *x* ← *m*; *f x* } ≤ *M*
⟨*proof*⟩

lemma *initialize-Bump-Init*:
shows ‹(*uncurry initialize-Bump-Init*, *uncurry* (λ*N n*. *RES* (*bump-heur-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*⟩

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*)) ∈ *isa-vmtf-init* $\mathcal{A}$ *M* ⟹
((*ns*, *m*, *fst-As*, *lst-As*, *next-search*)) ∈ *isa-vmtf-init* $\mathcal{A}$ (*L* # *M*)›
⟨*proof*⟩

lemma *isa-vmtf-init-consD'*:
‹*vm* ∈ *isa-vmtf-init* $\mathcal{A}$ *M* ⟹ *vm* ∈ *isa-vmtf-init* $\mathcal{A}$ (*L* # *M*)›
⟨*proof*⟩

lemma *vmtf-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *L* ∈ *vmtf* $\mathcal{A}$ *M* ⟹ *L* ∈ *vmtf* $\mathcal{B}$ *M*›
⟨*proof*⟩
lemma *isa-vmtf-init-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *isa-vmtf-init* $\mathcal{A}$ *M* =*isa-vmtf-init* $\mathcal{B}$ *M*›
⟨*proof*⟩
lemma *isa-acids-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *acids* $\mathcal{A}$ = *acids* $\mathcal{B}$›
⟨*proof*⟩

lemma *distinct-atoms-rel-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *distinct-atoms-rel* $\mathcal{A}$ = *distinct-atoms-rel* $\mathcal{B}$›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *bump-heur-init-consD'*:
‹*vm* ∈ *bump-heur-init* $\mathcal{A}$ *M* ⟹ *vm* ∈ *bump-heur-init* $\mathcal{A}$ (*Propagated L n* # *M*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *in-mset-rel-eq-f-iff-set*:
‹⟨{(*c*, *a*). *a* = *f c*}⟩*mset-rel* = {(*b*, *a*). *a* = *f* ‘# *b*}›
⟨*proof*⟩

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*›
⟨*proof*⟩

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-m $\mathcal{A}$ *W N* = *set-mset* (*dom-m N*) $\wedge$
set-mset
(*all-lits-of-mm*

$(\{\#mset\ (fst\ x).\ x \in\#\ ran\text{-}m\ N\#\} + (NE{+}NEk) + (UE{+}UEk) + NS + US + N0 + U0)) \subseteq$
set-mset $(\mathcal{L}_{all}\ \mathcal{A}) \wedge$
$(W', W) \in \langle Id \rangle$*map-fun-rel* $(D_0\ \mathcal{A}) \wedge$
isasat-input-bounded $\mathcal{A} \wedge$
distinct vdom $\wedge$
ivdom $=$ *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-no-WL-wl* :: ‹*nat multiset* ⇒ *bool* ⇒ (- × *nat twl-st-wl-init'*) *set*› **where**
‹*twl-st-heur-parsing-no-WL-wl* $\mathcal{A}$ *unbdd* =
{(*S*,
(*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*,*Q*)).
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* (*dom-m N*) $\subseteq$ *set 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 $\mathcal{A}$ $\varphi \wedge$
no-dup M $\wedge$
cach-refinement-empty $\mathcal{A}$ *cach* $\wedge$
set-mset (*all-lits-of-mm* $(\{\#mset\ (fst\ x).\ x \in\#\ ran\text{-}m\ N\#\} + (NE{+}NEk) + (UE{+}UEk) + NS + US + N0 + U0))$
$\subseteq$ *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 $\wedge$ *ivdom* = *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-no-WL-wl-no-watched* :: ‹*nat multiset* ⇒ *bool* ⇒ (*twl-st-wl-heur-init-full* × *nat twl-st-wl-init*) *set*› **where**
‹*twl-st-heur-parsing-no-WL-wl-no-watched* $\mathcal{A}$ *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* ⟶ ¬*failed*) ∧
((*unbdd* ∨ ¬*failed*) ⟶
(*valid-arena N′ N* (*set vdom*) ∧ *set-mset* (*dom-m N*) ⊆ *set vdom*) ∧ *ivdom* = *vdom*) ∧ (*M′*, *M*) ∈ *trail-pol* 𝒜 ∧
(*D′*, *D*) ∈ *option-lookup-clause-rel* 𝒜 ∧
j ≤ *length M* ∧
Q = *uminus* ‘# *lit-of* ‘# *mset* (*drop j* (*rev M*)) ∧
vm ∈ *bump-heur-init* 𝒜 *M* ∧
phase-saving 𝒜 φ ∧
no-dup M ∧
cach-refinement-empty 𝒜 *cach* ∧
set-mset (*all-lits-of-mm* ({#*mset* (*fst x*). *x* ∈# *ran-m N*#} + (*NE*+*NEk*) + (*UE*+*UEk*) + *NS* + *US* + *N0* + *U0*))
⊆ *set-mset* (*L*$_{all}$ 𝒜) ∧
(*W′*, *empty-watched* 𝒜) ∈ ⟨*Id*⟩*map-fun-rel* (*D*$_0$ 𝒜) ∧
isasat-input-bounded 𝒜 ∧
distinct vdom ∧
clss-size-corr N NE UE NEk UEk NS US N0 U0 lcount ∧
(*mark*, {#}) ∈ *lookup-clause-rel* 𝒜
}›

definition *twl-st-heur-post-parsing-wl* :: ‹*bool* ⇒ (*twl-st-wl-heur-init-full* × *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* ⟶ ¬*failed*) ∧
((*unbdd* ∨ ¬*failed*) ⟶
((*M′*, *M*) ∈ *trail-pol* (*all-atms N* ((*NE*+*NEk*) + (*UE*+*UEk*) + *NS* + *US* + *N0* + *U0*)) ∧
set-mset (*dom-m N*) ⊆ *set vdom* ∧
valid-arena N′ N (*set vdom*) ∧ *ivdom*=*vdom*)) ∧
(*D′*, *D*) ∈ *option-lookup-clause-rel* (*all-atms N* ((*NE*+*NEk*) + (*UE*+*UEk*) + *NS* + *US* + *N0* + *U0*)) ∧
j ≤ *length M* ∧
Q = *uminus* ‘# *lit-of* ‘# *mset* (*drop j* (*rev M*)) ∧
vm ∈ *bump-heur-init* (*all-atms N* ((*NE*+*NEk*) + (*UE*+*UEk*) + *NS* + *US* + *N0* + *U0*)) *M* ∧
phase-saving (*all-atms N* ((*NE*+*NEk*) + (*UE*+*UEk*) + *NS* + *US* + *N0* + *U0*)) φ ∧
no-dup M ∧
cach-refinement-empty (*all-atms N* ((*NE*+*NEk*) + (*UE*+*UEk*) + *NS* + *US* + *N0* + *U0*)) *cach* ∧
vdom-m (*all-atms N* ((*NE*+*NEk*) + (*UE*+*UEk*) + *NS* + *US* + *N0* + *U0*)) *W N* ⊆ *set vdom* ∧
set-mset (*all-lits-of-mm* ({#*mset* (*fst x*). *x* ∈# *ran-m N*#} + (*NE*+*NEk*) + (*UE*+*UEk*) + *NS* + *US* + *N0* + *U0*))
⊆ *set-mset* (*L*$_{all}$ (*all-atms N* ((*NE*+*NEk*) + (*UE*+*UEk*) + *NS* + *US* + *N0* + *U0*))) ∧
(*W′*, *W*) ∈ ⟨*Id*⟩*map-fun-rel* (*D*$_0$ (*all-atms N* ((*NE*+*NEk*) + (*UE*+*UEk*) + *NS* + *US* + *N0* + *U0*))) ∧
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›*
⟨*proof*⟩

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›*
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ∧ *get-conflict-init-wl S* = *None*]$_f$
nat-lit-lit-rel ×$_r$ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd* → ⟨*twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*⟩
nres-rel›
⟨*proof*⟩

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* ≠ []›
⟨*proof*⟩

context
fixes *unbdd* :: *bool* **and** *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*›
⟨*proof*⟩

lemma *add-init-pre2*: ‹*2* ≤ *length C'*›
⟨*proof*⟩ **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*›
⟨*proof*⟩

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 -›)
⟨*proof*⟩

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* 𝒜 *unbdd*›
⟨*proof*⟩
end

lemma *learned-clss-l-fmupd-if-in*:
assumes ‹*C′* ∈# *dom-m 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*⟩

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*
›
⟨*proof*⟩

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-*$\mathcal{L}_{in}$ 𝒜 (*mset C*) ∧ *distinct C*]$_f$
Id ×$_r$ *twl-st-heur-parsing-no-WL* 𝒜 *unbdd* → ⟨*twl-st-heur-parsing-no-WL* 𝒜 *unbdd*⟩ *nres-rel*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*) ∈
[λ*D*. *D* = *None*]$_f$ *option-lookup-clause-rel* $\mathcal{A}$ → ⟨*option-lookup-clause-rel* $\mathcal{A}$⟩*nres-rel*›
⟨*proof*⟩

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* = (λ*S*. *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*) ∈
[λ*S*. *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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)) ∈

$[\lambda(C, \text{-}).\ \textit{literals-are-in-}\mathcal{L}_{in}\ \mathcal{A}\ (\textit{mset}\ C)\ ]_f$
$\langle \textit{Id} \rangle \textit{list-rel} \times_r \textit{twl-st-heur-parsing-no-WL}\ \mathcal{A}\ \textit{unbdd} \to$
$\langle \textit{twl-st-heur-parsing-no-WL}\ \mathcal{A}\ \textit{unbdd} \rangle\ \textit{nres-rel}\rangle$
$\langle \textit{proof} \rangle$

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* C_0 *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)}
  })›
  ⟨proof⟩

named-theorems twl-st-heur-parsing-no-WL
lemma [twl-st-heur-parsing-no-WL]:
  assumes ‹(S, T) ∈ twl-st-heur-parsing-no-WL 𝒜 unbdd›
  shows ‹(get-trail-init-wl-heur S, get-trail-init-wl T) ∈ trail-pol 𝒜›
  ⟨proof⟩


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 ⟷ get-conflict-init-wl S = None›
  ⟨proof⟩

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) ∈
    twl-st-heur-parsing-no-WL 𝒜 unbdd →f ⟨Id⟩nres-rel›
  ⟨proof⟩
```

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* ⟷ *tautology* (*mset C*)) ∧
(¬*tauto* ⟶ *remdups-mset* (*mset C*) = *mset C′*) ∧
(*S′*, *T*) ∈ *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* ⟷ *tautology* (*mset C*)) ∧
(¬*tauto* ⟶ *remdups-mset* (*mset C*) = *mset C′* ∧ *distinct C′*) ∧
(*S′*, *T*) ∈ *twl-st-heur-parsing-no-WL* $\mathcal{A}$ *unbdd*}›
⟨*proof*⟩

lemma *pre-simplify-clause-lookup-st-remdups-clause*:
fixes *D* :: ‹*nat clause-l*›
assumes ‹(*C*, *C′*) ∈ *Id*› ‹(*S*, *T*) ∈ *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*⟩

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*})›
⟨*proof*⟩

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*⟩

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*›
  ⟨*proof*⟩

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*⟩

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*
  }›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

definition *add-to-atms-ext* **where**
  ‹*add-to-atms-ext* = (λ*i* (*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*⟩

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*⟩

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*⟩

lemma *init-valid-rep-in-set-iff*:
‹*init-valid-rep x1b x2* ⟹ *x* ∈ *x2* ⟷ (*x* < *length x1b* ∧ (*x1b*!*x*) *mod 2* = *1*)›
⟨*proof*⟩

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*⟩

definition *extract-atms-cls* :: ‹*'a clause-l* ⇒ *'a set* ⇒ *'a set*› **where**
‹*extract-atms-cls C* $\mathcal{A}_{in}$ = *fold* (λ*L* $\mathcal{A}_{in}$. *insert* (*atm-of L*) $\mathcal{A}_{in}$) *C* $\mathcal{A}_{in}$›

definition *extract-atms-cls-i* :: ‹*nat clause-l* ⇒ *nat set* ⇒ *nat set nres*› **where**
‹*extract-atms-cls-i C* $\mathcal{A}_{in}$ = *nfoldli C* (λ-. *True*)
(λ*L* $\mathcal{A}_{in}$. *do* {
ASSERT(*atm-of L* ≤ *unat32-max div 2*);
RETURN(*insert* (*atm-of L*) $\mathcal{A}_{in}$)})
$\mathcal{A}_{in}$›

lemma *fild-insert-insert-swap*:
‹*fold* (λ*L*. *insert* (*f L*)) *C* (*insert a* $\mathcal{A}_{in}$) = *insert a* (*fold* (λ*L*. *insert* (*f L*)) *C* $\mathcal{A}_{in}$)›
⟨*proof*⟩

lemma *extract-atms-cls-alt-def*: ‹*extract-atms-cls C* $\mathcal{A}_{in}$ = $\mathcal{A}_{in}$ ∪ *atm-of* ‘ *set C*›
⟨*proof*⟩

lemma *extract-atms-cls-i-extract-atms-cls*:
‹(*uncurry extract-atms-cls-i*, *uncurry* (*RETURN oo extract-atms-cls*))
∈ [λ(*C*, $\mathcal{A}_{in}$). ∀ *L*∈*set C*. *nat-of-lit L* ≤ *unat32-max*]$_f$
⟨*Id*⟩*list-rel* ×$_f$ *Id* → ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

lemma *extract-atms-clss-alt-def*:
‹*extract-atms-clss N* $\mathcal{A}_{in}$ = $\mathcal{A}_{in}$ ∪ ((⋃ *C*∈*set N*. *atm-of* ‘ *set C*))›
⟨*proof*⟩

lemma *finite-extract-atms-clss*[*simp*]: ‹*finite* (*extract-atms-clss CS'* {})› **for** *CS'*
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *extract-atms-cls-Nil*[*simp*]:
‹*extract-atms-cls* [] $\mathcal{A}_{in}$ = $\mathcal{A}_{in}$›
⟨*proof*⟩

lemma *extract-atms-clss-Cons*[*simp*]:
‹*extract-atms-clss* (*C* # *Cs*) *N* = *extract-atms-clss Cs* (*extract-atms-cls C N*)›
⟨*proof*⟩

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* {#} = {#}›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩
lemma *clss-size-corr-empty* [*simp*]: ‹*clss-size-corr fmempty* {#} {#} {#} {#} {#} {#} {#} {#} (*0*, *0* ,*0*, *0*, *0*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*⟩

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*⟩

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*))›
⟨*proof*⟩

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*⟩

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* *A* *au* ⟹ *A* ≠ {#} ⟹ ∃ *y*. *an* = *Some y*›
‹((*ak*, *al*, *am*, *an*, *bc*)) ∈ *isa-vmtf-init* *A* *au* ⟹ *A* ≠ {#} ⟹ ∃ *y*. *am* = *Some y*›
⟨*proof*⟩

lemma *isa-vmtf-init-isa-vmtf*: ‹*A* ≠ {#} ⟹ ((*ak*, *al*, *Some am*, *Some an*, *bc*))
∈ *isa-vmtf-init* *A* *au* ⟹ ((*ak*, *al*, *am*, *an*, *bc*))
∈ *vmtf* *A* *au*›
⟨*proof*⟩

lemma *bump-heur-init-isa-vmtf*: ‹*A* ≠ {#} ⟹ *x* ∈ *bump-heur-init* *A* *M* ⟹ *finalize-bump-init x* ≤ ⇓*Id* (*SPEC* (λ*x*. *x* ∈ *bump-heur* *A* *M*))›
⟨*proof*⟩

lemma *heuristic-rel-initI*:
‹*phase-saving* *A* *φ* ⟹ *length φ′* = *length φ* ⟹ *length φ″* = *length φ* ⟹ *phase-saving* *A* *g* ⟹
heuristic-rel *A* (*Restart-Heuristics* ((*fema*, *sema*, *ccount*, *0*, (*φ*,*a*, *φ′*,*b*,*φ″*,*c*,*d*), *e*, *f*, *g*, *h*)))›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*›
⟨*proof*⟩

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($\mathcal{A}_{in}$, *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 = (*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* = $\mathcal{A}_{in}$ ∧
distinct N ∧ (∀ *L*∈*set N*. *L* < *n*) ∧ *m* = *2* ∗ *n* ∧ *isasat-input-bounded* $\mathcal{A}_{in}$]$_f$
⟨*Id*⟩*list-rel* ×$_f$ *nat-rel* ×$_f$ *nat-rel* →
⟨*trail-pol* $\mathcal{A}_{in}$⟩ *nres-rel*›

⟨*proof*⟩

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*}›
⟨*proof*⟩

lemma *to-tuple-eq-iff*[*iff*]: ‹*to-tuple S* = *to-tuple T* ⟷ *S* = *T*›
⟨*proof*⟩

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*⟩

lemma *init-state-wl-D′*:
‹(*init-state-wl-D′*, *init-state-wl-heur*) ∈
[λ$\mathcal{A}_{in}$. *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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *all-blits-are-in-problem-init-blits-in*: ‹*all-blits-are-in-problem-init S* ⟹ *blits-in-*$\mathcal{L}_{in}$ *S*›
⟨*proof*⟩

lemma *correct-watching-init-blits-in-*$\mathcal{L}_{in}$:
assumes ‹*correct-watching-init S*›
shows ‹*blits-in-*$\mathcal{L}_{in}$ *S*›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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'*)›
⟨*proof*⟩

definition *isasat-init-fast-slow* :: ‹*twl-st-wl-heur-init* ⇒ *twl-st-wl-heur-init nres*› **where**
‹*isasat-init-fast-slow* =
(λ*S*::*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*›
⟨*proof*⟩

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* ≡ (λ*S*. *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
⟨*proof*⟩
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)]
⟨*proof*⟩

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')
⟨*proof*⟩

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*
⟨*proof*⟩

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
}
⟨*proof*⟩

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

}
⟨*proof*⟩

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*)
}
⟨*proof*⟩
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*)
}
⟨*proof*⟩

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'*)
⟨*proof*⟩
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*›
⟨*proof*⟩

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* ∧∗ Γ*1*)

(*f a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15*)
(Γ*2 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′*.
Γ*2 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* ** Γ*1′*›
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* ** Γ*1′*)
R (*case-tuple15 CP p*) (*case-tuple15*$(λ*2a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15*. *f′ a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15*)$*p′*)› (**is** ‹*?G* Γ›)
⟨*proof*⟩
apply1 (*rule hn-refine-cons-pre*[*OF FR*])
apply1 (*cases p*; *cases p′*; *simp add*: *tuple15-assn-conv*[*THEN tuple15-assn-ctxt*])
⟨*proof*⟩
applyS (*simp add*: *hn-ctxt-def*)
applyS *simp* ⟨*proof*⟩

lemma *case-tuple15-arity*[*sepref-monadify-arity*]:
‹*case-tuple15* ≡ λ*2fp p*. *SP case-tuple15*$(λ*2a b*. *fp*$*a*$*b*)$*p*›
⟨*proof*⟩

lemma *case-tuple15-comb*[*sepref-monadify-comb*]:
‹⋀*fp p*. *case-tuple15*$*fp*$*p* ≡ *Refine-Basic.bind*$(*EVAL*$*p*)$(λ*2p*. (*SP case-tuple15*$*fp*$*p*))›
⟨*proof*⟩

lemma *case-tuple15-plain-comb*[*sepref-monadify-comb*]:
EVAL$(*case-tuple15*$(λ*2a1 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*$*p*)$(λ*2p*. *case-tuple15*$(λ*2a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15*. *EVAL*$(*fp a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15*))$*p*)
⟨*proof*⟩

lemma *ho-tuple15-move*[*sepref-preproc*]: ‹*case-tuple15* (λ*a1 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*) =
(λ*p x*. *case-tuple15* (*f x*) *p*)›
⟨*proof*⟩

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*))
⟨*proof*⟩

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}
$\rightarrow_a$ *tuple15-int-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *RETURN-case-tuple15-inverse*: ‹*RETURN*
(*let* - = *M*
in ff) =
(*do* {- ← *mop-free M*;
RETURN (*ff*)})›
⟨*proof*⟩

sepref-definition *update-a-code*
is ‹*uncurry* (*RETURN oo update-a*)›
:: ‹*a-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *update-b-code*
is ‹*uncurry* (*RETURN oo update-b*)›
:: ‹*b-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *update-c-code*
is ‹*uncurry* (*RETURN oo update-c*)›
:: ‹*c-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *update-d-code*
is ‹*uncurry* (*RETURN oo update-d*)›
:: ‹*d-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *update-e-code*
is ‹*uncurry* (*RETURN oo update-e*)›
:: ‹*e-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *update-f-code*

is ‹*uncurry* (*RETURN oo update-f*)›
:: ‹*f-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *update-g-code*
is ‹*uncurry* (*RETURN oo update-g*)›
:: ‹*g-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *update-h-code*
is ‹*uncurry* (*RETURN oo update-h*)›
:: ‹*h-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *update-i-code*
is ‹*uncurry* (*RETURN oo update-i*)›
:: ‹*i-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *update-j-code*
is ‹*uncurry* (*RETURN oo update-j*)›
:: ‹*j-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *update-k-code*
is ‹*uncurry* (*RETURN oo update-k*)›
:: ‹*k-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *update-l-code*
is ‹*uncurry* (*RETURN oo update-l*)›
:: ‹*l-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *update-m-code*
is ‹*uncurry* (*RETURN oo update-m*)›
:: ‹*m-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *update-n-code*
is ‹*uncurry* (*RETURN oo update-n*)›
:: ‹*n-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *update-o-code*
is ‹*uncurry* (*RETURN oo update-o*)›
:: ‹*o-assn*^{d} $*_a$ *tuple15-int-assn*^{d} $\rightarrow_a$ *tuple15-int-assn*›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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} →$_a$ *a-assn* ×$_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *remove-b-code*
is ‹*RETURN o remove-b*›
:: ‹ *tuple15-int-assn*^{d} →$_a$ *b-assn* ×$_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *remove-c-code*
is ‹*RETURN o remove-c*›
:: ‹ *tuple15-int-assn*^{d} →$_a$ *c-assn* ×$_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *remove-d-code*
is ‹*RETURN o remove-d*›
:: ‹ *tuple15-int-assn*^{d} →$_a$ *d-assn* ×$_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *remove-e-code*
is ‹*RETURN o remove-e*›
:: ‹ *tuple15-int-assn*^{d} →$_a$ *e-assn* ×$_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *remove-f-code*
is ‹*RETURN o remove-f*›
:: ‹ *tuple15-int-assn*^{d} →$_a$ *f-assn* ×$_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *remove-g-code*
is ‹*RETURN o remove-g*›
:: ‹ *tuple15-int-assn*^{d} →$_a$ *g-assn* ×$_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *remove-h-code*
is ‹*RETURN o remove-h*›
:: ‹ *tuple15-int-assn*^{d} →$_a$ *h-assn* ×$_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *remove-i-code*
is ‹*RETURN o remove-i*›
:: ‹ *tuple15-int-assn*^{d} →$_a$ *i-assn* ×$_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *remove-j-code*
is ‹*RETURN o remove-j*›
:: ‹ *tuple15-int-assn*^{d} →$_a$ *j-assn* ×$_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *remove-k-code*
is ‹*RETURN o remove-k*›
:: ‹ *tuple15-int-assn*^{d} →$_a$ *k-assn* ×$_a$ *tuple15-int-assn*›

⟨*proof*⟩

sepref-definition *remove-l-code*
is ‹*RETURN o remove-l*›
:: ‹ *tuple15-int-assn*^{d} →$_a$ *l-assn* ×$_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *remove-m-code*
is ‹*RETURN o remove-m*›
:: ‹ *tuple15-int-assn*^{d} →$_a$ *m-assn* ×$_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *remove-n-code*
is ‹*RETURN o remove-n*›
:: ‹ *tuple15-int-assn*^{d} →$_a$ *n-assn* ×$_a$ *tuple15-int-assn*›
⟨*proof*⟩

sepref-definition *remove-o-code*
is ‹*RETURN o remove-o*›
:: ‹ *tuple15-int-assn*^{d} →$_a$ *o-assn* ×$_a$ *tuple15-int-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*
})›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *bottom-init-vmtf-code*
is ‹*uncurry0* (*RETURN bottom-init-vmtf*)›
:: ‹*unit-assn*^{k} →$_a$ *vmtf-init-assn*›
⟨*proof*⟩

schematic-goal *free-vmtf-remove-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE vmtf-init-assn ?a*›
⟨*proof*⟩

sepref-def *free-vmtf-remove*
  is ‹*mop-free*›
  :: ‹*vmtf-init-assn*^{d} →$_a$ *unit-assn*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

lemma *free-vmtf-init-assn-assn2*: ‹*MK-FREE vmtf-init-assn free-vmtf-remove*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *free-vdom*
is ‹*mop-free*›
:: ‹*vdom-fast-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

schematic-goal *free-vdom*[*sepref-frame-free-rules*]: ‹*MK-FREE vdom-fast-assn ?a*›
⟨*proof*⟩

lemma *free-vdom2*: ‹*MK-FREE vdom-fast-assn free-vdom*›
⟨*proof*⟩

sepref-def *free-phase-saver*
is ‹*mop-free*›
:: ‹*phase-saver-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

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*›

⟨*proof*⟩

schematic-goal *free-phase-saver*[*sepref-frame-free-rules*]: ‹*MK-FREE phase-saver-assn ?a*›
⟨*proof*⟩

lemma *free-phase-saver2*: ‹*MK-FREE phase-saver-assn free-phase-saver*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *free-bottom-init-bump-code*
is ‹*mop-free*›
:: ‹*heuristic-bump-init-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

lemma *free-vmtf-remove2*: ‹*MK-FREE heuristic-bump-init-assn free-bottom-init-bump-code*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *bottom-init-vmtf2-code*
is ‹*uncurry0* (*RETURN bottom-init-bump*)›
:: ‹*unit-assn*^{k} →$_a$ *heuristic-bump-init-assn*›
⟨*proof*⟩

definition *bottom-bool* **where**
‹*bottom-bool* = *False*›

sepref-def *bottom-bool-code*
is ‹*uncurry0* (*RETURN bottom-bool*)›
:: ‹*unit-assn*^{k} →$_a$ *bool1-assn*›
⟨*proof*⟩

sepref-def *free-bool*
is ‹*mop-free*›
:: ‹*bool1-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

schematic-goal *free-bool*[*sepref-frame-free-rules*]: ‹*MK-FREE bool1-assn ?a*›
⟨*proof*⟩

lemma *free-bool2*: ‹*MK-FREE bool1-assn free-bool*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *free-marked-struct-code*
is ‹*mop-free*›
:: ‹*marked-struct-assn*^{d} →$_a$ *unit-assn*›
⟨*proof*⟩

schematic-goal *free-marked-struct*[*sepref-frame-free-rules*]: ‹*MK-FREE marked-struct-assn ?a*›
⟨*proof*⟩

lemma *free-marked-struct2*: ‹*MK-FREE marked-struct-assn free-marked-struct-code*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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** $x\text{-}assn :: \langle 'x \Rightarrow 'q \Rightarrow assn\rangle$
  assumes *not-deleted-code-refine*:
    $\langle(uncurry15\ (\lambda a\ b\ c\ d\ e\ f\ g\ h\ i\ j\ k\ l\ m\ n\ ko\ C.\ kf\ C\ a\ b\ c\ d\ e\ f\ g\ h\ i\ j\ k\ l\ m\ n\ ko),$
    $uncurry15\ (\lambda a\ b\ c\ d\ e\ f\ g\ h\ i\ j\ k\ l\ m\ n\ ko\ C.\ f'\ C\ a\ b\ c\ d\ e\ f\ g\ h\ i\ j\ k\ l\ m\ n\ ko))$
    $\in [uncurry15\ (\lambda a\ b\ c\ d\ e\ f\ g\ h\ i\ j\ k\ l\ m\ n\ ko\ C.\ P\ C\ a\ b\ c\ d\ e\ f\ g\ h\ i\ j\ k\ l\ m\ n\ ko)]_a$
    $a\text{-}assn^k *_a b\text{-}assn^k *_a c\text{-}assn^k *_a d\text{-}assn^k *_a$
  $e\text{-}assn^k *_a f\text{-}assn^k *_a g\text{-}assn^k *_a h\text{-}assn^k *_a i\text{-}assn^k *_a$
  $j\text{-}assn^k *_a k\text{-}assn^k *_a l\text{-}assn^k *_a m\text{-}assn^k *_a$
  $n\text{-}assn^k *_a o\text{-}assn^k *_a (pure\ R)^k \to x\text{-}assn\rangle$
begin

context
begin
lemma *not-deleted-code-refine-tmp*:
  $\langle\bigwedge C\ C'.\ (C,\ C') \in R \Longrightarrow (uncurry14\ (kf\ C),\ uncurry14\ (f'\ C')) \in [uncurry14\ (P\ C')]_a$
    $a\text{-}assn^k *_a b\text{-}assn^k *_a c\text{-}assn^k *_a d\text{-}assn^k *_a$
  $e\text{-}assn^k *_a f\text{-}assn^k *_a g\text{-}assn^k *_a h\text{-}assn^k *_a i\text{-}assn^k *_a$
  $j\text{-}assn^k *_a k\text{-}assn^k *_a l\text{-}assn^k *_a m\text{-}assn^k *_a$
  $n\text{-}assn^k *_a o\text{-}assn^k \to x\text{-}assn\rangle$
  $\langle proof\rangle$
end
lemma *read-all-refine*:
  $\langle(uncurry\ (\lambda N\ C.\ read\text{-}all\text{-}st\text{-}code\ (kf\ C)\ N),$
    $uncurry\ (\lambda N\ C'.\ read\text{-}all\text{-}st\ (f'\ C')\ N))$
  $\in [uncurry\ (\lambda S\ C.\ P\ C$
  $(Tuple15\text{-}a\ S)$
  $(Tuple15\text{-}b\ S)$
  $(Tuple15\text{-}c\ S)$
  $(Tuple15\text{-}d\ S)$
  $(Tuple15\text{-}e\ S)$
  $(Tuple15\text{-}f\ S)$
  $(Tuple15\text{-}g\ S)$
  $(Tuple15\text{-}h\ S)$
  $(Tuple15\text{-}i\ S)$
  $(Tuple15\text{-}j\ S)$
  $(Tuple15\text{-}k\ S)$
  $(Tuple15\text{-}l\ S)$
  $(Tuple15\text{-}m\ S)$
  $(Tuple15\text{-}n\ S)$
  $(Tuple15\text{-}o\ S))]_a\ tuple15\text{-}int\text{-}assn^k *_a (pure\ R)^k \to x\text{-}assn\rangle$
  $\langle proof\rangle$

lemma *read-all-mop-refine*:
  $\langle(uncurry\ (\lambda N\ C.\ read\text{-}all\text{-}st\text{-}code\ (kf\ C)\ N),$
    $uncurry\ mop)$
  $\in tuple15\text{-}int\text{-}assn^k *_a (pure\ R)^k \to_a x\text{-}assn\rangle$
  $\langle proof\rangle$
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** $x\text{-}assn :: ‹'r \Rightarrow 'q \Rightarrow assn›$ **and** P
  assumes *not-deleted-code-refine*: $‹(uncurry\ (\lambda S\ C.\ kf\ C\ S),\ uncurry\ (\lambda S\ C.\ f'\ C\ S)) \in [uncurry\ (\lambda S\ C.\ P\ C\ S)]_a\ a\text{-}assn^k *_a (pure\ R)^k \to x\text{-}assn›$
begin
private lemma *not-deleted-code-refine'*:
  $‹(uncurry15\ (\lambda M\ \text{- - - - - - - - - - - - - - - }\ C.\ kf\ C\ M),\ uncurry15\ (\lambda M\ \text{- - - - - - - - - - - - - - - }\ C'.\ f'\ C'\ M)) \in [uncurry15\ (\lambda M\ \text{- - - - - - - - - - - - - - - }\ C.\ P\ C\ M)]_a$
   $a\text{-}assn^k *_a b\text{-}assn^k *_a c\text{-}assn^k *_a d\text{-}assn^k *_a$
  $e\text{-}assn^k *_a f\text{-}assn^k *_a g\text{-}assn^k *_a h\text{-}assn^k *_a i\text{-}assn^k *_a$
  $j\text{-}assn^k *_a k\text{-}assn^k *_a l\text{-}assn^k *_a m\text{-}assn^k *_a$
  $n\text{-}assn^k *_a o\text{-}assn^k *_a (pure\ R)^k \to x\text{-}assn›$
  ⟨*proof*⟩

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**
  $‹read\text{-}conflict\text{-}wl\text{-}heur\text{-}code\ kf \equiv IsaSAT\text{-}Init.read\text{-}all\text{-}st\text{-}code\ (\lambda M\ \text{- - - - - - - - - - - - - - - }.\ kf\ M)›$
abbreviation *read-conflict-wl-heur* :: ‹-› **where**
  $‹read\text{-}conflict\text{-}wl\text{-}heur\ kf \equiv IsaSAT\text{-}Init.read\text{-}all\text{-}st\ (\lambda M\ \text{- - - - - - - - - - - - - - - }.\ kf\ M)›$

context
  fixes R **and** kf **and** f' **and** $x\text{-}assn :: ‹'r \Rightarrow 'q \Rightarrow assn›$ **and** P
  assumes *not-deleted-code-refine*: $‹(uncurry\ (\lambda S\ C.\ kf\ C\ S),\ uncurry\ (\lambda S\ C.\ f'\ C\ S)) \in [uncurry\ (\lambda S\ C.\ P\ C\ S)]_a\ c\text{-}assn^k *_a (pure\ R)^k \to x\text{-}assn›$
begin
private lemma *not-deleted-code-refine-conflict*:
  $‹(uncurry15\ (\lambda\text{- - }M\ \text{- - - - - - - - - - - - - }\ C.\ kf\ C\ M),\ uncurry15\ (\lambda\text{- - }M\ \text{- - - - - - - - - - - - - }\ C'.\ f'\ C'\ M)) \in [uncurry15\ (\lambda\text{- - }M\ \text{- - - - - - - - - - - - - }\ C.\ P\ C\ M)]_a$
   $a\text{-}assn^k *_a b\text{-}assn^k *_a c\text{-}assn^k *_a d\text{-}assn^k *_a$
  $e\text{-}assn^k *_a f\text{-}assn^k *_a g\text{-}assn^k *_a h\text{-}assn^k *_a i\text{-}assn^k *_a$
  $j\text{-}assn^k *_a k\text{-}assn^k *_a l\text{-}assn^k *_a m\text{-}assn^k *_a$
  $n\text{-}assn^k *_a o\text{-}assn^k *_a (pure\ R)^k \to x\text{-}assn›$
  ⟨*proof*⟩

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** $x\text{-}assn :: ‹'r \Rightarrow 'q \Rightarrow assn›$ **and** P
  assumes *not-deleted-code-refine*: $‹((\lambda S.\ kf\ S),\ (\lambda S.\ f'\ S)) \in [(\lambda S.\ P\ S)]_a\ c\text{-}assn^k \to x\text{-}assn›$
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*›
⟨*proof*⟩

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-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*)]$_a$ *n-assn*^{k} *$*_a$ (*pure R*)^k → *x-assn*›
begin
private lemma *not-deleted-code-refine-n*:
‹(*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*›
⟨*proof*⟩

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*: $\langle(uncurry\ (\lambda S\ C.\ kf\ C\ S),\ uncurry\ (\lambda S\ C.\ f'\ C\ S)) \in [uncurry\ (\lambda S\ C.\ P\ C\ S)]_a\ m\text{-}assn^k *_a\ (pure\ R)^k \to x\text{-}assn\rangle$
begin
private lemma *not-deleted-code-refine-m*:
  $\langle(uncurry15\ (\lambda\text{- - - - - - - - - - - - - }M\text{ - - }C.\ kf\ C\ M),\ uncurry15\ (\lambda\text{- - - - - - - - - - - - - }M\text{ - - }C'.\ f'\ C'\ M)) \in [uncurry15\ (\lambda\text{- - - - - - - - - - - - - }M\text{ - - }C.\ P\ C\ M)]_a$
   $a\text{-}assn^k *_a\ b\text{-}assn^k *_a\ c\text{-}assn^k *_a\ d\text{-}assn^k *_a$
  $e\text{-}assn^k *_a\ f\text{-}assn^k *_a\ g\text{-}assn^k *_a\ h\text{-}assn^k *_a\ i\text{-}assn^k *_a$
  $j\text{-}assn^k *_a\ k\text{-}assn^k *_a\ l\text{-}assn^k *_a\ m\text{-}assn^k *_a$
  $n\text{-}assn^k *_a\ o\text{-}assn^k *_a\ (pure\ R)^k \to x\text{-}assn\rangle$
  ⟨*proof*⟩

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** $x\text{-}assn :: \langle 'r \Rightarrow 'q \Rightarrow assn\rangle$ **and** P
  assumes *not-deleted-code-refine*: $\langle((\lambda S.\ kf\ S),\ (\lambda S.\ f'\ S)) \in [(\lambda S.\ P\ S)]_a\ m\text{-}assn^k \to x\text{-}assn\rangle$
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* :: ⟨*twl-st-wl-heur-init* ⇒ -⟩ **where**
  ⟨*polarity-st-heur-init S L* = *polarity-pol* (*Tuple15-a S*) *L*⟩

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›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *distinct-atms-empty-code*
is ‹*distinct-atms-int-empty*›
:: ‹*sint64-nat-assn*^{k} →$_a$ *distinct-atoms-assn*›
⟨*proof*⟩

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} →$_a$ *hp-assn*›
⟨*proof*⟩

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*⟩

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'* $\mathcal{A}$ *n* = *do* {
ac ← *init-ACIDS0'* $\mathcal{A}$ *n*;
RETURN (*ac*, *0*)
}›

lemma *init-ACIDS'-alt*: ‹*init-ACIDS* (*mset N*) *n* = *init-ACIDS'* *N n*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *acids-heur-import-variable-code*
is ‹*uncurry acids-heur-import-variable*›
:: ‹*atom-assn*^{k} *∗$_a$ acids-assn2*^{d} →$_a$ *acids-assn2*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*⟩

sepref-def *initialise-ACIDS-rev-code*
is ‹*uncurry initialise-ACIDS-rev*›
:: ‹$[\lambda(N,\ n).\ True]_a$ $(arl64\text{-}assn\ atom\text{-}assn)^k *_a sint64\text{-}nat\text{-}assn^k \to acids\text{-}assn2$›
⟨*proof*⟩

sepref-def *initialise-VMTF-code*
is ‹*uncurry initialise-VMTF*›
:: ‹$[\lambda(N,\ n).\ True]_a$ $(arl64\text{-}assn\ atom\text{-}assn)^k *_a sint64\text{-}nat\text{-}assn^k \to vmtf\text{-}init\text{-}assn$›
⟨*proof*⟩

sepref-register *initialize-Bump-Init*
sepref-def *initialize-Bump-Init-code*
is ‹*uncurry initialize-Bump-Init*›
:: ‹$[\lambda(N,\ n).\ True]_a$ $(arl64\text{-}assn\ atom\text{-}assn)^k *_a sint64\text{-}nat\text{-}assn^k \to heuristic\text{-}bump\text{-}init\text{-}assn$›
⟨*proof*⟩

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)
}›
⟨*proof*⟩

sepref-def *propagate-unit-cls-code*
is ‹*uncurry* (*propagate-unit-cls-heur-b*)›
:: ‹$unat\text{-}lit\text{-}assn^k *_a isasat\text{-}init\text{-}assn^d \to_a isasat\text{-}init\text{-}assn$›
⟨*proof*⟩

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′ = (λunbdd S. RETURN S)*›

lemma *already-propagated-unit-cls-heur′-alt*:
‹*already-propagated-unit-cls-heur unbd L = already-propagated-unit-cls-heur′ unbd*›
⟨*proof*⟩

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} $\rightarrow_a$ *isasat-init-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)))
}›
⟨*proof*⟩

sepref-def *conflict-propagated-unit-cls-code*
is ‹*uncurry* (*conflict-propagated-unit-cls-heur-b*)›
:: ‹*unat-lit-assn*^{k} $*_a$ *isasat-init-assn*^{d} $\rightarrow_a$ *isasat-init-assn*›
⟨*proof*⟩

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'*›
⟨*proof*⟩

lemma *add-init-cls-code-bI2*:
assumes
‹*length at'* ≤ *Suc* (*Suc unat32-max*)›
shows ‹5 ≤ *snat64-max* − *length at'*›
⟨*proof*⟩

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›
⟨proof⟩

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)›
⟨proof⟩

lemma *op-list-list-idx-alt-def*: ‹op-list-list-idx xs i j = xs ! i ! j›
⟨proof⟩

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›
⟨proof⟩

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))}›
⟨proof⟩

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›
⟨proof⟩

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))

}›
⟨proof⟩

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*›
⟨proof⟩

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*›
⟨proof⟩

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*›
⟨proof⟩

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*)))
})›
⟨proof⟩

sepref-def *set-empty-clause-as-conflict-code*
is ‹*set-empty-clause-as-conflict-heur*›
:: ‹*isasat-init-assn*^{d} $\rightarrow_a$ *isasat-init-assn*›
⟨proof⟩

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′*›
⟨proof⟩

sepref-def *add-clause-to-others-code*
is ‹*add-clause-to-others-heur′*›
:: ‹*isasat-init-assn*^{d} $\rightarrow_a$ *isasat-init-assn*›
⟨proof⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*))
})›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *add-tautology-to-clauses′-impl*
is *add-tautology-to-clauses′*
:: ‹*isasat-init-assn*^{d} →$_a$ *isasat-init-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩
hide-const (**open**) *NEMonad.ASSERT NEMonad.RETURN*
lemma *while-nfoldli*:
do {
(-,σ) ← *WHILE*$_T$ (*FOREACH-cond c*) (λ*x*. *do* {*ASSERT* (*FOREACH-cond c x*); *FOREACH-body f x*}) (*l*,σ);
RETURN σ
} ≤ *nfoldli l c f* σ
⟨*proof*⟩

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} → *nat-lit-list-hm-assn*›
  ⟨*proof*⟩

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} →$_a$ *nat-lit-list-hm-assn*›
  ⟨*proof*⟩

lemma *extract-atms-clss-hnr*[*sepref-fr-rules*]:
  ‹(*uncurry extract-atms-clss-imp*, *uncurry* (*RETURN* ∘∘ *extract-atms-clss*))
    ∈ [λ(*a*, *b*). ∀ *C*∈*set a*. ∀ *L*∈*set C*. *nat-of-lit L* ≤ *unat32-max*]$_a$
      (*clauses-ll-assn*)^k *$_a$ *nat-lit-list-hm-assn*^{d} → *nat-lit-list-hm-assn*›
  ⟨*proof*⟩

sepref-def *extract-atms-clss-imp-empty-assn*
  is ‹*uncurry0 extract-atms-clss-imp-empty-rel*›
  :: ‹*unit-assn*^{k} →$_a$ *isasat-atms-ext-rel-assn*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

lemma *extract-atms-clss-imp-empty-rel-alt-def*:
  ‹*extract-atms-clss-imp-empty-rel* = (*RETURN* (*op-larray-custom-replicate 1024 0*, *0*, []))›
  ⟨*proof*⟩

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*)))
  })›
  ⟨*proof*⟩
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*›
  ⟨*proof*⟩

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} $\to_a$
    isasat-init-assn›
  ⟨*proof*⟩

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*})›
  ⟨*proof*⟩

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 $\times_a$ *uint32-nat-assn*›
  ⟨*proof*⟩

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 $\times_a$ *uint32-nat-assn*›
  ⟨*proof*⟩

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* $\times_a$ *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} $\to_a$ *lits-with-max-assn*›
    (**is** ‹*?c* ∈ [*?pre*]$_a$ *?im* → *?f*›)
⟨*proof*⟩

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} $\to_a$ *sint64-nat-assn*›
  ⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *atom-of-value-hnr*[*sepref-fr-rules*]:
‹(*Mreturn o* (λ*x*. *x*), *RETURN o atom-of-value*) ∈ [λ*n*. *n* < *2* ^*31*]$_a$ (*uint32-nat-assn*)^d → *atom-assn*›
⟨*proof*⟩

sepref-register *atom-of-value*

lemma [*sepref-gen-algo-rules*]: ‹*GEN-ALGO* (*Pos 0*) (*is-init unat-lit-assn*)›
⟨*proof*⟩

schematic-goal *mk-free-lbd-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE marked-struct-assn ?fr*›
⟨*proof*⟩

sepref-def *reduce-interval-init-impl*
is ‹*uncurry0* (*RETURN reduce-interval-init*)›
:: ‹*unit-assn*^{k} →$_a$ *word64-assn*›
⟨*proof*⟩

sepref-def *inprocessing-interval-init-impl*
is ‹*uncurry0* (*RETURN inprocessing-interval-init*)›
:: ‹*unit-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

sepref-def *rephasing-end-of-initial-phase-impl*
is ‹*uncurry0* (*RETURN rephasing-end-of-initial-phase*)›
:: ‹*unit-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

sepref-def *rephasing-initial-phase-impl*
is ‹*uncurry0* (*RETURN rephasing-initial-phase*)›
:: ‹*unit-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

sepref-def *subsuming-length-initial-phase-impl*
is ‹*uncurry0* (*RETURN subsuming-length-initial-phase*)›
:: ‹*unit-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

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} $\rightarrow_a$ *heuristic-int-assn*›
  ⟨*proof*⟩

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*)
    })›
    ⟨*proof*⟩

sepref-def *finalize-vmtf-init-code*
  is ‹*finalize-vmtf-init*›
  :: ‹*vmtf-init-assn*^{d} $\rightarrow_a$ *vmtf-assn*›
  ⟨*proof*⟩

sepref-def *finalize-bump-init-code*
  is ‹*finalize-bump-init*›
  :: ‹*heuristic-bump-init-assn*^{d} $\rightarrow_a$ *heuristic-bump-assn*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

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} $\rightarrow_a$ *trail-pol-fast-assn*›
  ⟨*proof*⟩

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} $\rightarrow_a$ *marked-struct-assn*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

sepref-register *empty-mark-struct combine-conflict combine-ccach combine-lcount*

lemma *init-state-wl-D′-alt-def*:
  ‹*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 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 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*)
    })›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

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*›
⟨*proof*⟩

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 =
    (λS. 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-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 (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 = []›
  ⟨proof⟩
```

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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'*›
⟨*proof*⟩

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*#}›
⟨*proof*⟩

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*#}›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)
›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *arena-act-pre-mark-used*:

‹*arena-act-pre arena C* ⟹
arena-act-pre (*mark-used arena C*) *C*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*))›
⟨*proof*⟩

lemma (**in** −) *not-ge-Suc0*: ‹¬*Suc 0* ≤ *n* ⟷ *n* = *0*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*⟩

lemma *phase-saving-le*: ‹*phase-saving* $\mathcal{A}$ *φ* ⟹ *A* ∈# $\mathcal{A}$ ⟹ *A* < *length φ*›
‹*phase-saving* $\mathcal{A}$ *φ* ⟹ *B* ∈# $\mathcal{L}_{all}$ $\mathcal{A}$ ⟹ *atm-of B* < *length φ*›
⟨*proof*⟩

lemma *trail-pol-nempty*: ‹¬((*[]*, *aa*, *ab*, *ac*, *ad*, *b*), *L* # *ys*) ∈ *trail-pol* $\mathcal{A}$›
⟨*proof*⟩

lemma *is-decided-hd-trail-wl-heur-hd-get-trail*:
‹(*RETURN o is-decided-hd-trail-wl-heur*, *RETURN o* (λ*M*. *is-decided* (*hd* (*get-trail-wl M*))))
∈ [λ*M*. *get-trail-wl M* ≠ *[]*]$_f$ *twl-st-heur-conflict-ana' r lcount* → ⟨*bool-rel*⟩ *nres-rel*›
⟨*proof*⟩

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
}
›
⟨*proof*⟩

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*⟩

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
    }›
  ⟨proof⟩

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⟩

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 →_a bool1-assn›
  ⟨proof⟩

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›
  :: ‹[(λS. fst S ≠ [] ∧ last-trail-pol-pre S)]_a trail-pol-fast-assn^k → bool1-assn›
  ⟨proof⟩

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 = ‹(λS. 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 (λM - - - - - - - - - - - - - - - - -. fst M ≠ [] ∧ last-trail-pol-pre M) = is-decided-hd-trail-wl-heur-pre›
  ⟨proof⟩


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} →$_a$ (*unat-lit-assn* ×$_a$ *sint64-nat-assn*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)
})›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)
})⟩
⟨*proof*⟩

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*⟩
⟨*proof*⟩

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*⟩
⟨*proof*⟩

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*›
⟨*proof*⟩

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
}›
⟨*proof*⟩

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*)
})

›
⟨proof⟩

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*›
⟨proof⟩

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*›
⟨proof⟩

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*›
⟨proof⟩

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*›
⟨proof⟩

lemma *le-unat64-max-minus-4-unat64-max*: ‹*a* ≤ *snat64-max* − *MIN-HEADER-SIZE* ⟹ *Suc a* < *max-snat 64*›
⟨proof⟩

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-watch-list-heur2-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)
})›
⟨proof⟩

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)›
⟨proof⟩

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›
⟨proof⟩

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›
⟨proof⟩

lemma [*sepref-fr-rules*]: ‹(Mreturn o Tuple17-get-d, RETURN o literals-to-update-wl-heur) ∈ isasat-bounded-assn^k
→$_a$ sint64-nat-assn›
⟨proof⟩

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)
}›
⟨proof⟩

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} $\rightarrow_a$ *isasat-bounded-assn* $\times_a$ *unat-lit-assn*›
  ⟨*proof*⟩

lemma *unit-propagation-outer-loop-wl-D-heur-fast*:
  ‹*length* (*get-clauses-wl-heur x*) $\leq$ *snat64-max* $\Longrightarrow$
      unit-propagation-outer-loop-wl-D-heur-inv x s' $\Longrightarrow$
      length (*get-clauses-wl-heur a1'*) =
      length (*get-clauses-wl-heur s'*) $\Longrightarrow$
      length (*get-clauses-wl-heur s'*) $\leq$ *snat64-max*›
  ⟨*proof*⟩

lemma *unit-propagation-outer-loop-wl-D-invI*:
  ‹*unit-propagation-outer-loop-wl-D-heur-inv* S_0 *S* $\Longrightarrow$
    isa-length-trail-pre (*get-trail-wl-heur S*)›
  ⟨*proof*⟩

sepref-def *unit-propagation-outer-loop-wl-D-fast-code*
  is ‹*unit-propagation-outer-loop-wl-D-heur*›
  :: ‹[λ*S*. *length* (*get-clauses-wl-heur S*) $\leq$ *snat64-max*]$_a$ *isasat-bounded-assn*^{d} $\rightarrow$ *isasat-bounded-assn*›
  ⟨*proof*⟩

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*⟩

end
theory *IsaSAT-Decide*
imports *IsaSAT-Decide-Defs*
begin

lemma (**in** −)*not-is-None-not-None*: ‹¬*is-None s* ⟹ *s* ≠ *None*›
⟨*proof*⟩

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* 𝒜 *focused M*;
RETURN (*L*, *Bump-Heuristics hstable* (*update-next-search L focused*) *foc tobmp*)
} *else do* {
(*L*, *hstable*) ← *acids-find-next-undef* 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* 𝒜 = (λ*M vm*. *do*{
(*L*, *vm*) ← *bump-find-next-undef* 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* 𝒜)) ∈
Id ×*r trail-pol* 𝒜 →*f* ⟨⟨*nat-rel*⟩*option-rel* ×*r Id*⟩*nres-rel* ›
⟨*proof*⟩

lemma *isa-vmtf-find-next-undef-vmtf-find-next-undef*:
‹(*uncurry isa-vmtf-find-next-undef-upd*, *uncurry* (*bump-find-next-undef-upd* 𝒜)) ∈
trail-pol 𝒜 ×*r Id* →*f*
⟨*trail-pol* 𝒜 ×*f Id* ×*f* ⟨*nat-rel*⟩*option-rel*⟩*nres-rel* ›
⟨*proof*⟩
term *isa-bump-find-next-undef*
term *bump-find-next-undef*
lemma *bump-find-next-undef-ref*:
assumes

vmtf: ‹$x \in$ *bump-heur* $\mathcal{A}$ M›
shows ‹*bump-find-next-undef* $\mathcal{A}$ x M
$\leq \Downarrow$ *Id* (*SPEC* (λ(L, *bmp*).
($L \neq$ *None* $\longrightarrow$ *bmp* $\in$ *bump-heur* $\mathcal{A}$ (*Decided* (*Pos* (*the* L)) # M)) $\wedge$
($L =$ *None* $\longrightarrow$ *bmp* $\in$ *bump-heur* $\mathcal{A}$ M) $\wedge$
($L =$ *None* $\longrightarrow$ ($\forall$ $L \in \#$ $\mathcal{L}_{all}$ $\mathcal{A}$. *defined-lit* M L)) $\wedge$
($L \neq$ *None* $\longrightarrow$ *Pos* (*the* L) $\in \#$ $\mathcal{L}_{all}$ $\mathcal{A}$ $\wedge$ *undefined-lit* M (*Pos* (*the* L)))))›
⟨*proof*⟩

definition *find-undefined-atm*
:: ‹*nat multiset* $\Rightarrow$ (*nat*,*nat*) *ann-lits* $\Rightarrow$ *bump-heuristics* $\Rightarrow$
(((*nat*,*nat*) *ann-lits* $\times$ *bump-heuristics*) $\times$ *nat option*) *nres*›
where
‹*find-undefined-atm* $\mathcal{A}$ M - = *SPEC*(λ((M', *vm*), L).
($L \neq$ *None* $\longrightarrow$ *Pos* (*the* L) $\in \#$ $\mathcal{L}_{all}$ $\mathcal{A}$ $\wedge$ *undefined-atm* M (*the* L)) $\wedge$
($L =$ *None* $\longrightarrow$ ($\forall$ $K \in \#$ $\mathcal{L}_{all}$ $\mathcal{A}$. *defined-lit* M K)) $\wedge$ $M = M' \wedge$
($L =$ *None* $\longrightarrow$ *vm* $\in$ *bump-heur* $\mathcal{A}$ M)$\wedge$
($L \neq$ *None* $\longrightarrow$ *vm* $\in$ *bump-heur* $\mathcal{A}$ (*Decided* (*Pos* (*the* L)) # M)))›

definition *lit-of-found-atm-D-pre* **where**
‹*lit-of-found-atm-D-pre* $\mathcal{A}$ = (λ((φ, -), L). $L \neq$ *None* $\longrightarrow$
(*the* $L \leq$ *unat32-max div 2* $\wedge$ *the* $L \in \#$ $\mathcal{A}$ $\wedge$ *phase-save-heur-rel* $\mathcal{A}$ φ))›

lemma *lit-of-found-atm-D-pre*:
‹*heuristic-rel* $\mathcal{A}$ *heur* $\Longrightarrow$ *isasat-input-bounded* $\mathcal{A}$ $\Longrightarrow$ ($L \neq$ *None* $\Longrightarrow$ *the* $L \in \#$ $\mathcal{A}$) $\Longrightarrow$
get-saved-phase-option-heur-pre (L) (*get-content heur*)›
⟨*proof*⟩

definition *find-unassigned-lit-wl-D-heur-pre* **where**
‹*find-unassigned-lit-wl-D-heur-pre* S $\longleftrightarrow$
(
$\exists$ T U.
(S, T) $\in$ *state-wl-l None* $\wedge$
(T, U) $\in$ *twl-st-l None* $\wedge$
twl-struct-invs U $\wedge$
literals-are-$\mathcal{L}_{in}$ (*all-atms-st* S) S $\wedge$
get-conflict-wl S = *None*
)›

definition *twl-st-heur-decide-find* :: ‹*nat literal* $\Rightarrow$ (*isasat* $\times$ *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*›
⟨*proof*⟩

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* $S = u\} \to$
⟨{((*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*⟩

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)›
⟨*proof*⟩

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)›

⟨*proof*⟩

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*⟩

lemma *bind-triple-unfold*:
‹*do* {
((*M*, *vm*), *L*) ← (*P* :: - *nres*);
f ((*M*, *vm*), *L*)
} =
do {
x ← *P*;
f x
}›
⟨*proof*⟩

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*›
⟨*proof*⟩

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})›
⟨*proof*⟩

sepref-def *decide-lit-wl-fast-code*
is ‹*uncurry decide-lit-wl-heur*›
:: ‹*unat-lit-assn*^{k} ∗$_a$ *isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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} →$_a$ *atom.option-assn*›

⟨*proof*⟩

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} *$\to_a$ atom.option-assn* *$\times_a$ heuristic-bump-assn*›
⟨*proof*⟩

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} *$\to_a$ vmtf-assn*›
⟨*proof*⟩

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} *$\to_a$ (trail-pol-fast-assn* *$\times_a$ heuristic-bump-assn)* *$\times_a$ atom.option-assn*›
⟨*proof*⟩

lemma *find-unassigned-lit-wl-D-heur2-alt-def*:
‹*find-unassigned-lit-wl-D-heur2* = (*λS. 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*)
})›
⟨*proof*⟩
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} *$\to_a$ isasat-bounded-assn* *$\times_a$ atom.option-assn*›
⟨*proof*⟩

sepref-definition *get-next-phase-heur-stats-impl′*
is ‹*uncurry2* (*λS C′ D′. get-next-phase-heur C′ D′ S*)›
:: ‹[*uncurry2* (*λS C D. True*)]$_a$ *heuristic-assn*^{k} *$*_a$ *bool1-assn*^{k} *$*_a$ *atom-assn*^{k} *$\to$ bool1-assn*›
⟨*proof*⟩

definition *get-next-phase-st′-impl* :: ‹*twl-st-wll-trail-fast2* ⇒ -› **where**
‹*get-next-phase-st′-impl* = (*λN 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′ = ‹ *λC D S. get-next-phase-heur C D S*› **and**
f = ‹*λC D S. get-next-phase-heur-stats-impl C D S*› **and**
x-assn = ‹*bool1-assn*› **and**
P = ‹(*λ- - -. True*)›
rewrites

‹(λN C D. read-heur-wl-heur-code (get-next-phase-heur-stats-impl C D) N) = get-next-phase-st′-impl›
and
‹(λN C D. read-heur-wl-heur (get-next-phase-heur C D) N) = get-next-phase-st′›
⟨proof⟩

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›
⟨proof⟩

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›
⟨proof⟩

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*⟩

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*›
  ⟨*proof*⟩

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*⟩

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*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

sepref-def *isasat-fast-impl*
is ‹*RETURN o isasat-fast*›
:: ‹*isasat-bounded-assn*^{k} →$_a$ *bool1-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

global-interpretation *twl-restart-ops id*
⟨*proof*⟩

global-interpretation *twl-restart id*
⟨*proof*⟩

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* = (λS. *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* = (λS.
RES UNIV)›

definition *number-clss-to-keep-impl* :: ‹*isasat* ⇒ *nat nres*› **where**
‹*number-clss-to-keep-impl* = (λS.
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 = (λS. 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 = (λf (I :: 'a ⇒ bool) P (ns :: (nat, nat) vmtf-node list, n) x. do {
      (-, x) ← WHILE_T^λ(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 = (λS. 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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*$_T^{\textit{find-local-restart-target-level-int-inv bmp cs}}$
(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*$_T$ $^{\lambda(\text{-}, i, j, \text{-}).\ i \le j \wedge j \le n}$
(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* = (λ*S*. *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*⟩

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*›

⟨*proof*⟩

lemma *twl-st-heur-restartD*:
‹$x \in$ *twl-st-heur-restart* $\Longrightarrow$ $x \in$ *twl-st-heur-restart-ana* (*length* (*get-clauses-wl-heur* (*fst* x)))›
⟨*proof*⟩

named-theorems *twl-st-heur-restart*

lemma [*twl-st-heur-restart*]:
assumes ‹(S, T) $\in$ *twl-st-heur-restart*›
shows ‹(*get-trail-wl-heur* S, *get-trail-wl* T) $\in$ *trail-pol* (*all-init-atms-st* T)›
⟨*proof*⟩

lemma *trail-pol-literals-are-in-*$\mathcal{L}_{in}$*-trail*:
‹(M', M) $\in$ *trail-pol* $\mathcal{A}$ $\Longrightarrow$ *literals-are-in-*$\mathcal{L}_{in}$*-trail* $\mathcal{A}$ M›
⟨*proof*⟩

lemma *refine-generalise1*: ‹$A \leq B \Longrightarrow$ *do* {$x \leftarrow B$; C x} $\leq D \Longrightarrow$ *do* {$x \leftarrow A$; C x} $\leq$ (D:: '*a nres*)›
⟨*proof*⟩

lemma *refine-generalise2*: $A \leq B \Longrightarrow$ *do* {$x \leftarrow$ *do* {$x \leftarrow B$; A' x}; C x} $\leq D \Longrightarrow$
do {$x \leftarrow$ *do* {$x \leftarrow A$; A' x}; C x} $\leq$ (D:: '*a nres*)
⟨*proof*⟩

lemma *trail-pol-no-dup*: ‹(M, M') $\in$ *trail-pol* $\mathcal{A}$ $\Longrightarrow$ *no-dup* M'›
⟨*proof*⟩

definition *remove-all-annot-true-clause-imp-wl-D-pre*
:: ‹*nat multiset* $\Rightarrow$ *nat literal* $\Rightarrow$ *nat twl-st-wl* $\Rightarrow$ *bool*›
where
‹*remove-all-annot-true-clause-imp-wl-D-pre* $\mathcal{A}$ L S $\longleftrightarrow$ ($L \in\#$ $\mathcal{L}_{all}$ $\mathcal{A}$)›

definition *remove-all-annot-true-clause-imp-wl-D-heur-pre* **where**
‹*remove-all-annot-true-clause-imp-wl-D-heur-pre* L S $\longleftrightarrow$
($\exists S'$. (S, S') $\in$ *twl-st-heur-restart*
$\wedge$ *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* $b \geq a$ *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* $\Longrightarrow$ *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* $\Longrightarrow$ *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*)›
⟨*proof*⟩

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*⟩

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*⟩

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*›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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**

‹$C \in\# \mathit{dom\text{-}m}\ (\mathit{get\text{-}clauses\text{-}wl}\ T) \Longrightarrow \mathit{arena\text{-}status}\ (\mathit{get\text{-}clauses\text{-}wl\text{-}heur}\ S)\ C = \mathit{LEARNED} \longleftrightarrow \neg \mathit{irred}\ (\mathit{get\text{-}clauses\text{-}wl}\ T)\ C$›
‹$C \in\# \mathit{dom\text{-}m}\ (\mathit{get\text{-}clauses\text{-}wl}\ T) \Longrightarrow \mathit{arena\text{-}length}\ (\mathit{get\text{-}clauses\text{-}wl\text{-}heur}\ S)\ C = \mathit{length}\ (\mathit{get\text{-}clauses\text{-}wl}\ T \propto C)$›
⟨*proof*⟩

lemma [*twl-st-heur-restart*]:
assumes
‹$(S,\ T) \in \mathit{twl\text{-}st\text{-}heur\text{-}restart\text{-}ana}\ r$› **and**
‹$C \in \mathit{set}\ (\mathit{get\text{-}avdom}\ S)$›
shows ‹$\mathit{clause\text{-}not\text{-}marked\text{-}to\text{-}delete\text{-}heur}\ S\ C \longleftrightarrow (C \in\# \mathit{dom\text{-}m}\ (\mathit{get\text{-}clauses\text{-}wl}\ T))$› **and**
‹$C \in\# \mathit{dom\text{-}m}\ (\mathit{get\text{-}clauses\text{-}wl}\ T) \Longrightarrow \mathit{arena\text{-}lit}\ (\mathit{get\text{-}clauses\text{-}wl\text{-}heur}\ S)\ C = \mathit{get\text{-}clauses\text{-}wl}\ T \propto C\ !\ 0$›**and**
‹$C \in\# \mathit{dom\text{-}m}\ (\mathit{get\text{-}clauses\text{-}wl}\ T) \Longrightarrow \mathit{arena\text{-}status}\ (\mathit{get\text{-}clauses\text{-}wl\text{-}heur}\ S)\ C = \mathit{LEARNED} \longleftrightarrow \neg \mathit{irred}\ (\mathit{get\text{-}clauses\text{-}wl}\ T)\ C$›
‹$C \in\# \mathit{dom\text{-}m}\ (\mathit{get\text{-}clauses\text{-}wl}\ T) \Longrightarrow \mathit{arena\text{-}length}\ (\mathit{get\text{-}clauses\text{-}wl\text{-}heur}\ S)\ C = \mathit{length}\ (\mathit{get\text{-}clauses\text{-}wl}\ T \propto C)$›
⟨*proof*⟩

lemma [*twl-st-heur-restart*]:
assumes
‹$(S,\ T) \in \mathit{twl\text{-}st\text{-}heur\text{-}restart}$› **and**
‹$C \in \mathit{set}\ (\mathit{get\text{-}tvdom}\ S)$›
shows ‹$\mathit{clause\text{-}not\text{-}marked\text{-}to\text{-}delete\text{-}heur}\ S\ C \longleftrightarrow (C \in\# \mathit{dom\text{-}m}\ (\mathit{get\text{-}clauses\text{-}wl}\ T))$› **and**
‹$C \in\# \mathit{dom\text{-}m}\ (\mathit{get\text{-}clauses\text{-}wl}\ T) \Longrightarrow \mathit{arena\text{-}lit}\ (\mathit{get\text{-}clauses\text{-}wl\text{-}heur}\ S)\ C = \mathit{get\text{-}clauses\text{-}wl}\ T \propto C\ !\ 0$›**and**
‹$C \in\# \mathit{dom\text{-}m}\ (\mathit{get\text{-}clauses\text{-}wl}\ T) \Longrightarrow \mathit{arena\text{-}status}\ (\mathit{get\text{-}clauses\text{-}wl\text{-}heur}\ S)\ C = \mathit{LEARNED} \longleftrightarrow \neg \mathit{irred}\ (\mathit{get\text{-}clauses\text{-}wl}\ T)\ C$›
‹$C \in\# \mathit{dom\text{-}m}\ (\mathit{get\text{-}clauses\text{-}wl}\ T) \Longrightarrow \mathit{arena\text{-}length}\ (\mathit{get\text{-}clauses\text{-}wl\text{-}heur}\ S)\ C = \mathit{length}\ (\mathit{get\text{-}clauses\text{-}wl}\ T \propto C)$›
⟨*proof*⟩

lemma [*twl-st-heur-restart*]:
assumes
‹$(S,\ T) \in \mathit{twl\text{-}st\text{-}heur\text{-}restart\text{-}ana}\ r$› **and**
‹$C \in \mathit{set}\ (\mathit{get\text{-}tvdom}\ S)$›
shows ‹$\mathit{clause\text{-}not\text{-}marked\text{-}to\text{-}delete\text{-}heur}\ S\ C \longleftrightarrow (C \in\# \mathit{dom\text{-}m}\ (\mathit{get\text{-}clauses\text{-}wl}\ T))$› **and**
‹$C \in\# \mathit{dom\text{-}m}\ (\mathit{get\text{-}clauses\text{-}wl}\ T) \Longrightarrow \mathit{arena\text{-}lit}\ (\mathit{get\text{-}clauses\text{-}wl\text{-}heur}\ S)\ C = \mathit{get\text{-}clauses\text{-}wl}\ T \propto C\ !\ 0$›**and**
‹$C \in\# \mathit{dom\text{-}m}\ (\mathit{get\text{-}clauses\text{-}wl}\ T) \Longrightarrow \mathit{arena\text{-}status}\ (\mathit{get\text{-}clauses\text{-}wl\text{-}heur}\ S)\ C = \mathit{LEARNED} \longleftrightarrow \neg \mathit{irred}\ (\mathit{get\text{-}clauses\text{-}wl}\ T)\ C$›
‹$C \in\# \mathit{dom\text{-}m}\ (\mathit{get\text{-}clauses\text{-}wl}\ T) \Longrightarrow \mathit{arena\text{-}length}\ (\mathit{get\text{-}clauses\text{-}wl\text{-}heur}\ S)\ C = \mathit{length}\ (\mathit{get\text{-}clauses\text{-}wl}\ T \propto C)$›
⟨*proof*⟩

lemma *number-clss-to-keep-impl-number-clss-to-keep*:
‹$(\mathit{number\text{-}clss\text{-}to\text{-}keep\text{-}impl},\ \mathit{number\text{-}clss\text{-}to\text{-}keep}) \in \mathit{Id} \rightarrow_f \langle \mathit{nat\text{-}rel} \rangle \mathit{nres\text{-}rel}$›
⟨*proof*⟩

lemma *in-set-delete-index-and-swapD*:
‹$x \in \mathit{set}\ (\mathit{delete\text{-}index\text{-}and\text{-}swap}\ \mathit{xs}\ i) \Longrightarrow x \in \mathit{set}\ \mathit{xs}$›
⟨*proof*⟩

lemma *delete-index-vdom-heur-twl-st-heur-restart-ana*:
‹$(S,\ T) \in \mathit{twl\text{-}st\text{-}heur\text{-}restart\text{-}ana}\ r \Longrightarrow i < \mathit{length}\ (\mathit{get\text{-}tvdom}\ S) \Longrightarrow$

get-tvdom S ! i ∉# dom-m (get-clauses-wl T) ⟹
(delete-index-vdom-heur i S, T) ∈ twl-st-heur-restart-ana r›
⟨*proof*⟩

lemma *incr-wasted-st*:
assumes
‹(S, T) ∈ twl-st-heur-restart-ana r›
shows *‹(incr-wasted-st C S, T) ∈ twl-st-heur-restart-ana r›*
⟨*proof*⟩

lemma *twl-st-heur-restart-same-annotD*:
‹(S, T) ∈ twl-st-heur-restart ⟹ Propagated L C ∈ set (get-trail-wl T) ⟹
Propagated L C′ ∈ set (get-trail-wl T) ⟹ C = C′›
‹(S, T) ∈ twl-st-heur-restart ⟹ Propagated L C ∈ set (get-trail-wl T) ⟹
Decided L ∈ set (get-trail-wl T) ⟹ False›
⟨*proof*⟩

lemma $\mathcal{L}_{all}$*-mono*:
‹set-mset A ⊆ set-mset B ⟹ L ∈# $\mathcal{L}_{all}$ A ⟹ L ∈# $\mathcal{L}_{all}$ B›
⟨*proof*⟩

lemma *all-lits-of-mm-mono2*:
‹x ∈# (all-lits-of-mm A) ⟹ set-mset A ⊆ set-mset B ⟹ x ∈# (all-lits-of-mm B)›
⟨*proof*⟩

lemma $\mathcal{L}_{all}$*-init-all*:
‹L ∈# $\mathcal{L}_{all}$ (all-init-atms-st x1a) ⟹ L ∈# $\mathcal{L}_{all}$ (all-atms-st x1a)›
⟨*proof*⟩

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))›
⟨*proof*⟩

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)›
⟨*proof*⟩

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*⟩

lemmas *trail-pol-replace-annot-in-trail-spec2* =
trail-pol-replace-annot-in-trail-spec[*of* ‹− -›, *simplified*]

lemma $\mathcal{L}_{all}$*-ball-all*:
‹(∀ *L* ∈# $\mathcal{L}_{all}$ (*all-atms N NUE*). *P L*) = (∀ *L* ∈# *all-lits N NUE*. *P L*)›
‹(∀ *L* ∈# $\mathcal{L}_{all}$ (*all-init-atms N NUE*). *P L*) = (∀ *L* ∈# *all-init-lits N NUE*. *P L*)›
⟨*proof*⟩

lemma *clss-size-corr-restart-intro3*[*intro*]:
‹*clss-size-corr-restart N NE UE NEk UEk NS US N0 U0 lcount* ⟹
clss-size-corr-restart N NE UE NEk UEk NS {#} *N0* {#} (*clss-size-resetUS0 lcount*)›
⟨*proof*⟩

lemma *twl-st-heur-restart-ana-US-empty*:
‹*NO-MATCH* {#} *US* ⟹ *NO-MATCH* {#} *U0* ⟹ *NO-MATCH* {#} *UE* ⟹ (*S*, *M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *W*, *Q*) ∈ *twl-st-heur-restart-ana r* ⟹
(*set-learned-count-wl-heur* (*clss-size-resetUS0* (*get-learned-count S*)) *S*, *M*, *N*, *D*, *NE*, {#}, *NEk*, *UEk*, *NS*, {#}, *N0*, {#}, *W*, *Q*)
∈ *twl-st-heur-restart-ana r*›
⟨*proof*⟩

fun *equality-except-trail-empty-US-wl* :: ‹'*v twl-st-wl* ⇒ '*v twl-st-wl* ⇒ *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'*) ⟷
N = *N'* ∧ *D* = *D'* ∧ *NE* = *NE'* ∧ *NEk* = *NEk'* ∧ *UEk* = *UEk'* ∧ *NS* = *NS'* ∧ *US* = {#} ∧ *UE* = {#} ∧
N0' = *N0* ∧ *U0* = {#} ∧ *WS* = *WS'* ∧ *Q* = *Q'*›

lemma *equality-except-conflict-wl-get-clauses-wl*:
‹*equality-except-conflict-wl S Y* ⟹ *get-clauses-wl S* = *get-clauses-wl Y*› **and**
equality-except-conflict-wl-get-trail-wl:
‹*equality-except-conflict-wl S Y* ⟹ *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* ⟹ *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*⟹ *get-clauses-wl S* = *get-clauses-wl Y*›
⟨*proof*⟩

lemma *isasat-replace-annot-in-trail-replace-annot-in-trail-spec*:
‹(((*L*, *C*), *S*), ((*L'*, *C'*), *S'*)) ∈ *Id* ×$_f$ *Id* ×$_f$ *twl-st-heur-restart-ana' r u* ⟹
isasat-replace-annot-in-trail L C S ≤
⇓{(*U*, *U'*). (*U*, *U'*) ∈ *twl-st-heur-restart-ana' r u* ∧
get-clauses-wl-heur U = *get-clauses-wl-heur S* ∧
get-learned-count U = *clss-size-resetUS0* (*get-learned-count S*) ∧
get-vdom U = *get-vdom S* ∧
equality-except-trail-empty-US-wl U' S'}
(*replace-annot-wl L' C' S'*)›
⟨*proof*⟩

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*)

$\in$ *set* (*get-all-ann-decomposition* (*get-trail-wl T*))) $\wedge$
count-decided M1 = *0* $\wedge$ *k* = *length M1*)›
⟨*proof*⟩

lemma *twl-st-heur-restart-isa-length-trail-get-trail-wl*:
‹(*S*, *T*) $\in$ *twl-st-heur-restart-ana r* $\Longrightarrow$ *mop-isa-length-trail* (*get-trail-wl-heur S*) = *RETURN* (*length* (*get-trail-wl T*))›
⟨*proof*⟩

lemma *twl-st-heur-restart-count-decided-st-alt-def*:
fixes *S* :: *isasat*
shows ‹(*S*, *T*) $\in$ *twl-st-heur-restart-ana r* $\Longrightarrow$ *count-decided-st-heur S* = *count-decided* (*get-trail-wl T*)›
⟨*proof*⟩

lemma *twl-st-heur-restart-trailD*:
‹(*S*, *T*) $\in$ *twl-st-heur-restart-ana r* $\Longrightarrow$
(*get-trail-wl-heur S*, *get-trail-wl T*) $\in$ *trail-pol* (*all-init-atms-st T*)›
⟨*proof*⟩

lemma *no-dup-nth-proped-dec-notin*:
‹*no-dup M* $\Longrightarrow$ *k* < *length M* $\Longrightarrow$ *M* ! *k* = *Propagated L C* $\Longrightarrow$ *Decided L* $\notin$ *set M*›
⟨*proof*⟩

lemma *get-literal-and-reason*:
assumes
‹((*k*, *S*), *k'*, *T*) $\in$ *nat-rel* $\times_f$ *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'* $\wedge$ *C'* = *the C* $\wedge$ *C* ≠ *None*}
(*SPEC* (λ*p*. *rev* (*get-trail-wl T*) ! *k'* = *Propagated* (*fst p*) (*snd p*)))›
⟨*proof*⟩

lemma *red-in-dom-number-of-learned-ge1*: ‹*C'* ∈# *dom-m baa* $\Longrightarrow$ ¬ *irred baa C'* $\Longrightarrow$ *Suc 0* ≤ *size* (*learned-clss-l baa*)›
⟨*proof*⟩

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*) $\in$ *set* (*get-all-ann-decomposition M*) $\wedge$
count-decided M' = *0*) $\wedge$
(*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)})›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

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*⟩

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*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*⟩

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*)
}
})›
⟨*proof*⟩

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*⟩

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* (λ*c*. (*c*, *log-clause T C*′) ∈ *unit-rel*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*⟩

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 (λi. 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
})›
⟨proof⟩

definition *remove-one-annot-true-clause-imp-wl-D-heur* :: ‹*isasat* ⇒ *isasat nres*›
where
‹*remove-one-annot-true-clause-imp-wl-D-heur* = (λS. 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›
⟨proof⟩

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 -›)
⟨proof⟩

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 = (λf $\mathcal{A}_0$ I P x. *do* {
  $\mathcal{A}$ ← *SPEC*($\lambda\mathcal{A}$. *set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{A}_0$ ∧ *distinct-mset* $\mathcal{A}$);
  (-, x) ← *WHILE*$_T$$^{\lambda(\mathcal{A},\ x).\ I\ \mathcal{A}\ x}$
    ($\lambda(\mathcal{B},\ x).\ \mathcal{B} \neq \{\#\} \wedge P\ x$)
    ($\lambda(\mathcal{B},\ x).$ *do* {
      ASSERT($\mathcal{B} \neq \{\#\}$);
      A ← *SPEC* ($\lambda A.\ A \in\#\ \mathcal{B}$);
      x ← f A x;
      RETURN (*remove1-mset* A $\mathcal{B}$, x)
    })
    ($\mathcal{A}$, x);
  RETURN x
})›

definition *iterate-over-*$\mathcal{L}_{all}$ :: ‹-› **where**
  iterate-over-$\mathcal{L}_{all}$*-alt-def*: ‹*iterate-over-*$\mathcal{L}_{all}$ f $\mathcal{A}$ I = *iterate-over-*$\mathcal{L}_{all}$ C f $\mathcal{A}$ I (λ-. *True*)›

lemmas *iterate-over-*$\mathcal{L}_{all}$*-def* =
  iterate-over-$\mathcal{L}_{all}$*-alt-def*[*unfolded iterate-over-*$\mathcal{L}_{all}$*C-def simp-thms*]

lemma *iterate-over-VMTFC-iterate-over-*$\mathcal{L}_{all}$ C:
  fixes x :: $'a$
  assumes *vmtf*: ‹(*ns*, *m*, *fst-As*, *lst-As*, *next-search*) ∈ *vmtf* $\mathcal{A}$ M› **and**
    nempty: ‹$\mathcal{A} \neq \{\#\}$› ‹*isasat-input-bounded* $\mathcal{A}$› **and**
    II': ‹$\bigwedge x\ \mathcal{B}$. *set-mset* $\mathcal{B}$ ⊆ *set-mset* $\mathcal{A}$ ⟹ I' $\mathcal{B}$ x ⟹ I x› **and**
    ‹$\bigwedge x$. I x ⟹ P x = Q x›
  shows ‹*iterate-over-VMTFC* f I P (*ns*, *Some fst-As*) x ≤ ⇓ *Id* (*iterate-over-*$\mathcal{L}_{all}$ C f $\mathcal{A}$ I' Q x)›
⟨*proof*⟩

lemma *iterate-over-VMTF-iterate-over-*$\mathcal{L}_{all}$:
  fixes x :: $'a$
  assumes *vmtf*: ‹(*ns*, *m*, *fst-As*, *lst-As*, *next-search*) ∈ *vmtf* $\mathcal{A}$ M› **and**
    nempty: ‹$\mathcal{A} \neq \{\#\}$› ‹*isasat-input-bounded* $\mathcal{A}$› ‹$\bigwedge x\ \mathcal{B}$. *set-mset* $\mathcal{B}$ ⊆ *set-mset* $\mathcal{A}$ ⟹ I' $\mathcal{B}$ x ⟹ I x›
  shows ‹*iterate-over-VMTF* f I (*ns*, *Some fst-As*) x ≤ ⇓ *Id* (*iterate-over-*$\mathcal{L}_{all}$ f $\mathcal{A}$ I' x)›
  ⟨*proof*⟩

definition *arena-is-packed* :: ‹*arena* ⇒ *nat clauses-l* ⇒ *bool*› **where**
‹*arena-is-packed arena N* ⟷ *length arena* = ($\sum$ C ∈# *dom-m N*. *length* (*N* ∝ *C*) + *header-size* (*N* ∝ *C*))›

lemma *arena-is-packed-empty*[*simp*]: ‹*arena-is-packed* [] *fmempty*›
  ⟨*proof*⟩

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*)›
  ⟨*proof*⟩

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*⟩

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)›
⟨*proof*⟩

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) ∧
correct-watching' (M, N', D, *NE*, *UE*, *NEk'*, *UEk'*, *NS'*, *US*, *N0*, *U0*, Q', *WS'*) ∧
literals-are-$\mathcal{L}_{in}$*'* (M, N', D, *NE*, *UE*, *NEk'*, *UEk'*, *NS'*, *US*, *N0*, *U0*, Q', *WS'*)))›

lemma *blits-in-*$\mathcal{L}_{in}$*'-restart-wl-spec0'*:
‹*literals-are-*$\mathcal{L}_{in}$*'* (a, *aq*, *ab*, *ac*, *ad*, *NEk*, *UEk*, *ae*, *af*, *N0*, *U0*, Q, b) ⟹
literals-are-$\mathcal{L}_{in}$*'* (a, *aq*, *ab*, *ac*, *ad*, *NEk*, *UEk*, *ae*, *af*, *N0*, *U0*, {#}, b)›
⟨*proof*⟩

abbreviation *twl-st-heur-restart'''u* **where**
‹*twl-st-heur-restart'''u r u* ≡
{(S, T). (S, T) ∈ *twl-st-heur-restart* ∧ *length* (*get-clauses-wl-heur S*) = r ∧
learned-clss-count S ≤ u}›

abbreviation *twl-st-heur-restart''''u* **where**
‹*twl-st-heur-restart''''u r u* ≡
{(S, T). (S, T) ∈ *twl-st-heur-restart* ∧ *length* (*get-clauses-wl-heur S*) ≤ r ∧
learned-clss-count S ≤ u}›

fun *correct-watching'''* :: ‹- ⇒ *'v twl-st-wl* ⇒ *bool*› **where**
‹*correct-watching'''* $\mathcal{A}$ (M, N, D, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, Q, W) ⟷
(∀ L ∈# *all-lits-of-mm* $\mathcal{A}$.
distinct-watched (W L) ∧
(∀ (i, K, b)∈#*mset* (W L).
i ∈# *dom-m N* ∧ K ∈ *set* (N ∝ i) ∧ K ≠ L ∧
correctly-marked-as-binary N (i, K, b)) ∧
fst ‘# *mset* (W L) = *clause-to-update* L (M, N, D, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, {#}, {#}))›

declare *correct-watching'''.simps*[*simp del*]

lemma *correct-watching'''-add-clause*:
assumes
corr: ‹*correct-watching'''* $\mathcal{A}$ ((*a*, *aa*, *CD*, *ac*, *ad*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, *Q*, *b*))› **and**
leC: ‹*2* ≤ *length C*› **and**
i-notin[*simp*]: ‹*i* ∉# *dom-m aa*› **and**
dist[*iff*]: ‹*C* ! *0* ≠ *C* ! *Suc 0*›
shows ‹*correct-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*)]))›
⟨*proof*⟩

lemma *rewatch-correctness*:
assumes *empty*: ‹⋀*L*. *L* ∈# *all-lits-of-mm* $\mathcal{A}$ ⟹ *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* $\mathcal{A}$)›
shows
‹*rewatch N W* ≤ *SPEC*(λ*W*. *correct-watching'''* $\mathcal{A}$ (*M*, *N*, *C*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, N_0, U_0, *Q*, *W*))›
⟨*proof*⟩

lemma *heuristic-rel-incr-restartI*[*intro*!]:
‹*heuristic-rel* $\mathcal{A}$ *heur* ⟹ *heuristic-rel* $\mathcal{A}$ (*incr-restart-phase-end end-of-phase heur*)›
⟨*proof*⟩

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*›
⟨*proof*⟩
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*›
⟨*proof*⟩

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'*›
⟨*proof*⟩

lemma *twl-st-heur-restart-state-simp*:
assumes ‹(*S*, *S'*) ∈ *twl-st-heur-restart*›
shows
twl-st-heur-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-state-simp-watched2: ‹*C* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st S'*) ⟹
nat-of-lit C < *length*(*get-watched-wl-heur S*)›
⟨*proof*⟩

lemma *twl-st-heur-restart-ana-state-simp*:
assumes ‹$(S,\ S') \in$ *twl-st-heur-restart-ana* u›
shows
twl-st-heur-restart-ana-state-simp-watched: ‹$C \in\#\ \mathcal{L}_{all}$ (*all-init-atms-st* S') $\Longrightarrow$
watched-by-int $S\ C =$ *watched-by* $S'\ C$›
‹$C \in\#\ \mathcal{L}_{all}$ (*all-init-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-restart-ana-state-simp-watched2: ‹$C \in\#\ \mathcal{L}_{all}$ (*all-init-atms-st* S') $\Longrightarrow$
nat-of-lit $C <$ *length*(*get-watched-wl-heur* S)›
⟨*proof*⟩

lemma *twl-st-heur-restart-ana-watchlist-in-vdom*:
‹*get-watched-wl-heur x2e* ! *nat-of-lit L* ! *x1d* = $(a,\ b) \Longrightarrow$
$(x2e,\ x2f) \in$ *twl-st-heur-restart-ana* $r \Longrightarrow L \in\#\ \mathcal{L}_{all}$ (*all-init-atms-st x2f*) $\Longrightarrow$
x1d < *length* (*get-watched-wl-heur x2e* ! *nat-of-lit L*) $\Longrightarrow$
$a \in$ *set* (*get-vdom x2e*)›
⟨*proof*⟩

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*
$(M',\ M) \in$ *trail-pol* (*all-init-atms-st* T) ∧
valid-arena N' N (*set* (*get-vdom-aivdom vdom*)) ∧
$(D',\ D) \in$ *option-lookup-clause-rel* (*all-init-atms-st* T) ∧
$(D = None \longrightarrow j \leq$ *length* $M)$ ∧
$Q =$ *uminus* ‘# *lit-of* ‘# *mset* (*drop* j (*rev* M)) ∧
$(W',\ W) \in \langle Id \rangle$*map-fun-rel* ($D_0$ (*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 $N \subseteq$ *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*
}›

⟨*proof*⟩

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*⟩

lemma D_0-*cong'*: ‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B}$ ⟹ *x* ∈ D_0 $\mathcal{A}$ ⟹ *x* ∈ D_0 $\mathcal{B}$›
⟨*proof*⟩
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 $\mathcal{B}$)›
⟨*proof*⟩

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*
⟨*proof*⟩
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*
⟨*proof*⟩
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*
⟨*proof*⟩

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
}›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *find-local-restart-target-level-st-alt-def*:
‹*find-local-restart-target-level-st* = (λ*S*. *do* {
find-local-restart-target-level-int (*get-trail-wl-heur S*) (*get-vmtf-heur S*)})›
⟨*proof*⟩

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 {
$(M, Q') \leftarrow SPEC(\lambda(M', Q').\ (\exists K\ M2.\ (Decided\ K\ \#\ M', M2) \in set\ (get\text{-}all\text{-}ann\text{-}decomposition\ M) \wedge$
$Q' = \{\#\}) \vee (M' = M \wedge Q' = Q))$;
RETURN $(M, N, D, NE, UE, NEk, UEk, NS, US, N0, U0, Q', W)$
}
})›
⟨*proof*⟩

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*⟩

lemma *distinct-mset-union-iff*:
‹*distinct-mset* (*xs* + *ys*) = (*distinct-mset xs* ∧ *distinct-mset ys* ∧ *set-mset xs* ∩ *set-mset ys* = {})›
⟨*proof*⟩

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*)›
⟨*proof*⟩

lemma *remove-deleted-clauses-from-avdom*:
assumes ‹*aivdom-inv-dec avdom0* (*dom-m N*)›
shows ‹*remove-deleted-clauses-from-avdom N avdom0* ≤ *SPEC*(λ*aivdom*. *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*⟩

lemma *arena-status-mark-unused*[*simp*]:
‹*arena-status* (*mark-unused arena C*) *D* = *arena-status arena D*›
⟨*proof*⟩

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*⟩

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* 𝒜 ⟹ *set-mset* (*all-atms N NUE*) = *set-mset* 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*)›
⟨*proof*⟩

lemma *bind-result-subst-iff*:
‹*P* ≤ ⇓ {(*a*,*b*). (*f a*, *b*) ∈ *R*} *g* ⟷
do {
a ← *P*;
RETURN (*f a*)
} ≤ ⇓ *R g*›
⟨*proof*⟩

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*⟩

lemma *mset-inter-eqD*: ‹*x1m* ∩# *af* = *xa* ∩# *af* ⟹
set-mset x1m ∩ *set-mset* (*af*) = *set-mset xa* ∩ *set-mset af*› **for** *x1m af xa*
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*
      (λ*vdom'*.
       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*⟩

lemma *specify-left-RES*:
  assumes *m* ≤ *RES* Φ
  assumes ⋀*x*. *x* ∈ Φ ⟹ *f x* ≤ *M*
  shows *do* { *x* ← *m*; *f x* } ≤ *M*
  ⟨*proof*⟩

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*⟩

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'*)›
  ⟨*proof*⟩

lemma *sort-clauses-by-score-invI*:
  ‹*valid-sort-clause-score-pre a b* ⟹
       mset b = *mset a2'* ⟹ *valid-sort-clause-score-pre a a2'*›
  ⟨*proof*⟩

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*)›
  ⟨*proof*⟩

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*›
⟨*proof*⟩

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*⟩

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*⟩

lemma *mark-to-delete-clauses-wl-D-heur-alt-def*:
‹*mark-to-delete-clauses-wl-D-heur* = (λ*S0*. *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$_T^{\lambda\text{-. }True}$ (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)
  })⟩
⟨proof⟩
```

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)
        })⟩
    ⟨proof⟩

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›
  ⟨proof⟩

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⟩

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*))
}›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *schedule-next-reduction-sttwl-st-heur*:
‹(*S*,*T*)∈*twl-st-heur* ⟹ (*schedule-next-reduction-st S*, *T*) ∈ *twl-st-heur*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

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*⟩

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* ∈# 𝒜› **and**
L'-L: ‹(*A'*, *A*) ∈ *nat-lit-lit-rel*› **and**
W: ‹(*W'* , *W*) ∈ ⟨*Id*⟩*map-fun-rel* (*D*$_0$ 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* ∈# $\mathcal{L}_{all}$ 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* 𝒜 *vdom0 arena W*)
(*cdcl-GC-clauses-prog-copy-wl-entry N* (*W A*) *A N'*)›
(**is** ‹- ≤ ⇓ (*?R*) -›)
⟨*proof*⟩

lemma *isasat-GC-clauses-prog-single-wl*:
assumes
‹(*X*, *X'*) ∈ *isasat-GC-refl* 𝒜 *vdom0 arena0*› **and**
X: ‹*X* = (*arena*, (*arena'*, *aivdom*), *W*)› ‹*X'* = (*N*, *N'*, *W'*)› **and**
L: ‹*A* ∈# 𝒜› **and**
st: ‹(*A*, *A'*) ∈ *Id*› **and**
st': ‹*narena* = (*arena'*, *aivdom*)› **and**
ae: ‹*length arena0* = *length arena*› **and**
le-all: ‹∀ *L* ∈# $\mathcal{L}_{all}$ 𝒜. *length* (*W' L*) ≤ *length arena*›
shows ‹*isasat-GC-clauses-prog-single-wl arena narena W A*
≤ ⇓ (*isasat-GC-refl* 𝒜 *vdom0 arena0*)
(*cdcl-GC-clauses-prog-single-wl N W' A' N'*)›
(**is** ‹- ≤ ⇓ *?R* -›)
⟨*proof*⟩

definition *cdcl-GC-clauses-prog-wl2* **where**
‹*cdcl-GC-clauses-prog-wl2* = (λ*N0* 𝒜*0 WS*. *do* {
𝒜 ← *SPEC*(λ𝒜. *set-mset* 𝒜 = *set-mset* 𝒜*0*);
(-, (*N*, *N'*, *WS*)) ← *WHILE*$_T$$^{cdcl\text{-}GC\text{-}clauses\text{-}prog\text{-}wl\text{-}inv}$ 𝒜 *N0*
(λ(ℬ, -). ℬ ≠ {#})
(λ(ℬ, (*N*, *N'*, *WS*)). *do* {

ASSERT($\mathcal{B} \neq \{\#\}$);
A ← *SPEC* ($\lambda A.\ A \in\#\ \mathcal{B}$);
(*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* $\mathcal{B} \Longrightarrow$
cdcl-GC-clauses-prog-wl-inv $\mathcal{A}$ *N* ({#}, *x*) $\Longrightarrow$ *cdcl-GC-clauses-prog-wl-inv* $\mathcal{B}$ *N* ({#}, *x*)›
⟨*proof*⟩

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*⟩

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*⟩

lemma *length-watched-le''*:
assumes
xb-x'a: ‹(*x1a*, *x1*) ∈ *twl-st-heur-restart*› **and**
prop-inv: ‹*correct-watching'-nobin x1*›
shows ‹∀ *x2* ∈# $\mathcal{L}_{all}$ (*all-init-atms-st x1*). *length* (*watched-by x1 x2*) ≤ *length* (*get-clauses-wl-heur x1a*)›
⟨*proof*⟩

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*⟩

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*⟩

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** (N0, (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})›
⟨*proof*⟩

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*′)›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*⟩

lemma *watchlist-put-binaries-first*:
assumes ‹*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*)›
shows ‹*watchlist-put-binaries-first W* ≤ ⇓(⟨*Id*⟩*map-fun-rel* (D_0 *A*)) (*SPEC* (λ*W*. *correct-watching'''*
B (*M*, *N*, *D*, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, N_0, U_0, *Q*, *W*)))›
⟨*proof*⟩

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*⟩

lemma *GC-remap-dom-m-subset*:
‹*GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) ⟹ *dom-m old'* ⊆# *dom-m old*›

⟨*proof*⟩

lemma *rtranclp-GC-remap-dom-m-subset*:
‹*rtranclp GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) ⟹ *dom-m old'* ⊆# *dom-m old*›
⟨*proof*⟩

lemma *GC-remap-mapping-unchanged*:
‹*GC-remap* (*old*, *m*, *new*) (*old'*, *m'*, *new'*) ⟹ *C* ∈ *dom m* ⟹ *m' C* = *m C*›
⟨*proof*⟩

lemma *rtranclp-GC-remap-mapping-unchanged*:
‹*GC-remap*** (*old*, *m*, *new*) (*old'*, *m'*, *new'*) ⟹ *C* ∈ *dom m* ⟹ *m' C* = *m C*›
⟨*proof*⟩

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'*)›
⟨*proof*⟩

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'*)›
⟨*proof*⟩

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'*)›
⟨*proof*⟩

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'*)›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

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)›
⟨*proof*⟩

sepref-def *schedule-next-reduce-st-impl*
is ‹*uncurry* (*RETURN oo schedule-next-reduce-st*)›
:: ‹*word64-assn*^{k} *$*_a$ *isasat-bounded-assn*^{d} *$\to_a$ isasat-bounded-assn›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *reduceint-impl*
is ‹*uncurry0* (*RETURN reduceint*)›
:: ‹*unit-assn*^{k} *$\to_a$ word64-assn›
⟨*proof*⟩

lemma *schedule-next-reduction-stI2*:
‹*1* ≤ *a* ⟹ *0* < *a*› **for** *a* :: ‹*64 word*›
⟨*proof*⟩

lemma *schedule-next-reduction-stI3*: ‹*word-log2 n div 2* < *64*› **for** *n* :: ‹*64 word*›
⟨*proof*⟩

sepref-def *schedule-next-reduction-st-impl*
is ‹*RETURN o schedule-next-reduction-st*›
:: ‹*isasat-bounded-assn*^{d} *$\to_a$ isasat-bounded-assn›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*)›
⟨*proof*⟩

find-theorems *hn-refine PASS*
sepref-def *vmtf-array-nxt-score-code*
is ‹*RETURN o vmtf-array-nxt-score*›
:: ‹*vmtf-assn*^{k} *$\to_a$ uint64-nat-assn›
⟨*proof*⟩

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*))›
⟨*proof*⟩

sepref-def *current-vmtf-array-nxt-score-code*
is ‹*RETURN o current-vmtf-array-nxt-score*›
:: ‹*heuristic-bump-assn*^{k} *$\to_a$ uint64-nat-assn›

⟨*proof*⟩

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} $\rightarrow_a$ *uint32-nat-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)))
})›
⟨*proof*⟩

sepref-def *empty-Q-fast-code*
is ‹*empty-Q*›
:: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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} $\rightarrow_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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} $\to_a$ *vdom-fast-assn*›
⟨*proof*⟩

sepref-register *remove-deleted-clauses-from-avdom arena-status DELETED*
lemma *mark-to-delete-clauses-wl-D-heur-is-Some-iff*:
‹*D = Some C* $\longleftrightarrow$ *D* $\neq$ *None* $\wedge$ ((*the D*) = *C*)›
⟨*proof*⟩

sepref-def *mop-marked-as-used-impl*
is ‹*uncurry mop-marked-as-used*›
:: ‹*arena-fast-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $\to_a$ *unat-assn' TYPE*(*2*)›
⟨*proof*⟩

sepref-def *MINIMUM-DELETION-LBD-impl*
is ‹*uncurry0* (*RETURN MINIMUM-DELETION-LBD*)›
:: ‹*unit-assn*^{k} $\to_a$ *uint32-nat-assn*›
⟨*proof*⟩

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} $\to_a$ *bool1-assn*›
⟨*proof*⟩

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*) $\leq$ *snat64-max*]$_a$
trail-pol-fast-assn^k $*_a$ *arena-fast-assn*^{d} $*_a$ *aivdom-assn*^{d} $\to$
arena-fast-assn $\times_a$ *aivdom-assn*›
⟨*proof*⟩

definition *lbd-sort-clauses* :: ‹*arena* $\Rightarrow$ *aivdom2* $\Rightarrow$ *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*) $\leq$ *snat64-max*]$_a$ *arena-fast-assn*^{k} $*_a$ *aivdom-int-assn*^{d} $\to$ *aivdom-int-assn*›
⟨*proof*⟩

lemma
map-vdom-aivdom-int2:
‹(*uncurry* (λ*arena. map-vdom-aivdom-int* (*f arena*)), *uncurry* (λ*arena. map-vdom-aivdom* (*f arena*)))
$\in$ *Id* $\times_r$ *aivdom-rel* $\to_f$ ⟨*aivdom-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma *get-aivdom-eq-aivdom-iff*:
‹*IsaSAT-VDom.get-aivdom b* = (*x1*, *a*, *aa*, *ba*) ⟷ *b* = *AIvdom* (*x1*, *a*, *aa*, *ba*)›
⟨*proof*⟩

lemma *quicksort-clauses-by-score-sort*:
‹(*uncurry lbd-sort-clauses*, *uncurry sort-clauses-by-score*) ∈
Id ×$_r$ *aivdom-rel* →$_f$ ⟨*aivdom-rel*⟩*nres-rel*›
⟨*proof*⟩

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*⟩

end

lemma *sort-vdom-heur-alt-def*:
‹*sort-vdom-heur* = (λ*S*$_0$. *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*)))
})›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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* (N_0 ∝ *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*, N_0, {#}, *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* = (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 ∧ 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*(λ*b*::*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*)}
}›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*))›
⟨*proof*⟩

sepref-def *set-conflict-to-false-code*
is ‹*RETURN o set-conflict-to-false*›
:: ‹*conflict-option-rel-assn*^{d} →$_a$ *conflict-option-rel-assn*›
⟨*proof*⟩

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
}
})›
⟨proof⟩

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›
⟨proof⟩

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›
⟨proof⟩

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'›
⟨proof⟩

definition simplify-clause-with-unit2-rel **where**
‹simplify-clause-with-unit2-rel N_0 C=
{((unc, N, L, b, i), (unc', b', N')).
(C ∈# dom-m N ⟶ N' = N) ∧
(C ∉# dom-m N ⟶ fmdrop C N' = N) ∧
(¬b ⟶ length (N' ∝ C) = 1 ⟶ C ∉# dom-m N ∧ N' ∝ C = [L]) ∧

(*if* $b \vee i \leq 1$ *then* $C \notin\# \ dom\text{-}m\ N$ *else* $C \in\# dom\text{-}m\ N) \wedge$
$(b{=}b') \wedge$
$(\neg b \longrightarrow i = size\ (N' \propto C)) \wedge$
$C \in\#\ dom\text{-}m\ N' \wedge$
$(b \vee i \leq 1 \longrightarrow size\ (learned\text{-}clss\text{-}lf\ N) = size\ (learned\text{-}clss\text{-}lf\ N_0) - (if\ irred\ N_0\ C\ then\ 0\ else\ 1)) \wedge$
$(\neg(b \vee i \leq 1) \longrightarrow size\ (learned\text{-}clss\text{-}lf\ N) = size\ (learned\text{-}clss\text{-}lf\ N_0)) \wedge$
$(C \in\#\ dom\text{-}m\ N \longrightarrow dom\text{-}m\ N = dom\text{-}m\ N_0) \wedge$
$(C \in\#\ dom\text{-}m\ N \longrightarrow irred\ N\ C = irred\ N_0\ C \wedge irred\ N\ C = irred\ N'\ C) \wedge$
$(C \notin\#\ dom\text{-}m\ N \longrightarrow dom\text{-}m\ N = remove1\text{-}mset\ C\ (dom\text{-}m\ N_0)) \wedge$
$unc{=}unc' \wedge$
simplify-clause-with-unit2-rel-simp-wo unc $N\ N_0\ N'\rangle$

lemma *simplify-clause-with-unit2-simplify-clause-with-unit*:
fixes $N\ N_0 :: \langle 'v\ clauses\text{-}l\rangle$ **and** $N' :: \langle 'a\rangle$
assumes $\langle C \in\#\ dom\text{-}m\ N\rangle$ $\langle no\text{-}dup\ M\rangle$ **and**
st: $\langle(M,M') \in Id\rangle$ $\langle(C,C') \in Id\rangle$ $\langle(N,N_0) \in Id\rangle$
shows
$\langle simplify\text{-}clause\text{-}with\text{-}unit2\ C\ M\ N \leq \Downarrow\ (simplify\text{-}clause\text{-}with\text{-}unit2\text{-}rel\ N_0\ C)$
$(simplify\text{-}clause\text{-}with\text{-}unit\ C'\ M'\ N_0)\rangle$
⟨*proof*⟩

lemma *all-learned-all-lits-all-atms-st*:
$\langle set\text{-}mset\ (all\text{-}learned\text{-}lits\text{-}of\text{-}wl\ T) = set\text{-}mset\ (all\text{-}learned\text{-}lits\text{-}of\text{-}wl\ S) \Longrightarrow$
$set\text{-}mset\ (all\text{-}init\text{-}lits\text{-}of\text{-}wl\ T) = set\text{-}mset\ (all\text{-}init\text{-}lits\text{-}of\text{-}wl\ S) \Longrightarrow$
$set\text{-}mset\ (all\text{-}atms\text{-}st\ T) = set\text{-}mset\ (all\text{-}atms\text{-}st\ S)\rangle$
⟨*proof*⟩

lemma *simplify-clause-with-unit-st2-simplify-clause-with-unit-st*:
fixes $S :: \langle nat\ twl\text{-}st\text{-}wl\rangle$
assumes $\langle(C,C') \in Id\rangle$ $\langle(S,S') \in Id\rangle$
shows
$\langle simplify\text{-}clause\text{-}with\text{-}unit\text{-}st2\ C\ S \leq \Downarrow Id\ (simplify\text{-}clause\text{-}with\text{-}unit\text{-}st\text{-}wl\ C'\ S')\rangle$
⟨*proof*⟩

lemma *simplify-clauses-with-unit-st2-simplify-clauses-with-unit-st*:
assumes $\langle(S,S') \in Id\rangle$
shows
$\langle simplify\text{-}clauses\text{-}with\text{-}unit\text{-}st2\ S \leq \Downarrow Id\ (simplify\text{-}clauses\text{-}with\text{-}unit\text{-}st\text{-}wl\ S')\rangle$
⟨*proof*⟩

lemma *simplify-clause-with-unit2-alt-def*:
$\langle simplify\text{-}clause\text{-}with\text{-}unit2\ C\ M\ N_0 = do\ \{$
ASSERT$(C \in\#\ dom\text{-}m\ N_0)$;
let $l = length\ (N_0 \propto C)$;
$(i,\ j,\ N,\ del,\ is\text{-}true) \leftarrow WHILE_T{}^{\lambda(i,\ j,\ N,\ del,\ b).\ C \in\#\ dom\text{-}m\ N}$
$(\lambda(i,\ j,\ N,\ del,\ b).\ \neg b \wedge j < l)$
$(\lambda(i,\ j,\ N,\ del,\ is\text{-}true).\ do\ \{$
ASSERT$(i < length\ (N \propto C) \wedge j < length\ (N \propto C) \wedge C \in\#\ dom\text{-}m\ N \wedge i \leq j)$;
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* $(i,\ j{+}1,\ N,\ add\text{-}mset\ L\ del,\ True)$
else if $val = Some\ False$
then *RETURN* $(i,\ j{+}1,\ N,\ add\text{-}mset\ L\ del,\ False)$

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 $(C \in\# \textit{dom-m}\ N \wedge i \leq \textit{length}\ (N \propto C))$;
ASSERT $(\textit{is-true} \vee j = \textit{length}\ (N_0 \propto C))$;
let $L = N \propto C\ !\ 0$;
if is-true $\vee\ i \leq 1$
then RETURN $(False,\ \textit{fmdrop}\ C\ N,\ L,\ \textit{is-true},\ i)$
else if $i{=}j \wedge \neg$ *is-true then RETURN* $(True,\ N,\ L,\ \textit{is-true},\ i)$
else do {
let $N = N(C \hookrightarrow (\textit{take}\ i\ (N \propto C)))$ *in RETURN* $(False,\ N,\ L,\ \textit{is-true},\ i)$}
}›
⟨*proof*⟩

lemma *normalize-down-return-spec*: ‹$\Downarrow A\ ((RETURN\ o\ f)\ c) = SPEC\ (\lambda a.\ (a,\ f\ c) \in \{(a,b).\ (a,b) \in A \wedge\ b = f\ c\})$›
⟨*proof*⟩

lemma *arena-length-le-length-arena*:
‹$C' \in\# \textit{dom-m}\ N \Longrightarrow$
valid-arena arena N vdom $\Longrightarrow$
arena-length arena $C' <$ *length arena*›
⟨*proof*⟩

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*) $\Longrightarrow$ *length* (*get-clauses-wl S* $\propto$ *C*) $\leq$ *Suc* (*unat32-max div 2*)›
⟨*proof*⟩

lemma *isa-simplify-clause-with-unit2-isa-simplify-clause-with-unit*:
assumes ‹*valid-arena arena N vdom*› **and**
trail: ‹$(M,\ M') \in$ *trail-pol* $\mathcal{A}$› **and**
lits: ‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* $\mathcal{A}$ (*mset* ‘# *ran-mf N*)› **and**
C: ‹$(C,C') \in Id$› **and**
le: ‹*length* $(N \propto C) \leq$ *Suc* (*unat32-max div 2*)›
shows ‹*isa-simplify-clause-with-unit2 C M arena* $\leq \Downarrow$
(*bool-rel* $\times_r$ $\{(arena',\ N).$ *valid-arena arena' N vdom* $\wedge$ *length arena'* = *length arena*$\}$ $\times_r$
Id $\times_r$ *bool-rel* $\times_r$ *nat-rel*)
(*simplify-clause-with-unit2 C' M' N*)›
⟨*proof*⟩

lemma *literals-are-in-mm-clauses*:
‹*literals-are-in-*$\mathcal{L}_{in}$*-mm* (*all-atms-st T*) (*mset* ‘# *ran-mf* (*get-clauses-wl T*))›
⟨*proof*⟩

lemma *mop-arena-status*:
assumes ‹$C \in\# \textit{dom-m}\ N$› **and** ‹$(C,C') \in$*nat-rel*›
‹*valid-arena arena N vdom*›
shows

‹mop-arena-status arena C
≤ SPEC
(λc. (c, irred N C′)
∈ {(a,b). (a = IRRED ⟷ b) ∧ (a = LEARNED ⟷ ¬b) ∧ (irred N C′ = b)})›
⟨proof⟩

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 𝒜 = *all-init-atms-st* (M, N, D, *NE*, *UE*, *NEk*, *UEk*, *NS*, *US*, *N0*, *U0*, Q, W) *in*
(M′, M) ∈ *trail-pol* 𝒜 ∧
valid-arena N′ N (*set* (*get-vdom-aivdom vdom*)) ∧
(D′, D) ∈ *option-lookup-clause-rel* 𝒜 ∧
(D = *None* ⟶ j ≤ *length* M) ∧
Q = *uminus* ‘# *lit-of* ‘# *mset* (*drop* j (*rev* M)) ∧
(W′, W) ∈ ⟨*Id*⟩*map-fun-rel* (D₀ 𝒜) ∧
vm ∈ *bump-heur* 𝒜 M ∧
no-dup M ∧
clvls ∈ *counts-maximum-level* M D ∧
cach-refinement-empty 𝒜 *cach* ∧
out-learned M D *outl* ∧
clss-size-corr-restart N *NE* {#} *NEk UEk NS* {#} *N0* {#} *lcount* ∧
vdom-m 𝒜 W N ⊆ *set* (*get-vdom-aivdom vdom*) ∧
aivdom-inv-dec vdom (*dom-m* N) ∧
isasat-input-bounded 𝒜 ∧
isasat-input-nempty 𝒜 ∧
old-arena = [] ∧
heuristic-rel 𝒜 *heur* ∧
(*occs*, *empty-occs-list* 𝒜) ∈ *occurrence-list-ref*
}›
⟨proof⟩

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›
⟨*proof*⟩

lemma *aivdom-inv-mono*:
‹B ⊆# A ⟹ *aivdom-inv* (v, *x1y*, *x2aa*, *tvdom*) A ⟹ *aivdom-inv* (v, *x1y*, *x2aa*, *tvdom*) B›
⟨*proof*⟩

lemma *aivdom-inv-dec-mono*:
‹B ⊆# A ⟹ *aivdom-inv-dec vdom* A ⟹ *aivdom-inv-dec vdom* B›

⟨*proof*⟩

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* ∧ *L* ∈# $\mathcal{L}_{all}$ (*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 $T = (M,\ N,\ D,\ NE,\ UE,\ NEk,\ UEk,\ (\textit{if } irr \textit{ then } add\text{-}mset\ E \textit{ else } id)\ NS,\ (\textit{if } \neg irr \textit{ then } add\text{-}mset\ E \textit{ else } id)\ US,\ N0,\ U0,\ Q,\ W)$;
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
}
})⟩
⟨*proof*⟩

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 $a \leq w$ ∧ *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 $a \leq w$ ∧ *get-vmtf-heur* a = vm ∧
learned-clss-count $a \leq$ *learned-clss-count* S ∧
length (*get-watched-wl-heur* a) = lw} (*simplify-clause-with-unit-st2* C' S')› (**is** ‹- ≤ ⇓?A -›)
⟨*proof*⟩

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›
⟨*proof*⟩

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*⟩

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)›
⟨*proof*⟩

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')›
⟨*proof*⟩

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*)
}›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

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*›

⟨*proof*⟩

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*⟩

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 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 (get-vmtf-heur-array S ! A), CS, S)
    })
    (Some (get-vmtf-heur-fst S₀), CS, S₀);
   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)›
  ⟨proof⟩
```

lemma *all-atms-st-add-kep*[*simp*]:
‹$L \in\# \mathcal{L}_{all}$ (*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))›
‹$L \in\# \mathcal{L}_{all}$ (*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))›
⟨*proof*⟩

lemma *clss-size-corr-in-dom-red-clss-size-lcount-ge0*:
‹$C \in\#$ *dom-m* $N \Longrightarrow \neg$*irred* N $C \Longrightarrow$ *clss-size-corr* N *NE UE NEk UEk NS US N0 U0 lcount* ⟹
clss-size-lcount lcount ≥ *Suc 0*›
⟨*proof*⟩

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* $S \leq u$ ∧ *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). $C \in\#$ *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*⟩

lemma [*simp*]: ‹(S, x) ∈ *state-wl-l None* ⟹
defined-lit (*get-trail-l* x) L ⟷ *defined-lit* (*get-trail-wl* S) L›
⟨*proof*⟩

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 ($L' \in\#$ $\mathcal{L}_{all}$ (*all-init-atms-st* (M, N, D, NE, UE, NEk, UEk, NS, US, $N0$, $U0$, Q, W)));
ASSERT ($L \in\#$ $\mathcal{L}_{all}$ (*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 ∧ $C \in\#$ *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* ($N \propto C$)) *else id*) NS, (*if irred* N C *then id else add-mset* (*mset* ($N \propto C$))) 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*⟩

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*⟩

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*)›
⟨*proof*⟩

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 (*all-init-atms-st T*)) = *set-mset* (*all-init-atms-st S*)))›
⟨*proof*⟩

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*›
⟨*proof*⟩

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\text{-}binary\text{-}clauses\text{-}inv\text{-}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⟩
```

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*⟩

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*⟩

lemma *lambda-split-second*: ‹(λ(*a*, *x*). *f a x*) = (λ(*a*,*b*,*c*:: *isasat*). *f a* (*b*,*c*))›
⟨*proof*⟩

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*

(λA (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
}›
⟨proof⟩

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 (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);
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
  }⟩
  ⟨proof⟩

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⟩

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⟩

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⟩


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⟩

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')⟩
  ⟨proof⟩
```

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-assn′ TYPE*(*64*))^k →$_a$ *ahm-assn*›
⟨*proof*⟩

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-assn*)^k ×$_a$ *unat-lit-assn*^{k} →$_a$ *bool1-assn*›
⟨*proof*⟩

sepref-def *ahm-get-marked-code*
is ‹*uncurry ahm-get-marked*›
:: ‹(*ahm-assn*)^k ×$_a$ *unat-lit-assn*^{k} →$_a$ *sint-assn′ TYPE*(*64*)›
⟨*proof*⟩

sepref-def *ahm-empty-code*
is ‹*ahm-empty*›
:: ‹(*ahm-assn*)^d →$_a$ *ahm-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *encoded-irred-index-irred-int-impl*
is ‹*RETURN o encoded-irred-index-irred-int*›
:: ‹(*sint-assn'* *TYPE*(*64*))^k →$_a$ *bool1-assn*›
⟨*proof*⟩

lemma *nat-sint-snat*: ‹*0* ≤ *sint xi* ⟹ (*nat* (*sint xi*) = *snat xi*)›
⟨*proof*⟩

lemma [*sepref-fr-rules*]:
‹(*Mreturn*, *RETURN o nat*) ∈ [λ*a*. *a* ≥ *0*]$_a$ (*sint-assn'* *TYPE*(*64*))^k → *sint64-nat-assn*›
⟨*proof*⟩
lemma [*sepref-fr-rules*]:
‹(*Mreturn o* (λ*x*. −*x*), *RETURN o uminus*) ∈ [λ*a*. *a* ≤ *int snat64-max* ∧ −*a* ≤ *int snat64-max*]$_a$
(*sint-assn'* *TYPE*(*64*))^k → (*sint-assn'* *TYPE*(*64*))›
⟨*proof*⟩

lemma *encoded-irred-index-get-int-alt-def*:
‹*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* (*0*−*a*))}›
⟨*proof*⟩
sepref-def *encoded-irred-index-irred-get-impl*
is ‹*encoded-irred-index-get-int*›
:: ‹(*sint-assn'* *TYPE*(*64*))^k →$_a$ *sint64-nat-assn*›
⟨*proof*⟩

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 ∧ *a* ≤ *snat64-max*]$_f$ *nat-rel* ×$_r$ *bool-rel* → ⟨*encoded-irred-indices*⟩*nres-rel*›
⟨*proof*⟩

lemma *int-snat-sint*: ‹¬ *sint xi* < *0* ⟹ *int* (*snat* (*xi*::*64 word*)) = *sint xi*›
⟨*proof*⟩

lemma [*sepref-fr-rules*]:
‹(*Mreturn*, *RETURN o int*) ∈ (*snat-assn'* *TYPE*(*64*))^k →$_a$ (*sint-assn'* *TYPE*(*64*))›
⟨*proof*⟩

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*))}›

⟨*proof*⟩

sepref-def *encoded-irred-index-set-int-impl*
is ‹*uncurry encoded-irred-index-set-int*›
:: ‹*sint64-nat-assn*^{k} $*_a$ *bool1-assn*^{k} $\rightarrow_a$ (*sint-assn'* *TYPE*(*64*))›
⟨*proof*⟩

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 $\rightarrow_a$ *ahm-assn*›
⟨*proof*⟩

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 $\rightarrow_a$ *ahm-assn*›
⟨*proof*⟩

definition *ahm-full-assn* :: ‹-› **where**
‹*ahm-full-assn* = *hr-comp* (*larray64-assn* (*sint-assn'* *TYPE*(*64*)) $\times_a$ *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*›
⟨*proof*⟩

lemma *ahm-set-marked-set-marked*:
‹(*uncurry2 ahm-set-marked*, *uncurry2 set-marked*)
∈ (*array-hash-map-rel encoded-irred-indices*) $\times_f$ *nat-lit-lit-rel* $\times_f$ *encoded-irred-indices* → ⟨*array-hash-map-rel encoded-irred-indices*⟩*nres-rel*›
⟨*proof*⟩

lemma *ahm-update-marked-update-marked*:
‹(*uncurry2 ahm-update-marked*, *uncurry2 update-marked*)
∈ (*array-hash-map-rel encoded-irred-indices*) $\times_f$ *nat-lit-lit-rel* $\times_f$ *encoded-irred-indices* → ⟨*array-hash-map-rel encoded-irred-indices*⟩*nres-rel*›
⟨*proof*⟩

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
})›
⟨*proof*⟩

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*›
⟨*proof*⟩

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
}›

⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-register *should-eliminate-pure-st*

sepref-def *should-eliminate-pure-st-impl*
is ‹*RETURN o should-eliminate-pure-st*›
:: ‹*isasat-bounded-assn*^{k} →$_a$ *bool1-assn*›
⟨*proof*⟩

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 ×$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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*)}›
⟨*proof*⟩

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* S_0), *CS*, S_0);

RETURN S
})›
⟨*proof*⟩

sepref-def *isa-deduplicate-binary-clauses-code*
is *isa-mark-duplicated-binary-clauses-as-garbage-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*›
⟨*proof*⟩

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* (*D*$_0$ (*all-init-atms-st S'*)) ∧
(*occs2*, *occs2'*) ∈ ⟨*bool-rel*⟩*map-fun-rel* (*D*$_0$ (*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* (*D*$_0$ (*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\text{-}pure\text{-}literal\text{-}count\text{-}occs\text{-}clause\text{-}wl\text{-}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 S₀ = (do {
   ASSERT (isa-pure-literal-deletion-wl-pre S₀);
   (eliminated, S) ← iterate-over-VMTF
    (λA (eliminated, T). do {
      ASSERT (get-vmtf-heur-array S₀ = 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 S₀ = get-vmtf-heur-array S);
         RETURN (eliminated + 1, S)}
        else RETURN (eliminated, T)
      })
     (λ(-, S). get-vmtf-heur-array S₀ = (get-vmtf-heur-array S))
     (get-vmtf-heur-array S₀, Some (get-vmtf-heur-fst S₀)) (0 :: 64 word, S₀);
   RETURN (eliminated, S)
})›

definition isa-pure-literal-deletion-wl :: ‹bool list ⇒ isasat ⇒ (64 word × isasat) nres› where
  ‹isa-pure-literal-deletion-wl occs S₀ = (do {
  ASSERT (isa-pure-literal-deletion-wl-pre S₀);
  (-, eliminated, S) ← WHILE_T^λ(n, -, S). get-vmtf-heur-array S₀ = 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 S₀));
      ASSERT (A ≤ unat32-max div 2);
      ASSERT (get-vmtf-heur-array S₀ = 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 S₀ = 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 S₀), 0, S₀);
  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') ∈ twl-st-heur-restart-ana' r u›
    ‹(occs, occs') ∈ ⟨bool-rel⟩map-fun-rel (D₀ (all-init-atms-st S'))›
    ‹(C,C')∈nat-rel›
  shows ‹isa-pure-literal-count-occs-clause-wl C S occs remaining ≤⇓{((occs, remaining), occs').
    (occs, occs') ∈ ⟨bool-rel⟩map-fun-rel (D₀ (all-init-atms-st S'))}
    (pure-literal-count-occs-clause-wl C' S' occs')›
⟨proof⟩
```

lemma *distinct-mset-add-subset-iff*: ‹*distinct-mset* $(A+B) \Longrightarrow A + B \subseteq\# C \longleftrightarrow A \subseteq\# C \wedge B \subseteq\# C$›
⟨*proof*⟩

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* $S \leq$
⇓ (*bool-rel* $\times_f$ ⟨*bool-rel*⟩*map-fun-rel* (D_0 (*all-init-atms-st* S')))
(*pure-literal-count-occs-wl* S')›
⟨*proof*⟩

lemma *lookup-clause-rel-cong*:
‹*set-mset* $\mathcal{A}$ = *set-mset* $\mathcal{B} \Longrightarrow L \in$ *lookup-clause-rel* $\mathcal{A} \Longrightarrow L \in$ *lookup-clause-rel* $\mathcal{B}$›
⟨*proof*⟩

lemma *isa-propagate-pure-bt-wl-propagate-pure-bt-wl*:
assumes S_0 T: ‹$(S_0, T) \in$ *twl-st-heur-restart-ana'* r u› **and**
‹$(L,L') \in Id$›
shows ‹*isa-propagate-pure-bt-wl* L $S_0 \leq$⇓{(U, V). $(U,V) \in$*twl-st-heur-restart-ana'* r $u \wedge$ *get-vmtf-heur* U = *get-vmtf-heur* S_0} (*propagate-pure-bt-wl* L' T)›
⟨*proof*⟩

lemma *isa-pure-literal-deletion-wl-pure-literal-deletion-wl*:
assumes S_0 T: ‹$(S_0, T) \in$ *twl-st-heur-restart-ana'* r u› **and**
occs: ‹$(occs, occs') \in$ ⟨*bool-rel*⟩*map-fun-rel* (D_0 (*all-init-atms-st* T))›
shows ‹*isa-pure-literal-deletion-wl* *occs* $S_0 \leq$⇓{((-, U), V). $(U,V) \in$*twl-st-heur-restart-ana'* r u} (*pure-literal-deletion-wl* *occs'* T)›
⟨*proof*⟩

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) $\leq$ *snat64-max* $\wedge$ *learned-clss-count* $S \leq$ *unat64-max*]$_a$
sint64-nat-assn^k $*_a$ *isasat-bounded-assn*^{k} $*_a$ (*larray-assn'* *TYPE*(64) *bool1-assn*)^d $*_a$ *word64-assn*^{d}
$\rightarrow$ *larray-assn'* *TYPE*(64) *bool1-assn* $\times_a$ *word64-assn*›
⟨*proof*⟩

sepref-register *isa-pure-literal-count-occs-wl*
sepref-def *isa-pure-literal-count-occs-wl-code*
is *isa-pure-literal-count-occs-wl*
:: ‹[λS. *length* (*get-clauses-wl-heur* S) $\leq$ *snat64-max* $\wedge$ *learned-clss-count* $S \leq$ *unat64-max*]$_a$
isasat-bounded-assn^k $\rightarrow$ *bool1-assn* $\times_a$ *larray-assn'* *TYPE*(64) *bool1-assn*›
⟨*proof*⟩

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})›
⟨*proof*⟩

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} *$\to_a$ isasat-bounded-assn*›
⟨*proof*⟩

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}$ (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);
mop-free occs;
RETURN (*eliminated*, *S*)
})›
⟨*proof*⟩

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*›
⟨*proof*⟩

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 (λL. 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 = (λS$_0$. 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-*$\mathcal{L}_{in}$' *S*›
shows ‹*subsume-clauses-match2 C C' S D* ≤ ⇓*Id* (*subsume-clauses-match E F N*)›
⟨*proof*⟩

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*)) ∧
(¬*changed* ⟶ *D'* = *D* ∧ *occs'* = *occs* ∧ *get-clauses-wl S* ∝ *C* = *get-clauses-wl* S_0 ∝ *C*) ∧
correct-occurence-list S occs' (*if changed then remove1-mset C cands else cands*)

(*if changed then size* (*get-clauses-wl* S_0 $\propto$ C) *else* n) $\wedge$
all-occurrences (*all-init-atms-st* S) $occs' \subseteq\#$ *add-mset* C (*all-occurrences* (*all-init-atms-st* S) $occs$)
$\wedge$
$(S,T) \in Id$
}›

lemma *literals-are-$\mathcal{L}_{in}$'-all-init-atms-alt-def*:
‹*literals-are-$\mathcal{L}_{in}$'* $S \Longrightarrow$
set-mset (*all-init-atms-st* S) = *set-mset* (*all-atms-st* S)›
⟨*proof*⟩

lemma
inter-mset-empty-remove1-msetI:
‹$A \cap\# B = \{\#\} \Longrightarrow A \cap\#$ *remove1-mset* c $B = \{\#\}$› **and**
inter-mset-empty-removeAll-msetI:
‹$A \cap\# B = \{\#\} \Longrightarrow A \cap\#$ *removeAll-mset* c $B = \{\#\}$›
⟨*proof*⟩

lemma *push-to-occs-list2*:
assumes *occs*: ‹*correct-occurence-list* S *occs cands* n›
‹$C \in\#$ *dom-m* (*get-clauses-wl* S)›
‹2 $\leq$ *length* (*get-clauses-wl* S $\propto$ C)› **and**
lin: ‹*literals-are-$\mathcal{L}_{in}$'* S› **and**
undef: ‹$\forall L \in$ *set* (*get-clauses-wl* S $\propto$ C). *undefined-lit* (*get-trail-wl* S) L› **and**
notin: ‹$C \notin\#$ *all-occurrences* (*mset-set* (*fst occs*)) *occs*›
shows ‹*push-to-occs-list2* C S *occs* $\leq$ *SPEC* (λc. (c, ()) $\in$ {(*occs'*, *occs''*).
all-occurrences (*all-init-atms-st* S) *occs'* = *add-mset* C (*all-occurrences* (*all-init-atms-st* S) *occs*) $\wedge$
correct-occurence-list S *occs'* (*remove1-mset* C *cands*) (*max* n (*length* (*get-clauses-wl* S $\propto$ C))) $\wedge$ *fst* *occs* = *fst occs''*})›
⟨*proof*⟩

lemma *maybe-push-to-occs-list2*:
assumes *occs*: ‹*correct-occurence-list* S *occs cands* n›
‹$C \in\#$ *dom-m* (*get-clauses-wl* S)›
‹2 $\leq$ *length* (*get-clauses-wl* S $\propto$ C)› **and**
lin: ‹*literals-are-$\mathcal{L}_{in}$'* S› **and**
undef: ‹$\forall L \in$ *set* (*get-clauses-wl* S $\propto$ C). *undefined-lit* (*get-trail-wl* S) L› **and**
notin: ‹$C \notin\#$ *all-occurrences* (*mset-set* (*fst occs*)) *occs*›
shows ‹*maybe-push-to-occs-list2* C S *occs* $\leq$ *SPEC* (λc. (c, ()) $\in$ {(*occs'*, *occs''*).
(*all-occurrences* (*all-init-atms-st* S) *occs'* = *add-mset* C (*all-occurrences* (*all-init-atms-st* S) *occs*) $\vee$
all-occurrences (*all-init-atms-st* S) *occs'* = *all-occurrences* (*all-init-atms-st* S) *occs*) $\wedge$
correct-occurence-list S *occs'* (*remove1-mset* C *cands*) (*max* n (*length* (*get-clauses-wl* S $\propto$ C))) $\wedge$ *fst* *occs* = *fst occs''*})›
⟨*proof*⟩

definition *forward-subsumption-one-wl2* :: ‹*nat* $\Rightarrow$ *nat multiset* $\Rightarrow$ *nat literal* $\Rightarrow$ *occurences* $\Rightarrow$ *clause-hash* $\Rightarrow$
nat twl-st-wl $\Rightarrow$ (*nat twl-st-wl* $\times$ *bool* $\times$ *occurences* $\times$ *clause-hash*) *nres*› **where**
‹*forward-subsumption-one-wl2* = (λC *cands* L *occs* D S. *do* {
ASSERT (*forward-subsumption-one-wl2-pre* C *cands* L S);
ASSERT (*atm-of* $L \in$ *fst occs*);
let ys = *occ-list occs* L;
let n = *length ys*;
(-, s) $\leftarrow$
$WHILE_T$ *forward-subsumption-one-wl2-inv* S C *cands ys* ($\lambda(i, s)$. $i < n \wedge s = NONE$)

```
    (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*›
⟨*proof*⟩

lemma *clause-hash-ref-cong*: ‹*set-mset A* = *set-mset B* ⟹ *clause-hash-ref A* = *clause-hash-ref B*› **for** *A B*
⟨*proof*⟩
lemma *remdups-mset-set-mset-cong*: ‹*set-mset A* = *set-mset B* ⟹ *remdups-mset A* = *remdups-mset B*› **for** *A B*
⟨*proof*⟩
lemma *all-occurrences-cong*: ‹*set-mset A* = *set-mset B* ⟹ *all-occurrences A* = *all-occurrences B*› **for** *A B*
⟨*proof*⟩

lemma *correct-occurence-list-mono-candsI*:
‹*correct-occurence-list Sa occs* (*cands*) *n* ⟹
correct-occurence-list Sa occs (*remove1-mset C cands*) *n*›
⟨*proof*⟩

lemma *correct-occurence-list-mono-candsI2*:
‹*correct-occurence-list Sa occs* (*add-mset C cands*) *n* ⟹
correct-occurence-list Sa occs (*cands*) *n*›
⟨*proof*⟩

lemma *correct-occurence-list-size-mono*:
‹*correct-occurence-list x1h occs cands n* ⟹ *m* ≥ *n* ⟹
correct-occurence-list x1h occs cands m›

⟨*proof*⟩

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*)›
⟨*proof*⟩

lemma *forward-subsumption-one-wl2-forward-subsumption-one-wl*:
fixes S_0 C
defines G: ‹$G \equiv$ *mset* (*get-clauses-wl* $S_0 \propto C$)›
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: ‹$n \leq$ *size* (*get-clauses-wl* $S_0 \propto C$)› **and**
C-occs: ‹$C \notin\#$ *all-occurrences* (*all-init-atms-st* S_0) *occs*› **and**
L: ‹*atm-of* $L \in\#$ *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 $S_0 \leq \Downarrow$
(*forward-subsumption-one-wl2-rel* S_0 *occs cands n* C D)
(*forward-subsumption-one-wl* E *cands*′ T_0)›
⟨*proof*⟩

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* $S \propto C$);
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* ⟶ $C \in\#$ *dom-m* (*get-clauses-wl* S));
D ← (*if* ¬*changed then mop-ch-remove-all* (*mset* (*get-clauses-wl* $S \propto C$)) 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* ∧
$n \leq$ *length* (*get-clauses-wl* $S_0 \propto C$) ∧
$C \notin\#$ *all-occurrences* (*all-init-atms-st* S_0) *occs* ∧
distinct-mset cands ∧
$C \in\#$ *dom-m* (*get-clauses-wl* S_0) ∧
G = *mset* (*get-clauses-wl* $S_0 \propto C$)›

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*⟩

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*)›
⟨*proof*⟩

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))
⟨*proof*⟩

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$_T^{\text{populate-occs-inv } S\ xs}$ *(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 (λcands'. mset cands' = mset cands ∧ sorted-wrt (λa 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*›
⟨*proof*⟩

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*⟩

definition *forward-subsumption-all-wl2* :: ‹*nat twl-st-wl ⇒ - nres*› **where**
‹*forward-subsumption-all-wl2 = (λS. 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*⟩

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*⟩
⟨*proof*⟩

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*⟩

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*⟩

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′)›
⟨proof⟩

lemma mop-arena-status2:
assumes ‹(C,C′)∈nat-rel› ‹C ∈ vdom›
‹valid-arena arena N vdom›
shows
‹mop-arena-status arena C
≤ SPEC
(λc. (c, C′ ∈# dom-m N)
∈ {(a,b). (b ⟶ (a = IRRED ⟷ irred N C) ∧ (a = LEARNED ⟷ ¬irred N C)) ∧ (a = DELETED ⟷ ¬b)})›
⟨proof⟩

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⟩

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)›
⟨proof⟩

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
⟨proof⟩

lemma mop-ch-remove-all-clauses-mop-ch-remove-all:
‹mop-ch-remove-all-clauses C D ≤ ⇓Id (mop-ch-remove-all C D)›
⟨proof⟩

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*⟩

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* (λ*L*. *L* ∈# *mset* (*get-clauses-wl S'* ∝ *C*))›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *in-cocc-content-iff*: ‹*C* ∈# *cocc-content occs* ⟷ (∃ *xs*. *xs* ∈ *set occs* ∧ *C* ∈ *set xs*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *set-mset-cocc-content-set*[*simp*]: ‹*set-mset* (*cocc-content occs*) = *cocc-content-set occs*›
⟨*proof*⟩

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*⟩

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*⟩

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'*)›
⟨*proof*⟩

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*))›
⟨*proof*⟩

lemma *vdom-m-promote-same*:
‹*C* ∈# *dom-m b* ⟹ *vdom-m A m* (*fmupd C* (*b* ∝ *C*, *True*) *b*) = *vdom-m A m* (*b*)›
⟨*proof*⟩

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-oc*
r u ∧ *get-occs U* = *get-occs S* ∧ *get-aivdom U* = *get-aivdom S*})›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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-wl 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-learned-clss-wl S*›
‹*IsaSAT-Setup.get-kept-unit-learned-clss-wl* (*remove-lit-from-clause-wl C L′ S*) = *IsaSAT-Setup.get-kept-unit-learned-clss-wl 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*›
⟨*proof*⟩

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*›
⟨*proof*⟩
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*)›

⟨*proof*⟩

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*)›
⟨*proof*⟩

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*⟩

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*))›
⟨*proof*⟩

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*⟩

lemma *count-list-distinct-If*: ‹*distinct xs* ⟹ *count-list xs x* = (*if x* ∈ *set xs then 1 else 0*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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-clss-wl S*›
‹*IsaSAT-Setup.get-kept-unit-learned-clss-wl* (*add-clause-to-subsumed b N S*) = *IsaSAT-Setup.get-kept-unit-learned-clss-wl 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-wl 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›
⟨proof⟩

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-wl 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-learned-clss-wl S›
‹IsaSAT-Setup.get-kept-unit-learned-clss-wl (add-clauses-to-subsumed-wl N S) = IsaSAT-Setup.get-kept-unit-learned-clss-wl 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›
⟨proof⟩

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 (λc. (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⟩

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⟩

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-l*
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*›
⟨*proof*⟩

lemma [*simp*]: ‹*get-occs* (*incr-wasted-st b S*) = *get-occs S*›
‹*learned-clss-count* (*incr-wasted-st b S*) = *learned-clss-count S*›
⟨*proof*⟩

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*⟩

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*)›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

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*⟩

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*) ∈ *Id*›
shows ‹*isa-subsume-or-strengthen-wl C x2a T*
≤ ⇓ {(*U*,*V*). (*U*,*V*)∈*twl-st-heur-restart-occs' r u* ∧ *get-occs U* = *get-occs T* ∧ *get-aivdom U* = *get-aivdom T*}
(*subsume-or-strengthen-wl C x2 S*)›
⟨*proof*⟩

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*⟩

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*) ∈ *occurrence-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*⟩

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*›
⟨*proof*⟩

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'*)›
⟨*proof*⟩

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*⟩

lemma *valid-occs-empty*:
assumes ‹(*occs*, *empty-occs-list A*) ∈ *occurrence-list-ref*›
shows ‹*valid-occs occs vdom*› **and** ‹*cocc-content-set occs* = {}›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*⟩

lemma *valid-occs-push-to-tvdom*[*intro*!]:
‹*valid-occs occs aivdom* ⟹ *valid-occs occs* (*push-to-tvdom C aivdom*)›
⟨*proof*⟩

lemma *valid-occs-empty-vdom*[*intro*!]:
‹*valid-occs occs vdom* ⟹ *valid-occs occs* (*empty-tvdom vdom*)›
⟨*proof*⟩

lemma *valid-arena-vdom-le-strong*:
assumes ‹*valid-arena arena N ovdm*›
shows ‹*card ovdm* ≤ *length arena* − *2*›
⟨*proof*⟩

lemma
assumes *SS'*: ‹(*S*, *S'*) ∈ *twl-st-heur-restart-occs' r u*›
shows
twl-st-heur-restart-occs'-avdom-vdom: ‹*x1* < *length* (*get-avdom S*) ⟹ *get-avdom S* ! *x1* ∈ *set* (*get-vdom S*)› **and**
twl-st-heur-restart-occs'-ivdom-vdom: ‹*x1* < *length* (*get-ivdom S*) ⟹ *get-ivdom S* ! *x1* ∈ *set* (*get-vdom S*)› **and**

twl-st-heur-restart-occs'-length-vdom-clauses: ‹*length* (*get-vdom S*) ≤ *length* (*get-clauses-wl-heur S*)›
and
twl-st-heur-restart-occs'-length-tvdom-clauses: ‹*length* (*get-tvdom S*) ≤ *length* (*get-clauses-wl-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*⟩

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*⟩

lemma *empty-occs2-replicate-empty*: ‹*empty-occs2 occs* ≤ *SPEC* (λ*c*. (*c*, *replicate* (*length occs*) []) ∈ *Id*)›
⟨*proof*⟩

lemma *empty-occs2-st-empty-occs-st*: ‹*empty-occs2-st S* ≤ ⇓*Id* (*empty-occs-st S*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

lemma [*intro*!]: ‹*heuristic-rel A heur* ⟹ *heuristic-rel A* (*reset-added-heur heur*)›
⟨*proof*⟩

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*⟩
lemma *mark-added-clause-heur2-id*:

assumes ‹$(S,T) \in$ *twl-st-heur-restart-occs'* $r\ u$› **and** ‹$C \in\#$ *dom-m* (*get-clauses-wl* T)› **and**
lits: ‹*literals-are-*$\mathcal{L}_{in}$*'* T›
shows ‹*mark-added-clause-heur2* $S\ C$
$\leq \Downarrow\{(U,\ V).\ (U,\ V) \in$ *twl-st-heur-restart-occs'* $r\ u \wedge$ (*get-clauses-wl-heur* U) = *get-clauses-wl-heur* $S \wedge$
learned-clss-count U = *learned-clss-count* $S \wedge$ *get-vdom* U = *get-vdom* $S \wedge$
get-occs U = *get-occs* $S\}$ (*RETURN* T)› (**is** ‹- $\leq\Downarrow ?R$ -›)
⟨*proof*⟩

lemma *isa-forward-reset-added-and-stats*:
‹$(S,\ S') \in$ *twl-st-heur-restart-occs'* $r\ u \Longrightarrow$
(*isa-forward-reset-added-and-stats* $S,\ S') \in$ *twl-st-heur-restart-occs'* $r\ u$›
⟨*proof*⟩

lemma [*intro*!]: ‹*heuristic-rel* $\mathcal{A}$ (*get-heur* S) $\Longrightarrow$ *heuristic-rel* $\mathcal{A}$ (*schedule-next-subsume delta* (*get-heur* S))›
⟨*proof*⟩

lemma *schedule-next-subsume-st*:
‹$(S,\ S') \in$ *twl-st-heur-restart-occs'* $r\ u \Longrightarrow$
(*schedule-next-subsume-st delta* $S,\ S') \in$ *twl-st-heur-restart-occs'* $r\ u$›
⟨*proof*⟩

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›
⟨*proof*⟩

lemma *forward-subsumption-finalize*:
assumes
‹$(S,\ S') \in$ *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: ‹$\forall\ C \in\#$ *mset shrunken*. $C \in$ *set* (*get-vdom* S)› **and**
lits: ‹*literals-are-*$\mathcal{L}_{in}$*'* S'›
shows ‹*forward-subsumption-finalize shrunken* $S \leq SPEC(\lambda c.\ (c,S') \in$ *twl-st-heur-restart-ana'* $r\ u$)› (**is** ‹- $\leq$ *?C*›)
⟨*proof*⟩

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')›
⟨*proof*⟩

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 $\rightarrow_f$ ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

lemma *mop-cch-add-all-clause-mop-ch-add-all*:
assumes ‹$(S,\ S') \in$ *twl-st-heur-restart-occs'* $r\ u$› ‹$(C,C') \in$*nat-rel*›
‹$(D,\ D') \in$*clause-hash*› **and** ‹$C' \in\#$ *dom-m* (*get-clauses-wl* S')›
shows ‹*mop-cch-add-all-clause* $S\ C\ D \leq \Downarrow$*clause-hash* (*mop-ch-add-all* (*mset* (*get-clauses-wl* $S' \propto C'$)) D')›

⟨*proof*⟩

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*›
⟨*proof*⟩

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*⟩

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'*)›
⟨*proof*⟩

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*⟩

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*))›
⟨*proof*⟩

sepref-def *incr-forward-strengthened-st-impl*
is ‹*RETURN o incr-forward-strengthened-st*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

sepref-def *incr-forward-tried-st-impl*
is ‹*RETURN o incr-forward-tried-st*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

sepref-def *incr-forward-rounds-st-impl*
is ‹*RETURN o incr-forward-rounds-st*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

sepref-def *incr-forward-subsumed-st-impl*
is ‹*RETURN o incr-forward-subsumed-st*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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} →$_a$ *vdom-fast-assn*›
⟨*proof*⟩

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} $\rightarrow_a$ *aivdom-int-assn*›
⟨*proof*⟩

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* $\rightarrow_f$ ⟨*aivdom-rel*⟩*nres-rel*›
⟨*proof*⟩

lemma *get-aivdom-eq-aivdom-iff*:
‹*IsaSAT-VDom.get-aivdom b* = (*x1*, *a*, *aa*, *ba*) ⟷ *b* = *AIvdom* (*x1*, *a*, *aa*, *ba*)›
⟨*proof*⟩
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 $\times_r$ *aivdom-rel* $\rightarrow_f$ ⟨*aivdom-rel*⟩*nres-rel*›
⟨*proof*⟩

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} $\rightarrow_a$ *aivdom-assn*›
(**is** ‹*?c* ∈ [*?pre*]$_a$ *?im* → *?f*›)
⟨*proof*⟩

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
}›

⟨*proof*⟩

sepref-def *sort-cands-by-length-impl*
is *sort-cands-by-length*
:: ‹$isasat\text{-}bounded\text{-}assn^d \to_a isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

sepref-register *mop-is-marked-added-heur-st*

sepref-def *isa-all-lit-clause-unset-impl*
is ‹*uncurry isa-all-lit-clause-unset*›
:: ‹$sint64\text{-}nat\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^k \to_a bool1\text{-}assn$›
⟨*proof*⟩

lemma *rdomp-aivdom-assn-length-avdomD*: ‹*rdomp aivdom-assn x* ⟹ *length* (*get-avdom-aivdom x*) < *max-snat 64*›
⟨*proof*⟩

lemma *rdomp-isasat-bounded-assn-length-avdomD*:
‹*rdomp isasat-bounded-assn x* ⟹ *length-avdom x* < *max-snat 64*›
⟨*proof*⟩

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*›
:: ‹$sint64\text{-}nat\text{-}assn^k *_a isasat\text{-}bounded\text{-}assn^k \to_a unat\text{-}lit\text{-}assn$›
⟨*proof*⟩

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*)
}›
⟨*proof*⟩

sepref-register *mop-cocc-list-append*
sepref-def *mop-cocc-list-append-impl*
is ‹*uncurry2 mop-cocc-list-append*›
:: ‹$[\lambda((C,occs), L).\ Suc\ (length\ (occs\ !\ nat\text{-}of\text{-}lit\ L)) < max\text{-}snat\ 64]_a$
$sint64\text{-}nat\text{-}assn^k *_a occs\text{-}assn^d *_a unat\text{-}lit\text{-}assn^k \to occs\text{-}assn$›
⟨*proof*⟩

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*)
}›
⟨*proof*⟩

sepref-def *empty-tvdom-st-impl*
  is *empty-tvdom-st*
  :: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

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*
  }›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

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} →$_a$ *bool1-assn*›
  ⟨*proof*⟩

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*)
  }›
  ⟨*proof*⟩

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*›

⟨*proof*⟩

lemma *isa-populate-occs-inv-isasat-fast-relaxedI*:
‹*isa-populate-occs-inv x cands* (*a1'*, *a2'*) ⟹ *isasat-fast-relaxed x* ⟹ *isasat-fast-relaxed a2'*›
⟨*proof*⟩

sepref-def *isa-populate-occs-code*
is *isa-populate-occs*
:: ‹[*isasat-fast-relaxed*]$_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
⟨*proof*⟩

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} →$_a$ *cch-assn*›
⟨*proof*⟩

sepref-def *mop-cch-add-impl*
is ‹*uncurry mop-cch-add*›
:: ‹*unat-lit-assn*^{k} *$_a$ *cch-assn*^{d} →$_a$ *cch-assn*›
⟨*proof*⟩

sepref-def *mop-cch-in-impl*
is ‹*uncurry mop-cch-in*›
:: ‹*unat-lit-assn*^{k} *$_a$ *cch-assn*^{k} →$_a$ *bool1-assn*›
⟨*proof*⟩

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} →$_a$ *cch-assn*›
⟨*proof*⟩

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} →$_a$ *bool1-assn*›
⟨*proof*⟩

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}›
⟨proof⟩

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›
‹(λC. (1, 0, C), SUBSUMED-BY) ∈ R1 → ⟨R1,R2,R3⟩subsumption-rel›
‹(λC 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›
⟨proof⟩

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›*
⟨proof⟩

sepref-definition *is-subsumed-impl*
is *‹RETURN o (λ(tag, -). tag = 1)›*
:: *‹subsumption-raw-assn^k →$_a$ bool1-assn›*
⟨proof⟩

sepref-definition *is-strengthened-impl*
is *‹RETURN o (λ(tag, -). tag = 2)›*
:: *‹subsumption-raw-assn^k →$_a$ bool1-assn›*
⟨proof⟩

sepref-definition *STRENGTHENED-BY-impl*
is *‹uncurry (RETURN oo (λC D. (2, C, D)))›*
:: *‹word-assn^k *$_a$ id-assn^k →$_a$ subsumption-raw-assn›*
⟨proof⟩

sepref-definition *SUBSUMED-BY-impl*
is *‹RETURN o (λC. (1, 0, C))›*
:: *‹word-assn^k →$_a$ subsumption-raw-assn›*
⟨proof⟩

sepref-definition *NONE-impl*
is ‹*uncurry0* (*RETURN* (*0*, *0*, *0*::*64 word*))›
:: ‹*unit-assn*^{k} →$_a$ *subsumption-raw-assn*›
⟨*proof*⟩

sepref-definition *subsumed-by-impl*
is ‹*RETURN o* (λ(*tag*, *C*, *D*). *D*)›
:: ‹*subsumption-raw-assn*^{k} →$_a$ *id-assn*›
⟨*proof*⟩

sepref-definition *strengthened-on-lit-impl*
is ‹*RETURN o* (λ(*tag*, *C*, *D*). *C*)›
:: ‹*subsumption-raw-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

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* ⟷ *is-NONE x*› ‹*NONE* = *x* ⟷ *is-NONE x*›
⟨*proof*⟩

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
}›
⟨*proof*⟩

schematic-goal *mk-free-lbd-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE subsumption-assn ?fr*›
⟨*proof*⟩

lemma [*safe-constraint-rules*]: ‹*CONSTRAINT is-pure subsumption-assn*›
⟨*proof*⟩

sepref-register *isa-subsume-clauses-match2*
sepref-def *isa-subsume-clauses-match2-impl*
is ‹*uncurry3 isa-subsume-clauses-match2*›
:: ‹*sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{k} $*_a$ *cch-assn*^{k} $\rightarrow_a$ *subsumption-assn*›
⟨*proof*⟩

sepref-def *mop-cch-remove-one-impl*
is ‹*uncurry mop-cch-remove-one*›
:: ‹*unat-lit-assn*^{k} $*_a$ *cch-assn*^{d} $\rightarrow_a$ *cch-assn*›
⟨*proof*⟩

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-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *arena-fast-assn*^{d} $\rightarrow_a$ *arena-fast-assn*›
⟨*proof*⟩

sepref-def *mop-cch-remove-all-clauses-impl*
is ‹*uncurry2 mop-cch-remove-all-clauses*›
:: ‹*isasat-bounded-assn*^{k} $*_a$ *sint64-nat-assn*^{k} $*_a$ *cch-assn*^{d} $\rightarrow_a$ *cch-assn*›
⟨*proof*⟩

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*⟩

lemma *arena-is-valid-clause-idxD2*: ‹*arena-is-valid-clause-idx b a* $\Longrightarrow$ $a -$ *Suc 0* $<$ *length b*›
‹*arena-is-valid-clause-idx b a* $\Longrightarrow$ *MAX-LENGTH-SHORT-CLAUSE* $<$ *arena-length b a* $\Longrightarrow$ $3 \leq a$›
‹*arena-is-valid-clause-idx b a* $\Longrightarrow$ *MAX-LENGTH-SHORT-CLAUSE* $<$ *arena-length b a* $\Longrightarrow$ $a - 3 <$ *length b*›
⟨*proof*⟩

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} $\rightarrow_a$ *arena-fast-assn*›
⟨*proof*⟩

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} $\rightarrow_a$ *arena-fast-assn*›
⟨*proof*⟩

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*)
}›
⟨*proof*⟩

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} $\rightarrow_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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
})›
⟨*proof*⟩

sepref-def *mark-garbage-heur-as-subsumed-impl*
is ‹*uncurry mark-garbage-heur-as-subsumed*›
:: ‹*sint64-nat-assn*^{k} $*_a$ *isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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} $\rightarrow_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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*)})›
⟨*proof*⟩

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} $\rightarrow_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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* $\times_a$ *subsumption-assn* $\times_a$ *cch-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *isasat-bounded-assn-get-vdomD*: ‹*rdomp isasat-bounded-assn a* ⟹ *length* (*get-tvdom a*) < *max-snat 64*›
⟨*proof*⟩

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 $\times_a$ *al-assn′ TYPE*(*64*) *sint64-nat-assn* $\times_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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*)
}›
⟨*proof*⟩

sepref-def *empty-occs2-impl*
is ‹*empty-occs2*›
:: ‹*occs-assn*^{d} $\rightarrow_a$ *occs-assn*›
⟨*proof*⟩

sepref-def *empty-occs2-st-impl*
is ‹*empty-occs2-st*›
:: ‹*isasat-bounded-assn*^{d} $\rightarrow_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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)›
⟨*proof*⟩

sepref-def *schedule-next-subsume-st*
is ‹*uncurry* (*RETURN oo schedule-next-subsume-st*)›
:: ‹*word64-assn*^{k} *∗*$_a$ *isasat-bounded-assn*^{d} *→*$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

sepref-def *forward-subsumption-finalize*
is ‹*uncurry forward-subsumption-finalize*›
:: ‹(*al-assn*′ *TYPE*(*64*) *sint64-nat-assn*)^k *∗*$_a$ *isasat-bounded-assn*^{d} *→*$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

sepref-def *isa-forward-subsumption-all-impl*
is *isa-forward-subsumption-all*
:: ‹[*isasat-fast-relaxed*]$_a$ *isasat-bounded-assn*^{d} *→ isasat-bounded-assn*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

sepref-def *get-subsumption-opts-impl*
is ‹*RETURN o get-subsumption-opts*›
:: ‹*isasat-bounded-assn*^{k} *→*$_a$ *bool1-assn*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

sepref-def *next-subsume-schedule-st-impl*
is ‹*RETURN o next-subsume-schedule-st*›
:: ‹*isasat-bounded-assn*^{k} *→*$_a$ *word-assn*›
⟨*proof*⟩

sepref-def *should-subsume-st*
is ‹*RETURN o should-subsume-st*›
:: ‹*isasat-bounded-assn*^{k} *→*$_a$ *bool1-assn*›

⟨*proof*⟩

sepref-def *isa-forward-subsume-impl*
is *isa-forward-subsume*
:: ‹$[$*isasat-fast-relaxed*$]_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
⟨*proof*⟩

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 ← *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 ≤ *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')›
  ⟨proof⟩


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›
  ⟨proof⟩

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 S₀ ≤⇓{((-, U), V). (U,V)∈twl-st-heur-restart-ana' r u}
(pure-literal-elimination-round-wl T)›
⟨proof⟩


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'›
  ⟨proof⟩

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*⟩

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*⟩

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*›
⟨*proof*⟩

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›
⟨*proof*⟩

sepref-def *isa-pure-literal-elimination-round-wl-code*
is *isa-pure-literal-elimination-round-wl*
:: ‹[λS. *length* (*get-clauses-wl-heur* S) ≤ *snat64-max* ∧ *learned-clss-count* S ≤ *unat64-max*]$_a$
isasat-bounded-assn^d → *word64-assn* ×$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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)›
⟨*proof*⟩

sepref-def *schedule-next-pure-lits-st-impl*
is ‹*RETURN o schedule-next-pure-lits-st*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

sepref-def *isa-pure-literal-elimination-wl-code*
is *isa-pure-literal-elimination-wl*
:: ‹[λS. *length* (*get-clauses-wl-heur* S) ≤ *snat64-max* ∧ *learned-clss-count* S ≤ *unat64-max*]$_a$
isasat-bounded-assn^d → *isasat-bounded-assn*›

⟨*proof*⟩

sepref-def *isa-pure-literal-eliminate-code*
is *isa-pure-literal-eliminate*
:: ‹[λ*S*. *length* (*get-clauses-wl-heur S*) ≤ *snat64-max* ∧ *learned-clss-count S* ≤ *unat64-max*]$_a$
isasat-bounded-assn^d → *isasat-bounded-assn*›
⟨*proof*⟩

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
mar-restart-decision-lvl
          this was taken from Lingeling IIRC term ≥ of-dat level ≥ (shiftr (sema) 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*)
}›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*⟩

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*⟩

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*)
}
}›
⟨*proof*⟩

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'*)›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

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)
})›
 ⟨proof⟩


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›
 ⟨proof⟩

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*⟩

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*⟩

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*⟩

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*›
  ⟨*proof*⟩

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*)
  })›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

lemma *trail-zeroed-until-state-alt-def*:

‹*RETURN o trail-zeroed-until-state = read-trail-wl-heur (RETURN ∘ trail-zeroed-until)*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-def *FLAG-no-restart-impl*
is ‹*uncurry0* (*RETURN FLAG-no-restart*)›
:: ‹*unit-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

sepref-def *FLAG-GC-restart-impl*
is ‹*uncurry0* (*RETURN FLAG-GC-restart*)›
:: ‹*unit-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

sepref-def *FLAG-Reduce-restart-impl*
is ‹*uncurry0* (*RETURN FLAG-Reduce-restart*)›
:: ‹*unit-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

sepref-def *FLAG-Inprocess-restart-impl*
is ‹*uncurry0* (*RETURN FLAG-Inprocess-restart*)›
:: ‹*unit-assn*^{k} →$_a$ *word-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*))
})›
⟨*proof*⟩

sepref-def *update-restart-phases-impl*
is ‹*update-restart-phases*›
:: ‹*isasat-bounded-assn*^{d} →$_a$ *isasat-bounded-assn*›
⟨*proof*⟩

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*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

sepref-def *uint32-nat-assn-impl*
  is ‹*uncurry0* (*RETURN max-restart-decision-lvl*)›
  :: ‹*unit-assn*^{k} →$_a$ *uint32-nat-assn*›
  ⟨*proof*⟩

sepref-def *GC-required-heur-fast-code*
  is ‹*uncurry GC-required-heur*›
  :: ‹*isasat-bounded-assn*^{k} *$_a$ *uint64-nat-assn*^{k} →$_a$ *bool1-assn*›
  ⟨*proof*⟩

sepref-def *GC-units-required-heur-fast-code*
  is ‹*RETURN o GC-units-required*›
  :: ‹*isasat-bounded-assn*^{k} →$_a$ *bool1-assn*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

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*;

$M \leftarrow$ *replace-reason-in-trail L C M*;
RETURN (*update-trail-wl-heur M* (*update-lcount-wl-heur lcount S*))
})›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

sepref-register *number-clss-to-keep*

lemma [*sepref-fr-rules*]:
‹(*Mreturn o id*, *RETURN o unat*) ∈ *word64-assn*^{k} $\to_a$ *uint64-nat-assn*›
⟨*proof*⟩

sepref-def *number-clss-to-keep-fast-code*
is ‹*number-clss-to-keep-impl*›
:: ‹*isasat-bounded-assn*^{k} $\to_a$ *sint64-nat-assn*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*›
⟨*proof*⟩

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* S_0 = *do* {
  (*brk*, *T*, -) ← $WHILE_T^{\textit{cdcl-twl-stgy-restart-abs-wl-heur-inv}\ S_0}$
    (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*, S_0::*isasat*, *learned-clss-count* S_0, *learned-clss-count* S_0, *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*›
  ⟨*proof*⟩

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*⟩

sepref-def *mark-to-delete-clauses-wl-D-heur-fast-impl*
  is ‹*mark-to-delete-clauses-wl-D-heur*›
  :: ‹[λ*S*. *length* (*get-clauses-wl-heur S*) ≤ *snat64-max*]$_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

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*›
  ⟨*proof*⟩

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*)›
⟨*proof*⟩

sepref-register *isasat-fast*

lemma *all-count-learned*[*simp*]: ‹*clss-size-allcount* (*get-learned-count S*) = *learned-clss-count S*›
⟨*proof*⟩

sepref-def *isasat-fast-code*
is ‹*RETURN o isasat-fast*›
:: ‹[λ*S*. *isasat-fast-relaxed S*]$_a$ *isasat-bounded-assn*^{k} → *bool1-assn*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*⟩

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*)›
⟨*proof*⟩

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$$^{\lambda(ebrk,\ brk,\ T,\ last\text{-}GC,\ last\text{-}Restart,\ n).}$ *cdcl-twl-stgy-restart-abs-wl-heur-inv S*$_0$ (*brk*, *T*, *last-GC*, *last-Resta*
(λ(*ebrk*, *brk*, -). ¬*brk* ∧ ¬*ebrk*)
(λ(*ebrk*, *brk*, *S*, *last-GC*, *last-Restart*, *n*).
do {
ASSERT(¬*brk* ∧ ¬*ebrk*);
ASSERT(*length* (*get-clauses-wl-heur S*) ≤ *unat64-max*);
S ← *convert-to-full-state-wl-heur S*;
T ← *unit-propagation-outer-loop-wl-D-heur S*;
ASSERT(*length* (*get-clauses-wl-heur T*) ≤ *unat64-max*);
ASSERT(*length* (*get-clauses-wl-heur T*) = *length* (*get-clauses-wl-heur S*));
(*brk*, *T*) ← *cdcl-twl-o-prog-wl-D-heur T*;
ASSERT(*length* (*get-clauses-wl-heur T*) ≤ *unat64-max*);
(*T*, *n*) ← *restart-prog-wl-D-heur T last-GC last-Restart n brk*;
ebrk ← *RETURN* (¬*isasat-fast T*);
RETURN (*ebrk*, *brk* ∨ ¬*get-conflict-wl-is-None-heur T*, *T*, *n*)
})
(*ebrk*, *False*, *S*$_0$::*isasat*, *learned-clss-count S*$_0$, *learned-clss-count S*$_0$, *0*);
ASSERT(*length* (*get-clauses-wl-heur T*) ≤ *unat64-max* ∧
get-old-arena T = []);
if ¬*brk then do* {
T ← *isasat-fast-slow T*;
(*brk*, *T*, -) ← *WHILE*$_T$$^{cdcl\text{-}twl\text{-}stgy\text{-}restart\text{-}abs\text{-}wl\text{-}heur\text{-}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

}›
⟨proof⟩

lemma *cdcl-twl-stgy-restart-prog-early-wl-heur-cdcl-twl-stgy-restart-prog-early-wl-D*:
assumes *r*: ‹$r \leq$ *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*⟩

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*›
⟨*proof*⟩

lemma *mark-to-delete-clauses-wl-D-heur-is-Some-iff*:
‹*D* = *Some C* ⟷ *D* ≠ *None* ∧ ((*the D*) = *C*)›
⟨*proof*⟩

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-GC*, *last-Restart*, *n*). *cdcl-twl-stgy-restart-abs-wl-heur-inv* S_0 (*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*);
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*)
}›
⟨*proof*⟩

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*⟩

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*⟩

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*))›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
  ⟨*proof*⟩

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*)›
  ⟨*proof*⟩

lemma *isasat-GC-clauses-prog-wl-alt-def*:
  ‹*isasat-GC-clauses-prog-wl* = (λ*S*$_0$. *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*
    })›
    ⟨*proof*⟩

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*›
  ⟨*proof*⟩

lemma *rewatch-heur-st-pre-alt-def*:
  ‹*rewatch-heur-st-pre* *S* ⟷ (∀ *i* ∈ *set* (*get-tvdom* *S*). *i* ≤ *snat64-max*)›
  ⟨*proof*⟩

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*›
⟨*proof*⟩

lemma *rewatch-heur-and-reorder-st-alt-def*:
‹*rewatch-heur-and-reorder-st* = (λ*S*$_0$. *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*)))
})›
⟨*proof*⟩

sepref-register *rewatch-heur-and-reorder rewatch-heur-and-reorder-vdom*
sepref-def *rewatch-heur-st-code*
is ‹*rewatch-heur-and-reorder-st*›
:: ‹[λ*S*. *rewatch-heur-st-pre* *S* ∧ *length* (*get-clauses-wl-heur* *S*) ≤ *snat64-max*]$_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*›
:: ‹[λ*S*. *length* (*get-clauses-wl-heur* *S*) ≤ *snat64-max*]$_a$ *isasat-bounded-assn*^{d} → *isasat-bounded-assn*›
⟨*proof*⟩

sepref-def *cdcl-twl-mark-clauses-to-delete-fast-code*
is ‹*cdcl-twl-mark-clauses-to-delete*›
:: ‹$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq snat64\text{-}max]_a\ isasat\text{-}bounded\text{-}assn^d \to isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

sepref-def *cdcl-twl-full-restart-wl-D-GC-heur-prog-fast-code*
is ‹*cdcl-twl-full-restart-wl-D-GC-heur-prog*›
:: ‹$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq snat64\text{-}max \wedge learned\text{-}clss\text{-}count\ S \leq unat64\text{-}max]_a$
$isasat\text{-}bounded\text{-}assn^d \to isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

sepref-def *cdcl-twl-full-restart-wl-D-inprocess-heur-prog-fast-code*
is ‹*cdcl-twl-full-restart-wl-D-inprocess-heur-prog*›
:: ‹$[\lambda S.\ length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S) \leq snat64\text{-}max \wedge learned\text{-}clss\text{-}count\ S \leq unat64\text{-}max]_a$
$isasat\text{-}bounded\text{-}assn^d \to isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

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*›
:: ‹$[\lambda((((S,\ -),\ -),\ n),\ -).\ isasat\text{-}fast\text{-}relaxed2\ S\ n]_a$
$isasat\text{-}bounded\text{-}assn^d *_a uint64\text{-}nat\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a uint64\text{-}nat\text{-}assn^k *_a$
$bool1\text{-}assn^k \to isasat\text{-}bounded\text{-}assn \times_a uint64\text{-}nat\text{-}assn \times_a uint64\text{-}nat\text{-}assn \times_a uint64\text{-}nat\text{-}assn$›
⟨*proof*⟩

lemma *cdcl-twl-stgy-restart-prog-bounded-wl-heurI1*:
assumes
‹*isasat-fast a1'b*›
‹$learned\text{-}clss\text{-}count\ a2'e \leq Suc\ (learned\text{-}clss\text{-}count\ a1'b)$›
shows ‹$learned\text{-}clss\text{-}count\ a2'e \leq unat64\text{-}max$›
⟨*proof*⟩

lemma *cdcl-twl-stgy-restart-prog-bounded-wl-heurI2*:
‹$isasat\text{-}fast\ x \Longrightarrow learned\text{-}clss\text{-}count\ x \leq unat64\text{-}max$›
⟨*proof*⟩

lemma *cdcl-twl-stgy-restart-prog-bounded-wl-heurI3*:
assumes
‹*isasat-fast S*›
‹$length\ (get\text{-}clauses\text{-}wl\text{-}heur\ T) \leq length\ (get\text{-}clauses\text{-}wl\text{-}heur\ S)$›
‹$learned\text{-}clss\text{-}count\ T \leq (learned\text{-}clss\text{-}count\ S)$›
shows ‹*isasat-fast T*›
⟨*proof*⟩

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*›
:: ‹$[\lambda S.\ isasat\text{-}fast\ S]_a\ isasat\text{-}bounded\text{-}assn^d \to bool1\text{-}assn \times_a isasat\text{-}bounded\text{-}assn$›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)))›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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)›
 ⟨proof⟩


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›
 ⟨proof⟩


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›
 ⟨proof⟩

lemma satisfiable-remdups-iff:
 ‹satisfiable ((λx. remdups-mset (mset x)) ` CS) ⟷ satisfiable (mset ` CS)›
 ⟨proof⟩

lemma true-annots-remdups-mset:
 ‹I ⊨as remdups-mset ` C ⟷ I ⊨as C›
 ⟨proof⟩

lemma SAT0-SAT:
 shows ‹SAT0 CS ≤ ⇓ {(S, T). T = pstate_W-of S} (SAT (mset `# mset CS))›
⟨proof⟩

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›
  ⟨proof⟩


lemma SAT-l-SAT0:
```

shows ‹*SAT-l CS* ≤ ⇓ {(*T*,*T'*). (*T*, *T'*) ∈ *twl-st-l None*} (*SAT0 CS*)›
⟨*proof*⟩

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 $\mathcal{A}_{in}'$ = *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* $\mathcal{A}_{in}'$));
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*) = {#});
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* $\mathcal{A}_{in}'$));
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*) = {#});
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* $\mathcal{A}_{in}'$));
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-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
}
}
}
}›
⟨*proof*⟩

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*⟩

lemma *init-dt-wl-pre*:
shows ‹*init-dt-wl-pre CS* (*to-init-state init-state-wl*)›
⟨*proof*⟩

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*⟩

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⟩


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)))›
  ⟨proof⟩


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 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'');
          - ← 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 𝒜in';
        (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 𝒜in' = mset-set (extract-atms-clss CS {});
          S ← init-state-wl-heur 𝒜in';
          (T::twl-st-wl-heur-init, -) ← init-dt-wl-heur True CS (S, []);
          let T = convert-state 𝒜in'' 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*⟩

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*⟩

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*⟩

lemma *all-lits-of-mm-empty-iff*: ‹*all-lits-of-mm A* = {#} ⟷ (∀ *C* ∈# *A*. *C* = {#})›
  ⟨*proof*⟩

lemma *all-lits-of-mm-extract-atms-clss*:

‹*L* ∈# (*all-lits-of-mm* (*mset* '# *mset CS*)) ⟷ *atm-of L* ∈ *extract-atms-clss CS* {}›
⟨*proof*⟩

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 $\mathcal{A}_{in}'$ = *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($\mathcal{A}_{in}''$ ≠ {#});
ASSERT(*isasat-input-bounded-nempty* $\mathcal{A}_{in}''$);
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* $\mathcal{A}_{in}'$;
(*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* $\mathcal{A}_{in}'$;
(*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($\mathcal{A}_{in}''$ ≠ {#});
ASSERT(*isasat-input-bounded-nempty* $\mathcal{A}_{in}''$);
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* $\mathcal{A}_{in}''$ *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($\mathcal{A}_{in}'' \neq \{\#\}$);
ASSERT(*isasat-input-bounded-nempty* $\mathcal{A}_{in}''$);
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)
}
}
}
}›
⟨*proof*⟩

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*⟩

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*)

$\leq \Downarrow (\{(S,T).\ (S,\ T) \in$ *twl-st-heur-parsing* (*mset-set* (*extract-atms-clss CS* {})) *True* $\wedge$
get-clauses-wl-heur-init S = *get-clauses-wl-heur-init U* $\wedge$
get-conflict-wl-heur-init S = *get-conflict-wl-heur-init U* $\wedge$
get-clauses-wl (*fst T*) = *get-clauses-wl* (*fst V*) $\wedge$
get-conflict-wl (*fst T*) = *get-conflict-wl* (*fst V*) $\wedge$
get-unit-clauses-wl (*fst T*) = *get-unit-clauses-wl* (*fst V*)} *O* {(*S*, *T*). *S* = (*T*, {#})})
(*rewatch-st* (*from-init-state V*))›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

lemma *IsaSAT-heur-IsaSAT*:
‹*IsaSAT-heur b CS* ≤ ⇓*model-stat-rel* (*IsaSAT CS*)›
⟨*proof*⟩

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*⟩

lemma *SAT-model-if-satisfiable′*:
‹(*uncurry* (λ-. *SAT′*), *uncurry* (λ-. *model-if-satisfiable*)) ∈
[λ(-, *CS*). *True*]$_f$ *Id* ×$_r$ *Id*→ ⟨*Id*⟩*nres-rel*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

lemma *ISASAT-SAT-l′*:
assumes
‹*isasat-input-bounded* (*mset-set* (⋃ *C*∈*set CS*. *atm-of* ‘ *set C*))›
shows ‹*IsaSAT CS* ≤ ⇓ *Id* (*SAT-l′ CS*)›
⟨*proof*⟩

lemma *SAT-l′-SAT0′*:
shows ‹*SAT-l′ CS* ≤ ⇓ *Id* (*SAT0′ CS*)›
⟨*proof*⟩

lemma *SAT0′-SAT′*:
shows ‹*SAT0′ CS* ≤ ⇓ *Id* (*SAT′* (*mset* ‘# *mset CS*))›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*⟩

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*(λ*T*. *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* (λ*T*. *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*⟩

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*⟩

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 𝒜in′ = 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 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-bounded-wl T
    }
  }
}›
```

lemma *SAT-l-bounded-alt-def*:

```
‹SAT-l-bounded CS = do{
  𝒜 ← RETURN (); ////////
  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 {
    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
}
}
}›
⟨*proof*⟩

lemma *SAT-wl-bounded-SAT-l-bounded*:
assumes
bounded: ‹*isasat-input-bounded* (*mset-set* (⋃ *C*∈*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*⟩

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*) ∈ [λ*CS*. *True*]*f* *Id*→
⟨{((*b*, *S*), (*b*′, *T*)). *b* = *b*′ ∧ (¬*b* ⟶ *S* = *T*)}⟩*nres-rel*›
(**is** ‹- ∈[λ*CS*. *?P CS*]*f* *Id* → -›)
⟨*proof*⟩

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*›
⟨*proof*⟩

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*)›
⟨*proof*⟩

lemma *SAT0-bounded′-SAT-bounded′*:
shows ‹*SAT0-bounded′ CS* ≤ ⇓ {((*b*, *S*), (*b′*, *T*)). *b* = *b′* ∧ (¬*b* ⟶ *S* = *T*)} (*SAT-bounded′* (*mset* ‘# *mset CS*))›
⟨*proof*⟩

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 $\mathcal{A}_{in}'$ = *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* $\mathcal{A}_{in}'$));
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*⟩

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 $\mathcal{A}_{in}'$ = *mset-set* (*extract-atms-clss CS* {});
ASSERT(*isasat-input-bounded* $\mathcal{A}_{in}'$);
ASSERT(*distinct-mset* $\mathcal{A}_{in}'$);
S ← *init-state-wl-heur* $\mathcal{A}_{in}'$;
(*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* $\mathcal{A}_{in}'$ *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($\mathcal{A}_{in}'$ ≠ {#});
ASSERT(*isasat-input-bounded-nempty* $\mathcal{A}_{in}'$);
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)
}
}
}›
⟨*proof*⟩

lemma *IsaSAT-heur-bounded-IsaSAT-bounded*:
‹*IsaSAT-bounded-heur b CS* ≤ ⇓(*bool-rel* ×$_f$ *model-stat-rel*) (*IsaSAT-bounded CS*)›
⟨*proof*⟩

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*)›
⟨*proof*⟩

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*))›
⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

sepref-def *print-confl-impl*
is ‹*RETURN o print-confl*›
:: ‹*word-assn*^{k} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-def *print-dec-impl*
is ‹*RETURN o print-dec*›
:: ‹*word-assn*^{k} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-def *print-res-impl*
is ‹*uncurry* (*RETURN oo print-res*)›
:: ‹*word-assn*^{k} *$_a$ *word-assn*^{k} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-def *print-lres-impl*
is ‹*uncurry* (*RETURN oo print-lres*)›
:: ‹*word-assn*^{k} *$_a$ *word-assn*^{k} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-def *print-uset-impl*
is ‹*RETURN o print-uset*›
:: ‹*word-assn*^{k} →$_a$ *unit-assn*›
⟨*proof*⟩

definition *print-irred-clss* :: ‹*64 word* ⇒ *unit*› **where**
‹*print-irred-clss - =* ()›

sepref-def *print-gc-impl*
is ‹*uncurry* (*RETURN oo print-gcs*)›
:: ‹*word-assn*^{k} *$_a$ *word-assn*^{k} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-def *print-irred-clss-impl*
is ‹*RETURN o print-irred-clss*›
:: ‹*word-assn*^{k} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-def *print-binary-unit-impl*
is ‹*RETURN o print-binary-unit*›
:: ‹*word-assn*^{k} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-def *print-purelit-rounds-impl*
is ‹*uncurry* (*RETURN oo print-purelit-rounds*)›
:: ‹*word-assn*^{k} *$_a$ *word-assn*^{k} →$_a$ *unit-assn*›
⟨*proof*⟩

sepref-def *print-purelit-elim-impl*
  is ‹*RETURN o print-purelit-elim*›
  :: ‹*word-assn*^{k} →$_a$ *unit-assn*›
  ⟨*proof*⟩

sepref-def *print-binary-red-removed-impl*
  is ‹*RETURN o print-binary-red-removed*›
  :: ‹*word-assn*^{k} →$_a$ *unit-assn*›
  ⟨*proof*⟩

sepref-def *print-forward-rounds-impl*
  is ‹*uncurry* (*RETURN oo print-forward-rounds*)›
  :: ‹*word-assn*^{k} *$_a$ *word-assn*^{k} →$_a$ *unit-assn*›
  ⟨*proof*⟩

sepref-def *print-forward-subsumed-impl*
  is ‹*uncurry* (*RETURN oo print-forward-subsumed*)›
  :: ‹*word-assn*^{k} *$_a$ *word-assn*^{k} →$_a$ *unit-assn*›
  ⟨*proof*⟩

sepref-def *print-forward-tried-impl*
  is ‹*uncurry* (*RETURN oo print-forward-tried*)›
  :: ‹*word-assn*^{k} *$_a$ *word-assn*^{k} →$_a$ *unit-assn*›
  ⟨*proof*⟩

sepref-def *print-forward-strengthened-impl*
  is ‹*uncurry* (*RETURN oo print-forward-strengthened*)›
  :: ‹*word-assn*^{k} *$_a$ *word-assn*^{k} →$_a$ *unit-assn*›
  ⟨*proof*⟩

sepref-def *print-rephased-impl*
  is ‹*uncurry* (*RETURN oo print-rephased*)›
  :: ‹*word-assn*^{k} *$_a$ *word-assn*^{k} →$_a$ *unit-assn*›
  ⟨*proof*⟩

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 →$_a$
     isasat-init-assn›
  ⟨*proof*⟩

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*⟩

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*›
⟨*proof*⟩

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*›
⟨*proof*⟩

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›
  ⟨proof⟩

lemma extract-model-of-state-stat-alt-def:
  ‹RETURN o extract-model-of-state-stat = (λS. 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)
    })›
  ⟨proof⟩

schematic-goal mk-free-lbd-assn[sepref-frame-free-rules]: ‹MK-FREE aivdom-assn ?fr›
  ⟨proof⟩

sepref-def extract-model-of-state-stat
  is ‹RETURN o extract-model-of-state-stat›
  :: ‹isasat-bounded-assn^d →a model-stat-assn›
  ⟨proof⟩

lemma extract-state-stat-alt-def:
  ‹RETURN o extract-state-stat = (λS. 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)})›
  ⟨proof⟩

sepref-def extract-state-stat
  is ‹RETURN o extract-state-stat›
  :: ‹isasat-bounded-assn^d →a model-stat-assn›
  ⟨proof⟩

sepref-def empty-conflict-code'
  is ‹uncurry0 (empty-conflict-code)›
  :: ‹unit-assn^k →a model-stat-assn›
  ⟨proof⟩

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›
  ⟨proof⟩
```

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*›
⟨*proof*⟩

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*)
}
}

}
})›
⟨*proof*⟩

lemma *isasat-fast-init-codeI*: ‹(*aa* :: *64 word*, *b*) ∈ *unat-rel64* ⟹ *b* ≤ *18446744073709551615*›
⟨*proof*⟩

sepref-def *isasat-fast-init-code*
is ‹*RETURN o isasat-fast-init*›
:: ‹*isasat-init-assn*^{k} →$_a$ *bool1-assn*›
⟨*proof*⟩

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*)
∈ [λ*x*. *distinct-mset x* ∧
(∀ *L*∈#$\mathcal{L}_{all}$ *x*. *nat-of-lit L* ≤ *unat32-max*)]$_a$ *lits-with-max-assn*^{k} → *isasat-init-assn*›
⟨*proof*⟩

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*⟩

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 (())›
⟨*proof*⟩

schematic-goal *mk-free-ghost-assn*[*sepref-frame-free-rules*]: ‹*MK-FREE ghost-assn ?fr*›
⟨*proof*⟩

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 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)}
    }›
⟨proof⟩

sepref-def IsaSAT-code
  is ‹uncurry IsaSAT-bounded-heur›
  :: ‹opts-assn^d *_a (clauses-ll-assn)^d →_a bool1-assn ×_a model-stat-assn›
⟨proof⟩


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 $\to_a$ *sint64-nat-assn*›
  ⟨*proof*⟩

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*]))›
  ⟨*proof*⟩
termination
  ⟨*proof*⟩

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} $\to_a$ *IICF-Array.array-assn sint32-nat-assn*›
  ⟨*proof*⟩

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* ×$_a$ *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›
 ⟨*proof*⟩

end