Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

July 17, 2023

Contents

0.1 CDCL Extensions 3
0.1.1 Optimisations 3
0.1.2 Encoding of partial SAT into total SAT 73
0.1.3 Partial MAX-SAT 98
0.2 Covering Models 137

theory *CDCL-W-BnB*
imports *CDCL.CDCL-W-Abstract-State*
begin

0.1 CDCL Extensions

A counter-example for the original version from the book has been found (see below). There is no simple fix, except taking complete models.

Based on Dominik Zimmer's thesis, we later reduced the problem of finding partial models to finding total models. We later switched to the more elegant dual rail encoding (thanks to the reviewer).

0.1.1 Optimisations

notation *image-mset* (**infixr** ‹'#› *90*)

The initial version was supposed to work on partial models directly. I found a counterexample while writing the proof:

Nitpicking 0.1.

Christoph's book draft 0.1. $(M;N;U;k;\top;O) \Rightarrow^{\textit{Propagate}} (ML^{C\vee L};N;U;k;\top;O)$
provided $C \vee L \in (N \cup U)$, $M \models \neg C$, L *is undefined in* M.

$(M;N;U;k;\top;O) \Rightarrow^{\textit{Decide}} (ML^{k+1};N;U;k+1;\top;O)$
provided L *is undefined in* M, *contained in* N.

$(M;N;U;k;\top;O) \Rightarrow^{\textit{ConflSat}} (M;N;U;k;D;O)$
provided $D \in (N \cup U)$ *and* $M \models \neg D$.

$(M;N;U;k;\top;O) \Rightarrow^{\textit{ConflOpt}} (M;N;U;k;\neg M;O)$
provided $O \neq \epsilon$ *and* $\mathrm{cost}(M) \geq \mathrm{cost}(O)$.

$(ML^{C\vee L};N;U;k;D;O) \Rightarrow^{\textit{Skip}} (M;N;U;k;D;O)$
provided $D \notin \{\top, \bot\}$ *and* $\neg L$ *does not occur in* D.

$(ML^{C\vee L};N;U;k;D \vee -(L);O) \Rightarrow^{\textit{Resolve}} (M;N;U;k;D \vee C;O)$
provided D *is of level* k.

$(M_1 K^{i+1} M_2;N;U;k;D \vee L;O) \Rightarrow^{\textit{Backtrack}} (M_1 L^{D\vee L};N;U \cup \{D \vee L\};i;\top;O)$
provided L *is of level* k *and* D *is of level* i.

$(M;N;U;k;\top;O) \Rightarrow^{\textit{Improve}} (M;N;U;k;\top;M)$
provided $M \models N$ *and* $O = \epsilon$ *or* $\mathrm{cost}(M) < \mathrm{cost}(O)$.

This calculus does not always find the model with minimum cost. Take for example the following cost function:

$$\mathrm{cost}: \begin{cases} P \to 3 \\ \neg P \to 1 \\ Q \to 1 \\ \neg Q \to 1 \end{cases}$$

and the clauses $N = \{P \vee Q\}$. *We can then do the following transitions:*
$(\epsilon, N, \varnothing, \top, \infty)$
$\Rightarrow^{\textit{Decide}} (P^1, N, \varnothing, \top, \infty)$
$\Rightarrow^{\textit{Improve}} (P^1, N, \varnothing, \top, (P, 3))$
$\Rightarrow^{\textit{conflictOpt}} (P^1, N, \varnothing, \neg P, (P, 3))$
$\Rightarrow^{\textit{backtrack}} (\neg P^{\neg P}, N, \{\neg P\}, \top, (P, 3))$
$\Rightarrow^{\textit{propagate}} (\neg P^{\neg P} Q^{P\vee Q}, N, \{\neg P\}, \top, (P, 3))$
$\Rightarrow^{\textit{improve}} (\neg P^{\neg P} Q^{P\vee Q}, N, \{\neg P\}, \top, (\neg P\, Q, 2))$
$\Rightarrow^{\textit{conflictOpt}} (\neg P^{\neg P} Q^{P\vee Q}, N, \{\neg P\}, P \vee \neg Q, (\neg P\, Q, 2))$
$\Rightarrow^{\textit{resolve}} (\neg P^{\neg P}, N, \{\neg P\}, P, (\neg P\, Q, 2))$
$\Rightarrow^{\textit{resolve}} (\epsilon, N, \{\neg P\}, \bot, (\neg P\, Q, 3))$
However, the optimal model is Q.

The idea of the proof (explained of the example of the optimising CDCL) is the following:

1. We start with a calculus OCDCL on (M, N, U, D, Op).

2. This extended to a state (M, N + *all-models-of-higher-cost*, U, D, Op).

3. Each transition step of OCDCL is mapped to a step in CDCL over the abstract state. The abstract set of clauses might be unsatisfiable, but we only use it to prove the invariants on the state. Only adding clause cannot be mapped to a transition over the abstract state, but adding clauses does not break the invariants (as long as the additional clauses do not contain duplicate literals).

4. The last proofs are done over CDCLopt.

We abstract about how the optimisation is done in the locale below: We define a calculus *cdcl-bnb* (for branch-and-bounds). It is parametrised by how the conflicting clauses are generated and the improvement criterion.

We later instantiate it with the optimisation calculus from Weidenbach's book.

Helper libraries

definition *model-on* :: ‹'*v partial-interp* ⇒ '*v clauses* ⇒ *bool*› **where**
‹*model-on I N* ⟷ *consistent-interp I* ∧ *atm-of* ‘ *I* ⊆ *atms-of-mm N*›

CDCL BNB

locale *conflict-driven-clause-learning-with-adding-init-clause-bnb*$_W$*-no-state* =
  state$_W$*-no-state*
    state-eq state
    — functions for the state:
      — access functions:
    trail init-clss learned-clss conflicting
      — changing state:
    cons-trail tl-trail add-learned-cls remove-cls
    update-conflicting

      — get state:
    init-state
  for
    state-eq :: ‹'*st* ⇒ '*st* ⇒ *bool*› (**infix** ‹∼› *50*) **and**
    state :: ‹'*st* ⇒ ('*v*, '*v clause*) *ann-lits* × '*v clauses* × '*v clauses* × '*v clause option* ×
      '*a* × '*b*› **and**
    trail :: ‹'*st* ⇒ ('*v*, '*v clause*) *ann-lits*› **and**
    init-clss :: ‹'*st* ⇒ '*v clauses*› **and**
    learned-clss :: ‹'*st* ⇒ '*v clauses*› **and**
    conflicting :: ‹'*st* ⇒ '*v clause option*› **and**

    cons-trail :: ‹('*v*, '*v clause*) *ann-lit* ⇒ '*st* ⇒ '*st*› **and**
    tl-trail :: ‹'*st* ⇒ '*st*› **and**
    add-learned-cls :: ‹'*v clause* ⇒ '*st* ⇒ '*st*› **and**
    remove-cls :: ‹'*v clause* ⇒ '*st* ⇒ '*st*› **and**
    update-conflicting :: ‹'*v clause option* ⇒ '*st* ⇒ '*st*› **and**

    init-state :: ‹'*v clauses* ⇒ '*st*› +
  fixes
    update-weight-information :: ‹('*v*, '*v clause*) *ann-lits* ⇒ '*st* ⇒ '*st*› **and**
    is-improving-int :: ‹('*v*, '*v clause*) *ann-lits* ⇒ ('*v*, '*v clause*) *ann-lits* ⇒ '*v clauses* ⇒ '*a* ⇒ *bool*› **and**
    conflicting-clauses :: ‹'*v clauses* ⇒ '*a* ⇒ '*v clauses*› **and**
    weight :: ‹'*st* ⇒ '*a*›

begin

abbreviation *is-improving* **where**
‹*is-improving M M′ S* ≡ *is-improving-int M M′* (*init-clss S*) (*weight S*)›

definition *additional-info′* :: ‹′*st* ⇒ ′*b*› **where**
‹*additional-info′ S* = (λ(-, -, -, -, -, *D*). *D*) (*state S*)›

definition *conflicting-clss* :: ‹′*st* ⇒ ′*v literal multiset multiset*› **where**
‹*conflicting-clss S* = *conflicting-clauses* (*init-clss S*) (*weight S*)›

While it would more be natural to add an sublocale with the extended version clause set, this actually causes a loop in the hierarchy structure (although with different parameters). Therefore, adding theorems (e.g. defining an inductive predicate) causes a loop.

definition *abs-state*
:: ‹′*st* ⇒ (′*v*, ′*v clause*) *ann-lit list* × ′*v clauses* × ′*v clauses* × ′*v clause option*›
where
‹*abs-state S* = (*trail S*, *init-clss S* + *conflicting-clss S*, *learned-clss S*,
conflicting S)›

end

locale *conflict-driven-clause-learning-with-adding-init-clause-bnb*$_W$*-ops* =
conflict-driven-clause-learning-with-adding-init-clause-bnb$_W$*-no-state*
state-eq state
— functions for the state:
— access functions:
trail init-clss learned-clss conflicting
— changing state:
cons-trail tl-trail add-learned-cls remove-cls
update-conflicting

— get state:
init-state
— Adding a clause:
update-weight-information is-improving-int conflicting-clauses weight
for
state-eq :: ‹′*st* ⇒ ′*st* ⇒ *bool*› (**infix** ‹∼› *50*) **and**
state :: ′*st* ⇒ (′*v*, ′*v clause*) *ann-lits* × ′*v clauses* × ′*v clauses* × ′*v clause option* ×
′*a* × ′*b* **and**
trail :: ‹′*st* ⇒ (′*v*, ′*v clause*) *ann-lits*› **and**
init-clss :: ‹′*st* ⇒ ′*v clauses*› **and**
learned-clss :: ‹′*st* ⇒ ′*v clauses*› **and**
conflicting :: ‹′*st* ⇒ ′*v clause option*› **and**

cons-trail :: ‹(′*v*, ′*v clause*) *ann-lit* ⇒ ′*st* ⇒ ′*st*› **and**
tl-trail :: ‹′*st* ⇒ ′*st*› **and**
add-learned-cls :: ‹′*v clause* ⇒ ′*st* ⇒ ′*st*› **and**
remove-cls :: ‹′*v clause* ⇒ ′*st* ⇒ ′*st*› **and**
update-conflicting :: ‹′*v clause option* ⇒ ′*st* ⇒ ′*st*› **and**

init-state :: ‹′*v clauses* ⇒ ′*st*› **and**
update-weight-information :: ‹(′*v*, ′*v clause*) *ann-lits* ⇒ ′*st* ⇒ ′*st*› **and**
is-improving-int :: (′*v*, ′*v clause*) *ann-lits* ⇒ (′*v*, ′*v clause*) *ann-lits* ⇒ ′*v clauses* ⇒
′*a* ⇒ *bool* **and**

conflicting-clauses :: ‹'*v clauses* ⇒ '*a* ⇒ '*v clauses*› **and**
weight :: ‹'*st* ⇒ '*a*› +
assumes
state-prop':
‹*state S* = (*trail S*, *init-clss S*, *learned-clss S*, *conflicting S*, *weight S*, *additional-info' S*)›
and
update-weight-information:
‹*state S* = (*M*, *N*, *U*, *C*, *w*, *other*) ⟹
∃ *w'*. *state* (*update-weight-information T S*) = (*M*, *N*, *U*, *C*, *w'*, *other*)› **and**
atms-of-conflicting-clss:
‹*atms-of-mm* (*conflicting-clss S*) ⊆ *atms-of-mm* (*init-clss S*)› **and**
distinct-mset-mset-conflicting-clss:
‹*distinct-mset-mset* (*conflicting-clss S*)› **and**
conflicting-clss-update-weight-information-mono:
‹$cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv* (*abs-state S*) ⟹ *is-improving M M' S* ⟹
conflicting-clss S ⊆# *conflicting-clss* (*update-weight-information M' S*)›
and
conflicting-clss-update-weight-information-in:
‹*is-improving M M' S* ⟹
negate-ann-lits M' ∈# *conflicting-clss* (*update-weight-information M' S*)›
begin

Conversion to CDCL **sublocale** *conflict-driven-clause-learning*$_W$ **where**
state-eq = *state-eq* **and**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
apply *unfold-locales*
unfolding *additional-info'-def additional-info-def* **by** (*auto simp*: *state-prop'*)

Overall simplification on states **declare** *reduce-trail-to-skip-beginning*[*simp*]

lemma *state-eq-weight*[*state-simp*, *simp*]: ‹*S* ∼ *T* ⟹ *weight S* = *weight T*›
apply (*drule state-eq-state*)
apply (*subst* (*asm*) *state-prop'*)+
by *simp*

lemma *conflicting-clause-state-eq*[*state-simp*, *simp*]:
‹*S* ∼ *T* ⟹ *conflicting-clss S* = *conflicting-clss T*›
unfolding *conflicting-clss-def* **by** *auto*

lemma
weight-cons-trail[*simp*]:
‹*weight* (*cons-trail L S*) = *weight S*› **and**
weight-update-conflicting[*simp*]:
‹*weight* (*update-conflicting C S*) = *weight S*› **and**

weight-tl-trail[*simp*]:
‹*weight* (*tl-trail S*) = *weight S*› **and**
weight-add-learned-cls[*simp*]:
‹*weight* (*add-learned-cls D S*) = *weight S*›
using *cons-trail*[*of S - - L*] *update-conflicting*[*of S*] *tl-trail*[*of S*] *add-learned-cls*[*of S*]
by (*auto simp*: *state-prop′*)

lemma *update-weight-information-simp*[*simp*]:
‹*trail* (*update-weight-information C S*) = *trail S*›
‹*init-clss* (*update-weight-information C S*) = *init-clss S*›
‹*learned-clss* (*update-weight-information C S*) = *learned-clss S*›
‹*clauses* (*update-weight-information C S*) = *clauses S*›
‹*backtrack-lvl* (*update-weight-information C S*) = *backtrack-lvl S*›
‹*conflicting* (*update-weight-information C S*) = *conflicting S*›
using *update-weight-information*[*of S*] **unfolding** *clauses-def*
by (*subst* (*asm*) *state-prop′*, *subst* (*asm*) *state-prop′*; *force*)+

lemma
conflicting-clss-cons-trail[*simp*]: ‹*conflicting-clss* (*cons-trail K S*) = *conflicting-clss S*› **and**
conflicting-clss-tl-trail[*simp*]: ‹*conflicting-clss* (*tl-trail S*) = *conflicting-clss S*› **and**
conflicting-clss-add-learned-cls[*simp*]:
‹*conflicting-clss* (*add-learned-cls D S*) = *conflicting-clss S*› **and**
conflicting-clss-update-conflicting[*simp*]:
‹*conflicting-clss* (*update-conflicting E S*) = *conflicting-clss S*›
unfolding *conflicting-clss-def* **by** *auto*

lemma *conflicting-abs-state-conflicting*[*simp*]:
‹*CDCL-W-Abstract-State.conflicting* (*abs-state S*) = *conflicting S*› **and**
clauses-abs-state[*simp*]:
‹$cdcl_W$*-restart-mset.clauses* (*abs-state S*) = *clauses S* + *conflicting-clss S*› **and**
abs-state-tl-trail[*simp*]:
‹*abs-state* (*tl-trail S*) = *CDCL-W-Abstract-State.tl-trail* (*abs-state S*)› **and**
abs-state-add-learned-cls[*simp*]:
‹*abs-state* (*add-learned-cls C S*) = *CDCL-W-Abstract-State.add-learned-cls C* (*abs-state S*)› **and**
abs-state-update-conflicting[*simp*]:
‹*abs-state* (*update-conflicting D S*) = *CDCL-W-Abstract-State.update-conflicting D* (*abs-state S*)›
by (*auto simp*: *conflicting.simps abs-state-def* $cdcl_W$*-restart-mset.clauses-def*
init-clss.simps learned-clss.simps clauses-def tl-trail.simps
add-learned-cls.simps update-conflicting.simps)

lemma *sim-abs-state-simp*: ‹$S \sim T \Longrightarrow$ *abs-state S* = *abs-state T*›
by (*auto simp*: *abs-state-def*)

lemma *reduce-trail-to-update-weight-information*[*simp*]:
‹*trail* (*reduce-trail-to M* (*update-weight-information M′ S*)) = *trail* (*reduce-trail-to M S*)›
unfolding *trail-reduce-trail-to-drop* **by** *auto*

lemma *additional-info-weight-additional-info′*: ‹*additional-info S* = (*weight S*, *additional-info′ S*)›
using *state-prop*[*of S*] *state-prop′*[*of S*] **by** *auto*

lemma
weight-reduce-trail-to[*simp*]: ‹*weight* (*reduce-trail-to M S*) = *weight S*› **and**
additional-info′-reduce-trail-to[*simp*]: ‹*additional-info′* (*reduce-trail-to M S*) = *additional-info′ S*›
using *additional-info-reduce-trail-to*[*of M S*] **unfolding** *additional-info-weight-additional-info′*
by *auto*

lemma *conflicting-clss-reduce-trail-to*[*simp*]:
‹*conflicting-clss* (*reduce-trail-to M S*) = *conflicting-clss S*›
unfolding *conflicting-clss-def* **by** *auto*

lemma *trail-trail* [*simp*]:
‹*CDCL-W-Abstract-State.trail* (*abs-state S*) = *trail S*›
by (*auto simp*: *abs-state-def* $cdcl_W$*-restart-mset-state*)

lemma [*simp*]:
‹*CDCL-W-Abstract-State.trail* ($cdcl_W$*-restart-mset.reduce-trail-to M* (*abs-state S*)) =
trail (*reduce-trail-to M S*)›
by (*auto simp*: *trail-reduce-trail-to-drop*
$cdcl_W$*-restart-mset.trail-reduce-trail-to-drop*)

lemma *abs-state-cons-trail*[*simp*]:
‹*abs-state* (*cons-trail K S*) = *CDCL-W-Abstract-State.cons-trail K* (*abs-state S*)› **and**
abs-state-reduce-trail-to[*simp*]:
‹*abs-state* (*reduce-trail-to M S*) = $cdcl_W$*-restart-mset.reduce-trail-to M* (*abs-state S*)›
subgoal by (*auto simp*: *abs-state-def cons-trail.simps*)
subgoal by (*induction rule*: *reduce-trail-to-induct*)
(*auto simp*: *reduce-trail-to.simps* $cdcl_W$*-restart-mset.reduce-trail-to.simps*)
done

lemma *learned-clss-learned-clss*[*simp*]:
‹*CDCL-W-Abstract-State.learned-clss* (*abs-state S*) = *learned-clss S*›
by (*auto simp*: *abs-state-def* $cdcl_W$*-restart-mset-state*)

lemma *state-eq-init-clss-abs-state*[*state-simp*, *simp*]:
‹$S \sim T \Longrightarrow$ *CDCL-W-Abstract-State.init-clss* (*abs-state S*) = *CDCL-W-Abstract-State.init-clss* (*abs-state T*)›
by (*auto simp*: *abs-state-def* $cdcl_W$*-restart-mset-state*)

lemma
init-clss-abs-state-update-conflicting[*simp*]:
‹*CDCL-W-Abstract-State.init-clss* (*abs-state* (*update-conflicting* (*Some D*) *S*)) =
CDCL-W-Abstract-State.init-clss (*abs-state S*)› **and**
init-clss-abs-state-cons-trail[*simp*]:
‹*CDCL-W-Abstract-State.init-clss* (*abs-state* (*cons-trail K S*)) =
CDCL-W-Abstract-State.init-clss (*abs-state S*)›
by (*auto simp*: *abs-state-def* $cdcl_W$*-restart-mset-state*)

CDCL with branch-and-bound **inductive** *conflict-opt* :: ‹$'st \Rightarrow 'st \Rightarrow$ *bool*› **for** *S T* :: $'st$ **where**
conflict-opt-rule:
‹*conflict-opt S T*›
if
‹*negate-ann-lits* (*trail S*) ∈# *conflicting-clss S*›
‹*conflicting S* = *None*›
‹$T \sim$ *update-conflicting* (*Some* (*negate-ann-lits* (*trail S*))) *S*›

inductive-cases *conflict-optE*: ‹*conflict-opt S T*›

inductive *improvep* :: ‹$'st \Rightarrow 'st \Rightarrow$ *bool*› **for** *S* :: $'st$ **where**
improve-rule:
‹*improvep S T*›

```
  if
    ‹is-improving (trail S) M' S› and
    ‹conflicting S = None› and
    ‹T ∼ update-weight-information M' S›

inductive-cases improveE: ‹improvep S T›

lemma invs-update-weight-information[simp]:
  ‹no-strange-atm (update-weight-information C S) = (no-strange-atm S)›
  ‹cdcl_W-M-level-inv (update-weight-information C S) = cdcl_W-M-level-inv S›
  ‹distinct-cdcl_W-state (update-weight-information C S) = distinct-cdcl_W-state S›
  ‹cdcl_W-conflicting (update-weight-information C S) = cdcl_W-conflicting S›
  ‹cdcl_W-learned-clause (update-weight-information C S) = cdcl_W-learned-clause S›
  unfolding no-strange-atm-def cdcl_W-M-level-inv-def distinct-cdcl_W-state-def cdcl_W-conflicting-def
    cdcl_W-learned-clause-alt-def cdcl_W-all-struct-inv-def by auto

lemma conflict-opt-cdcl_W-all-struct-inv:
  assumes ‹conflict-opt S T› and
    inv: ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state S)›
  shows ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state T)›
  using assms atms-of-conflicting-clss[of T] atms-of-conflicting-clss[of S]
  by (induction rule: conflict-opt.cases)
    (auto simp add: cdcl_W-restart-mset.no-strange-atm-def
        cdcl_W-restart-mset.cdcl_W-M-level-inv-def
        cdcl_W-restart-mset.distinct-cdcl_W-state-def cdcl_W-restart-mset.cdcl_W-conflicting-def
        cdcl_W-restart-mset.cdcl_W-learned-clause-alt-def cdcl_W-restart-mset.cdcl_W-all-struct-inv-def
        true-annots-true-cls-def-iff-negation-in-model
        in-negate-trial-iff cdcl_W-restart-mset-state cdcl_W-restart-mset.clauses-def
        distinct-mset-mset-conflicting-clss abs-state-def
      intro!: true-clss-cls-in)

lemma improve-cdcl_W-all-struct-inv:
  assumes ‹improvep S T› and
    inv: ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state S)›
  shows ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state T)›
  using assms atms-of-conflicting-clss[of T] atms-of-conflicting-clss[of S]
proof (induction rule: improvep.cases)
  case (improve-rule M' T)
  moreover have ‹all-decomposition-implies
     (set-mset (init-clss S) ∪ set-mset (conflicting-clss S) ∪ set-mset (learned-clss S))
     (get-all-ann-decomposition (trail S)) ⟹
    all-decomposition-implies
     (set-mset (init-clss S) ∪ set-mset (conflicting-clss (update-weight-information M' S)) ∪
      set-mset (learned-clss S))
     (get-all-ann-decomposition (trail S))›
      apply (rule all-decomposition-implies-mono)
      using improve-rule conflicting-clss-update-weight-information-mono[of S ‹trail S› M'] inv
      by (auto dest: multi-member-split)
   ultimately show ?case
      using conflicting-clss-update-weight-information-mono[of S ‹trail S› M']
      by (auto 6 2 simp add: cdcl_W-restart-mset.no-strange-atm-def
          cdcl_W-restart-mset.cdcl_W-M-level-inv-def
          cdcl_W-restart-mset.distinct-cdcl_W-state-def cdcl_W-restart-mset.cdcl_W-conflicting-def
          cdcl_W-restart-mset.cdcl_W-learned-clause-alt-def cdcl_W-restart-mset.cdcl_W-all-struct-inv-def
          true-annots-true-cls-def-iff-negation-in-model
          in-negate-trial-iff cdcl_W-restart-mset-state cdcl_W-restart-mset.clauses-def
```

image-Un distinct-mset-mset-conflicting-clss abs-state-def
simp del: *append-assoc*
dest: *no-dup-appendD consistent-interp-unionD*)
qed

$cdcl_W$*-restart-mset.*$cdcl_W$*-stgy-invariant* is too restrictive: $cdcl_W$*-restart-mset.no-smaller-confl* is needed but does not hold(at least, if cannot ensure that conflicts are found as soon as possible).

lemma *improve-no-smaller-conflict*:
assumes ‹*improvep S T*› **and**
‹*no-smaller-confl S*›
shows ‹*no-smaller-confl T*› **and** ‹*conflict-is-false-with-level T*›
using *assms* **apply** (*induction rule*: *improvep.induct*)
unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy-invariant-def*
by (*auto simp*: $cdcl_W$*-restart-mset-state no-smaller-confl-def* $cdcl_W$*-restart-mset.clauses-def*
exists-lit-max-level-in-negate-ann-lits)

lemma *conflict-opt-no-smaller-conflict*:
assumes ‹*conflict-opt S T*› **and**
‹*no-smaller-confl S*›
shows ‹*no-smaller-confl T*› **and** ‹*conflict-is-false-with-level T*›
using *assms* **by** (*induction rule*: *conflict-opt.induct*)
(*auto simp*: $cdcl_W$*-restart-mset-state no-smaller-confl-def* $cdcl_W$*-restart-mset.clauses-def*
exists-lit-max-level-in-negate-ann-lits $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy-invariant-def*)

fun *no-confl-prop-impr* **where**
‹*no-confl-prop-impr S* ⟷
no-step propagate S ∧ *no-step conflict S*›

We use a slighlty generalised form of backtrack to make conflict clause minimisation possible.

inductive *obacktrack* :: ‹*'st* ⇒ *'st* ⇒ *bool*› **for** *S* :: *'st* **where**
obacktrack-rule: ‹
conflicting S = *Some* (*add-mset L D*) ⟹
(*Decided K # M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail S*)) ⟹
get-level (*trail S*) *L* = *backtrack-lvl S* ⟹
get-level (*trail S*) *L* = *get-maximum-level* (*trail S*) (*add-mset L D'*) ⟹
get-maximum-level (*trail S*) *D'* ≡ *i* ⟹
get-level (*trail S*) *K* = *i* + *1* ⟹
D' ⊆# *D* ⟹
clauses S + *conflicting-clss S* ⊨*pm add-mset L D'* ⟹
T ∼ *cons-trail* (*Propagated L* (*add-mset L D'*))
(*reduce-trail-to M1*
(*add-learned-cls* (*add-mset L D'*)
(*update-conflicting None S*))) ⟹
obacktrack S T›

inductive-cases *obacktrackE*: ‹*obacktrack S T*›

inductive *cdcl-bnb-bj* :: ‹*'st* ⇒ *'st* ⇒ *bool*› **where**
skip: ‹*skip S S'* ⟹ *cdcl-bnb-bj S S'*› |
resolve: ‹*resolve S S'* ⟹ *cdcl-bnb-bj S S'*› |
backtrack: ‹*obacktrack S S'* ⟹ *cdcl-bnb-bj S S'*›

inductive-cases *cdcl-bnb-bjE*: ‹*cdcl-bnb-bj S T*›

inductive $ocdcl_W$*-o* :: ‹*'st* ⇒ *'st* ⇒ *bool*› **for** *S* :: *'st* **where**

decide: ‹*decide S S'* ⟹ *ocdcl*$_W$*-o S S'*› |
bj: ‹*cdcl-bnb-bj S S'* ⟹ *ocdcl*$_W$*-o S S'*›

inductive *cdcl-bnb* :: ‹*'st* ⇒ *'st* ⇒ *bool*› **for** *S* :: *'st* **where**
cdcl-conflict: ‹*conflict S S'* ⟹ *cdcl-bnb S S'*› |
cdcl-propagate: ‹*propagate S S'* ⟹ *cdcl-bnb S S'*› |
cdcl-improve: ‹*improvep S S'* ⟹ *cdcl-bnb S S'*› |
cdcl-conflict-opt: ‹*conflict-opt S S'* ⟹ *cdcl-bnb S S'*› |
cdcl-other': ‹*ocdcl*$_W$*-o S S'* ⟹ *cdcl-bnb S S'*›

inductive *cdcl-bnb-stgy* :: ‹*'st* ⇒ *'st* ⇒ *bool*› **for** *S* :: *'st* **where**
cdcl-bnb-conflict: ‹*conflict S S'* ⟹ *cdcl-bnb-stgy S S'*› |
cdcl-bnb-propagate: ‹*propagate S S'* ⟹ *cdcl-bnb-stgy S S'*› |
cdcl-bnb-improve: ‹*improvep S S'* ⟹ *cdcl-bnb-stgy S S'*› |
cdcl-bnb-conflict-opt: ‹*conflict-opt S S'* ⟹ *cdcl-bnb-stgy S S'*› |
cdcl-bnb-other': ‹*ocdcl*$_W$*-o S S'* ⟹ *no-confl-prop-impr S* ⟹ *cdcl-bnb-stgy S S'*›

lemma *ocdcl*$_W$*-o-induct*[*consumes 1*, *case-names decide skip resolve backtrack*]:
  fixes *S* :: *'st*
  assumes *cdcl*$_W$*-restart*: ‹*ocdcl*$_W$*-o S T*› **and**
    decideH: ⋀*L T*. *conflicting S* = *None* ⟹ *undefined-lit* (*trail S*) *L* ⟹
      atm-of L ∈ *atms-of-mm* (*init-clss S*) ⟹
      T ∼ *cons-trail* (*Decided L*) *S* ⟹
      P S T **and**
    skipH: ⋀*L C' M E T*.
      trail S = *Propagated L C'* # *M* ⟹
      conflicting S = *Some E* ⟹
      −*L* ∉# *E* ⟹ *E* ≠ {#} ⟹
      T ∼ *tl-trail S* ⟹
      P S T **and**
    resolveH: ⋀*L E M D T*.
      trail S = *Propagated L E* # *M* ⟹
      L ∈# *E* ⟹
      hd-trail S = *Propagated L E* ⟹
      conflicting S = *Some D* ⟹
      −*L* ∈# *D* ⟹
      get-maximum-level (*trail S*) ((*remove1-mset* (−*L*) *D*)) = *backtrack-lvl S* ⟹
      T ∼ *update-conflicting*
        (*Some* (*resolve-cls L D E*)) (*tl-trail S*) ⟹
      P S T **and**
    backtrackH: ⋀*L D K i M1 M2 T D'*.
      conflicting S = *Some* (*add-mset L D*) ⟹
      (*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail S*)) ⟹
      get-level (*trail S*) *L* = *backtrack-lvl S* ⟹
      get-level (*trail S*) *L* = *get-maximum-level* (*trail S*) (*add-mset L D'*) ⟹
      get-maximum-level (*trail S*) *D'* ≡ *i* ⟹
      get-level (*trail S*) *K* = *i+1* ⟹
      D' ⊆# *D* ⟹
      clauses S + *conflicting-clss S* ⊨*pm add-mset L D'* ⟹
      T ∼ *cons-trail* (*Propagated L* (*add-mset L D'*))
          (*reduce-trail-to M1*
            (*add-learned-cls* (*add-mset L D'*)
              (*update-conflicting None S*))) ⟹
       P S T
  shows ‹*P S T*›
  using *cdcl*$_W$*-restart* **apply** (*induct T rule*: *ocdcl*$_W$*-o.induct*)

subgoal using *assms(2)* **by** (*auto elim*: *decideE*; *fail*)
subgoal apply (*elim cdcl-bnb-bjE skipE resolveE obacktrackE*)
apply (*frule skipH*; *simp*; *fail*)
apply (*cases* ‹*trail S*›; *auto elim*!: *resolveE intro*!: *resolveH*; *fail*)
apply (*frule backtrackH*; *simp*; *fail*)
done
done

lemma *obacktrack-backtrackg*: ‹*obacktrack S T* ⟹ *backtrackg S T*›
unfolding *obacktrack.simps backtrackg.simps*
by *blast*

Pluging into normal CDCL

lemma *cdcl-bnb-no-more-init-clss*:
‹*cdcl-bnb S S′* ⟹ *init-clss S* = *init-clss S′*›
by (*induction rule*: *cdcl-bnb.cases*)
(*auto simp*: *improvep.simps conflict.simps propagate.simps*
conflict-opt.simps $ocdcl_W$*-o.simps obacktrack.simps skip.simps resolve.simps cdcl-bnb-bj.simps*
decide.simps)

lemma *rtranclp-cdcl-bnb-no-more-init-clss*:
‹$cdcl\text{-}bnb^{**}$ *S S′* ⟹ *init-clss S* = *init-clss S′*›
by (*induction rule*: *rtranclp-induct*)
(*auto dest*: *cdcl-bnb-no-more-init-clss*)

lemma *conflict-opt-conflict*:
‹*conflict-opt S T* ⟹ $cdcl_W$*-restart-mset.conflict* (*abs-state S*) (*abs-state T*)›
by (*induction rule*: *conflict-opt.cases*)
(*auto intro*!: $cdcl_W$*-restart-mset.conflict-rule*[*of* - ‹*negate-ann-lits* (*trail S*)›]
simp: $cdcl_W$*-restart-mset.clauses-def* $cdcl_W$*-restart-mset-state*
true-annots-true-cls-def-iff-negation-in-model abs-state-def
in-negate-trial-iff)

lemma *conflict-conflict*:
‹*conflict S T* ⟹ $cdcl_W$*-restart-mset.conflict* (*abs-state S*) (*abs-state T*)›
by (*induction rule*: *conflict.cases*)
(*auto intro*!: $cdcl_W$*-restart-mset.conflict-rule*
simp: *clauses-def* $cdcl_W$*-restart-mset.clauses-def* $cdcl_W$*-restart-mset-state*
true-annots-true-cls-def-iff-negation-in-model abs-state-def
in-negate-trial-iff)

lemma *propagate-propagate*:
‹*propagate S T* ⟹ $cdcl_W$*-restart-mset.propagate* (*abs-state S*) (*abs-state T*)›
by (*induction rule*: *propagate.cases*)
(*auto intro*!: $cdcl_W$*-restart-mset.propagate-rule*
simp: *clauses-def* $cdcl_W$*-restart-mset.clauses-def* $cdcl_W$*-restart-mset-state*
true-annots-true-cls-def-iff-negation-in-model abs-state-def
in-negate-trial-iff)

lemma *decide-decide*:
‹*decide S T* ⟹ $cdcl_W$*-restart-mset.decide* (*abs-state S*) (*abs-state T*)›
by (*induction rule*: *decide.cases*)
(*auto intro*!: $cdcl_W$*-restart-mset.decide-rule*
simp: *clauses-def* $cdcl_W$*-restart-mset.clauses-def* $cdcl_W$*-restart-mset-state*

true-annots-true-cls-def-iff-negation-in-model abs-state-def
in-negate-trial-iff)

lemma *skip-skip*:
‹*skip S T* ⟹ *cdcl*$_W$*-restart-mset.skip* (*abs-state S*) (*abs-state T*)›
by (*induction rule*: *skip.cases*)
(*auto intro*!: *cdcl*$_W$*-restart-mset.skip-rule*
simp: *clauses-def cdcl*$_W$*-restart-mset.clauses-def cdcl*$_W$*-restart-mset-state*
true-annots-true-cls-def-iff-negation-in-model abs-state-def
in-negate-trial-iff)

lemma *resolve-resolve*:
‹*resolve S T* ⟹ *cdcl*$_W$*-restart-mset.resolve* (*abs-state S*) (*abs-state T*)›
by (*induction rule*: *resolve.cases*)
(*auto intro*!: *cdcl*$_W$*-restart-mset.resolve-rule*
simp: *clauses-def cdcl*$_W$*-restart-mset.clauses-def cdcl*$_W$*-restart-mset-state*
true-annots-true-cls-def-iff-negation-in-model abs-state-def
in-negate-trial-iff)

lemma *backtrack-backtrack*:
‹*obacktrack S T* ⟹ *cdcl*$_W$*-restart-mset.backtrack* (*abs-state S*) (*abs-state T*)›
proof (*induction rule*: *obacktrack.cases*)
case (*obacktrack-rule L D K M1 M2 D′ i T*)
have *H*: ‹*set-mset* (*init-clss S*) ∪ *set-mset* (*learned-clss S*)
⊆ *set-mset* (*init-clss S*) ∪ *set-mset* (*conflicting-clss S*) ∪ *set-mset* (*learned-clss S*)›
by *auto*
have [*simp*]: ‹*cdcl*$_W$*-restart-mset.reduce-trail-to M1*
(*trail S*, *init-clss S* + *conflicting-clss S*, *add-mset D* (*learned-clss S*), *None*) =
(*M1*, *init-clss S* + *conflicting-clss S*, *add-mset D* (*learned-clss S*), *None*)› **for** *D*
using *obacktrack-rule* **by** (*auto simp add*: *cdcl*$_W$*-restart-mset-reduce-trail-to*
cdcl$_W$*-restart-mset-state*)
show *?case*
using *obacktrack-rule*
by (*auto intro*!: *cdcl*$_W$*-restart-mset.backtrack.intros*
simp: *cdcl*$_W$*-restart-mset-state abs-state-def clauses-def cdcl*$_W$*-restart-mset.clauses-def*
ac-simps)
qed

lemma *ocdcl*$_W$*-o-all-rules-induct*[*consumes 1*, *case-names decide backtrack skip resolve*]:
fixes *S T* :: *′st*
assumes
‹*ocdcl*$_W$*-o S T*› **and**
‹⋀*T*. *decide S T* ⟹ *P S T*› **and**
‹⋀*T*. *obacktrack S T* ⟹ *P S T*› **and**
‹⋀*T*. *skip S T* ⟹ *P S T*› **and**
‹⋀*T*. *resolve S T* ⟹ *P S T*›
shows ‹*P S T*›
using *assms* **by** (*induct T rule*: *ocdcl*$_W$*-o.induct*) (*auto simp*: *cdcl-bnb-bj.simps*)

lemma *cdcl*$_W$*-o-cdcl*$_W$*-o*:
‹*ocdcl*$_W$*-o S S′* ⟹ *cdcl*$_W$*-restart-mset.cdcl*$_W$*-o* (*abs-state S*) (*abs-state S′*)›
apply (*induction rule*: *ocdcl*$_W$*-o-all-rules-induct*)
apply (*simp add*: *cdcl*$_W$*-restart-mset.cdcl*$_W$*-o.simps decide-decide*; *fail*)
apply (*blast dest*: *backtrack-backtrack*)
apply (*blast dest*: *skip-skip*)
by (*blast dest*: *resolve-resolve*)

```
lemma cdcl-bnb-stgy-all-struct-inv:
  assumes ‹cdcl-bnb S T› and ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state S)›
  shows ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state T)›
  using assms
proof (induction rule: cdcl-bnb.cases)
  case (cdcl-conflict S')
  then show ?case
    by (blast dest: conflict-conflict cdcl_W-restart-mset.cdcl_W-stgy.intros
      intro: cdcl_W-restart-mset.cdcl_W-stgy-cdcl_W-all-struct-inv)
next
  case (cdcl-propagate S')
  then show ?case
    by (blast dest: propagate-propagate cdcl_W-restart-mset.cdcl_W-stgy.intros
      intro: cdcl_W-restart-mset.cdcl_W-stgy-cdcl_W-all-struct-inv)
next
  case (cdcl-improve S')
  then show ?case
    using improve-cdcl_W-all-struct-inv by blast
next
  case (cdcl-conflict-opt S')
  then show ?case
    using conflict-opt-cdcl_W-all-struct-inv by blast
next
  case (cdcl-other' S')
  then show ?case
    by (meson cdcl_W-restart-mset.cdcl_W-all-struct-inv-inv cdcl_W-restart-mset.other cdcl_W-o-cdcl_W-o)
qed

lemma rtranclp-cdcl-bnb-stgy-all-struct-inv:
  assumes ‹cdcl-bnb** S T› and ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state S)›
  shows ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state T)›
  using assms by induction (auto dest: cdcl-bnb-stgy-all-struct-inv)


lemma cdcl-bnb-stgy-cdcl_W-or-improve:
  assumes ‹cdcl-bnb S T› and ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state S)›
  shows ‹(λS T. cdcl_W-restart-mset.cdcl_W (abs-state S) (abs-state T) ∨ improvep S T) S T›
  using assms
  apply (induction rule: cdcl-bnb.cases)
  apply (auto dest!: propagate-propagate conflict-conflict
    intro: cdcl_W-restart-mset.cdcl_W.intros simp add: cdcl_W-restart-mset.W-conflict conflict-opt-conflict
      cdcl_W-o-cdcl_W-o cdcl_W-restart-mset.W-other)
  done


lemma rtranclp-cdcl-bnb-stgy-cdcl_W-or-improve:
  assumes ‹rtranclp cdcl-bnb S T› and ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state S)›
  shows ‹(λS T. cdcl_W-restart-mset.cdcl_W (abs-state S) (abs-state T) ∨ improvep S T)** S T›
  using assms
  apply (induction rule: rtranclp-induct)
  subgoal by auto
  subgoal for T U
    using cdcl-bnb-stgy-cdcl_W-or-improve[of T U] rtranclp-cdcl-bnb-stgy-all-struct-inv[of S T]
    by (smt rtranclp-unfold tranclp-unfold-end)
  done
```

lemma *eq-diff-subset-iff*: ‹$A = B + (A - B) \longleftrightarrow B \subseteq\# A$›
by (*metis mset-subset-eq-add-left subset-mset.add-diff-inverse*)

lemma *cdcl-bnb-conflicting-clss-mono*:
‹*cdcl-bnb S T* $\Longrightarrow$ *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*) $\Longrightarrow$
conflicting-clss S $\subseteq\#$ *conflicting-clss T*›
by (*auto simp*: *cdcl-bnb.simps ocdcl*$_W$*-o.simps improvep.simps cdcl-bnb-bj.simps*
obacktrack.simps conflict-opt.simps conflicting-clss-update-weight-information-mono elim!: *rulesE*)

lemma *cdcl-or-improve-cdclD*:
assumes ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)› **and**
‹*cdcl-bnb S T*›
shows ‹$\exists N$.
cdcl$_W$*-restart-mset.cdcl*$_W^{**}$ (*trail S*, *init-clss S* + *N*, *learned-clss S*, *conflicting S*) (*abs-state T*) $\wedge$
CDCL-W-Abstract-State.init-clss (*abs-state T*) = *init-clss S* + *N*›
proof −
have *inv-T*: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state T*)›
using *assms*(*1*) *assms*(*2*) *cdcl-bnb-stgy-all-struct-inv* **by** *blast*
consider
‹*improvep S T*› |
‹*cdcl*$_W$*-restart-mset.cdcl*$_W$ (*abs-state S*) (*abs-state T*)›
using *cdcl-bnb-stgy-cdcl*$_W$*-or-improve*[*of S T*] *assms* **by** *blast*
then show *?thesis*
proof *cases*
case *1*
then show *?thesis*
using *assms cdcl-bnb-stgy-cdcl*$_W$*-or-improve*[*of S T*]
unfolding *abs-state-def cdcl-bnb-no-more-init-clss*[*of S T*, *OF assms*(*2*)]
by (*auto simp*: *improvep.simps cdcl*$_W$*-restart-mset-state eq-diff-subset-iff*)
next
case *2*
let *?S′* = ‹(*trail S*, *init-clss S* + (*conflicting-clss S*) + (*conflicting-clss T* − *conflicting-clss S*),
learned-clss S, *conflicting S*)›
let *?S″* = ‹(*trail S*, *init-clss S* + *conflicting-clss T*, *learned-clss S*, *conflicting S*)›
let *?T′* = ‹(*trail T*, *init-clss T* + (*conflicting-clss T*) + (*conflicting-clss T* − *conflicting-clss S*),
learned-clss T, *conflicting T*)›
have *subs*: ‹*conflicting-clss S* $\subseteq\#$ *conflicting-clss T*›
using *cdcl-bnb-conflicting-clss-mono*[*of S T*] *assms* **by** *fast*
then have *H*[*simp*]: ‹*set-mset* (*conflicting-clss T* + (*conflicting-clss T* −
conflicting-clss S)) = *set-mset* (*conflicting-clss T*)›
apply (*auto simp flip*: *multiset-diff-union-assoc*[*OF subs*])
apply (*subst* (*asm*) *multiset-diff-union-assoc*[*OF subs*] *set-mset-union*)+
apply (*auto dest*: *in-diffD*)
apply (*subst multiset-diff-union-assoc*[*OF subs*] *set-mset-union*)+
apply (*auto dest*: *in-diffD*)
done
have [*simp*]: ‹*set-mset* (*init-clss T* + *conflicting-clss T* + *conflicting-clss T* −
conflicting-clss S) = *set-mset* (*init-clss T* + *conflicting-clss T*)›
by (*subst multiset-diff-union-assoc*, (*rule subs*))
(*simp only*: *H ac-simps*, *subst set-mset-union*, *subst H*, *simp*)
have ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv ?T′*›
by (*rule cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-clauses-cong*[*OF inv-T*])
(*auto simp*: *cdcl*$_W$*-restart-mset-state eq-diff-subset-iff abs-state-def subs*)
then have ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$ *?S′ ?T′*›

using *2* $cdcl_W$-*restart-mset*.$cdcl_W$-*enlarge-clauses*[*of* ‹*abs-state S*› ‹*abs-state T*› *?S′* ‹*conflicting-clss T* − *conflicting-clss S*› ‹{#}›]
by (*auto simp*: $cdcl_W$-*restart-mset-state abs-state-def subs*)
then have ‹$cdcl_W$-*restart-mset*.$cdcl_W$ *?S″* (*abs-state T*)›
using $cdcl_W$-*restart-mset*.$cdcl_W$-*clauses-cong*[*of* ‹*?S′*› *?T′ ?S″*]
$cdcl_W$-*restart-mset*.$cdcl_W$-*learnel-clss-mono*[*of* ‹*?S′*› *?T′*]
$cdcl_W$-*restart-mset*.$cdcl_W$-*restart-init-clss*[*OF* $cdcl_W$-*restart-mset*.$cdcl_W$-$cdcl_W$-*restart*, *of* ‹*?S′*› *?T′*]
unfolding *abs-state-def cdcl-bnb-no-more-init-clss*[*of S T*, *OF assms*(*2*)]
by (*auto simp*: $cdcl_W$-*restart-mset-state abs-state-def subs*)

then show *?thesis*
by (*auto intro*!: *exI*[*of* - ‹*conflicting-clss T*›] *simp*: *abs-state-def init-clss.simps*
cdcl-bnb-no-more-init-clss[*of S T*, *OF assms*(*2*)])
qed
qed

lemma *rtranclp-cdcl-or-improve-cdclD*:
assumes ‹$cdcl_W$-*restart-mset*.$cdcl_W$-*all-struct-inv* (*abs-state S*)› **and**
‹*cdcl-bnb*** *S T*›
shows ‹∃ *N*.
$cdcl_W$-*restart-mset*.$cdcl_W$** (*trail S*, *init-clss S* + *N*, *learned-clss S*, *conflicting S*) (*abs-state T*) ∧
CDCL-W-Abstract-State.init-clss (*abs-state T*) = *init-clss S* + *N*›
using *assms*(*2*,*1*)
proof (*induction rule*: *rtranclp-induct*)
case *base*
then show *?case* **by** (*auto intro*!: *exI*[*of* - ‹{#}›] *simp*: *abs-state-def init-clss.simps*)
next
case (*step T U*)
then obtain *N* **where**
st: ‹$cdcl_W$-*restart-mset*.$cdcl_W$** (*trail S*, *init-clss S* + *N*, *learned-clss S*, *conflicting S*)
(*abs-state T*)› **and**
eq: ‹*CDCL-W-Abstract-State.init-clss* (*abs-state T*) = *init-clss S* + *N*›
by *auto*
obtain *N′* **where**
st′: ‹$cdcl_W$-*restart-mset*.$cdcl_W$** (*trail T*, *init-clss T* + *N′*, *learned-clss T*, *conflicting T*)
(*abs-state U*)› **and**
eq′: ‹*CDCL-W-Abstract-State.init-clss* (*abs-state U*) = *init-clss T* + *N′*›
using *cdcl-or-improve-cdclD*[*of T U*] *rtranclp-cdcl-bnb-stgy-all-struct-inv*[*of S T*] *step*
by (*auto simp*: $cdcl_W$-*restart-mset-state*)
have *inv-T*: ‹$cdcl_W$-*restart-mset*.$cdcl_W$-*all-struct-inv* (*abs-state T*)›
using *rtranclp-cdcl-bnb-stgy-all-struct-inv step.hyps*(*1*) *step.prems* **by** *blast*
have [*simp*]: ‹*init-clss S* = *init-clss T*› ‹*init-clss T* = *init-clss U*›
using *rtranclp-cdcl-bnb-no-more-init-clss*[*OF step*(*1*)] *cdcl-bnb-no-more-init-clss*[*OF step*(*2*)]
by *fast*+
then have ‹*N* ⊆# *N′*›
using *eq eq′ inv-T cdcl-bnb-conflicting-clss-mono*[*of T U*] *step*
by (*auto simp*: *abs-state-def init-clss.simps*)

let *?S* = ‹(*trail S*, *init-clss S* + *N*, *learned-clss S*, *conflicting S*)›
let *?S′* = ‹(*trail S*, (*init-clss S* + *N*) + (*N′* − *N*), *learned-clss S*, *conflicting S*)›
let *?T′* = ‹(*trail T*, *init-clss T* + (*conflicting-clss T*) + (*N′* − *N*), *learned-clss T*, *conflicting T*)›
have ‹$cdcl_W$-*restart-mset*.$cdcl_W$** *?S′ ?T′*›
using *st eq* $cdcl_W$-*restart-mset*.*rtranclp*-$cdcl_W$-*enlarge-clauses*[*of ?S′ ?S* ‹*N′* − *N*› ‹{#}› ‹*abs-state T*›]
by (*auto simp*: $cdcl_W$-*restart-mset-state abs-state-def*)
moreover have ‹*init-clss T* + (*conflicting-clss T*) + (*N′* − *N*) = *init-clss T* + *N′*›

using *eq eq'* ‹*N* ⊆# *N'*›
by (*auto simp*: *abs-state-def init-clss.simps*)

ultimately have
‹*cdcl*$_W$*-restart-mset.cdcl*$_W$** (*trail S*, *init-clss S* + *N'*, *learned-clss S*, *conflicting S*)
(*abs-state U*)›
using *eq' st'* ‹*N* ⊆# *N'*› **unfolding** *abs-state-def*
by *auto*
then show *?case*
using *eq' st'* **by** (*auto intro*!: *exI*[*of* - *N'*])
qed

definition *cdcl-bnb-struct-invs* :: ‹*'st* ⇒ *bool*› **where**
‹*cdcl-bnb-struct-invs S* ⟷
atms-of-mm (*conflicting-clss S*) ⊆ *atms-of-mm* (*init-clss S*)›

lemma *cdcl-bnb-cdcl-bnb-struct-invs*:
‹*cdcl-bnb S T* ⟹ *cdcl-bnb-struct-invs S* ⟹ *cdcl-bnb-struct-invs T*›
using *atms-of-conflicting-clss*[*of* ‹*update-weight-information* - *S*›] **apply** −
by (*induction rule*: *cdcl-bnb.induct*)
(*force simp*: *improvep.simps conflict.simps propagate.simps*
conflict-opt.simps ocdcl$_W$*-o.simps obacktrack.simps skip.simps resolve.simps*
cdcl-bnb-bj.simps decide.simps cdcl-bnb-struct-invs-def)+

lemma *rtranclp-cdcl-bnb-cdcl-bnb-struct-invs*:
‹*cdcl-bnb*** *S T* ⟹ *cdcl-bnb-struct-invs S* ⟹ *cdcl-bnb-struct-invs T*›
by (*induction rule*: *rtranclp-induct*) (*auto dest*: *cdcl-bnb-cdcl-bnb-struct-invs*)

lemma *cdcl-bnb-stgy-cdcl-bnb*: ‹*cdcl-bnb-stgy S T* ⟹ *cdcl-bnb S T*›
by (*auto simp*: *cdcl-bnb-stgy.simps intro*: *cdcl-bnb.intros*)

lemma *rtranclp-cdcl-bnb-stgy-cdcl-bnb*: ‹*cdcl-bnb-stgy*** *S T* ⟹ *cdcl-bnb*** *S T*›
by (*induction rule*: *rtranclp-induct*)
(*auto dest*: *cdcl-bnb-stgy-cdcl-bnb*)

The following does *not* hold, because we cannot guarantee the absence of conflict of smaller level after *improve* and *conflict-opt*.

lemma *cdcl-bnb-all-stgy-inv*:
assumes ‹*cdcl-bnb S T*› **and** ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)› **and**
‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-stgy-invariant* (*abs-state S*)›
shows ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-stgy-invariant* (*abs-state T*)›
oops

lemma *skip-conflict-is-false-with-level*:
assumes ‹*skip S T*› **and**
struct-inv: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)› **and**
confl-inv:‹*conflict-is-false-with-level S*›
shows ‹*conflict-is-false-with-level T*›
using *assms*
proof *induction*
case (*skip-rule L C' M D T*) **note** *tr-S* = *this*(*1*) **and** *D* = *this*(*2*) **and** *T* = *this*(*5*)
have *conflicting*: ‹*cdcl*$_W$*-conflicting S*› **and**
lev: ‹*cdcl*$_W$*-M-level-inv S*›
using *struct-inv* **unfolding** *cdcl*$_W$*-conflicting-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*
cdcl$_W$*-M-level-inv-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-conflicting-def*
cdcl$_W$*-restart-mset.cdcl*$_W$*-M-level-inv-def*

```
    by (auto simp: abs-state-def cdcl_W-restart-mset-state)
  obtain La where
    ‹La ∈# D› and
    ‹get-level (Propagated L C' # M) La = backtrack-lvl S›
    using skip-rule confl-inv by auto
  moreover {
    have ‹atm-of La ≠ atm-of L›
    proof (rule ccontr)
      assume ‹¬ ?thesis›
      then have La: ‹La = L› using ‹La ∈# D› ‹− L ∉# D›
        by (auto simp add: atm-of-eq-atm-of)
      have ‹Propagated L C' # M ⊨as CNot D›
        using conflicting tr-S D unfolding cdcl_W-conflicting-def by auto
      then have ‹−L ∈ lits-of-l M›
        using ‹La ∈# D› in-CNot-implies-uminus(2)[of L D ‹Propagated L C' # M›] unfolding La
        by auto
      then show False using lev tr-S unfolding cdcl_W-M-level-inv-def consistent-interp-def by auto
    qed
    then have ‹get-level (Propagated L C' # M) La = get-level M La› by auto
  }
  ultimately show ?case using D tr-S T by auto
qed

lemma propagate-conflict-is-false-with-level:
  assumes ‹propagate S T› and
    struct-inv: ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state S)› and
    confl-inv:‹conflict-is-false-with-level S›
  shows ‹conflict-is-false-with-level T›
  using assms by (induction rule: propagate.induct) auto

lemma cdcl_W-o-conflict-is-false-with-level:
  assumes ‹cdcl_W-o S T› and
    struct-inv: ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state S)› and
    confl-inv: ‹conflict-is-false-with-level S›
  shows ‹conflict-is-false-with-level T›
  apply (rule cdcl_W-o-conflict-is-false-with-level-inv[of S T])
  subgoal using assms by auto
  subgoal using struct-inv unfolding cdcl_W-restart-mset.cdcl_W-all-struct-inv-def
      cdcl_W-M-level-inv-def cdcl_W-restart-mset.cdcl_W-M-level-inv-def
    by (auto simp: abs-state-def cdcl_W-restart-mset-state)
  subgoal using assms by auto
  subgoal using struct-inv unfolding distinct-cdcl_W-state-def
      cdcl_W-restart-mset.cdcl_W-all-struct-inv-def cdcl_W-restart-mset.distinct-cdcl_W-state-def
    by (auto simp: abs-state-def cdcl_W-restart-mset-state)
  subgoal using struct-inv unfolding cdcl_W-conflicting-def
      cdcl_W-restart-mset.cdcl_W-all-struct-inv-def cdcl_W-restart-mset.cdcl_W-conflicting-def
    by (auto simp: abs-state-def cdcl_W-restart-mset-state)
  done

lemma cdcl_W-o-no-smaller-confl:
  assumes ‹cdcl_W-o S T› and
    struct-inv: ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state S)› and
    confl-inv: ‹no-smaller-confl S› and
    lev: ‹conflict-is-false-with-level S› and
    n-s: ‹no-confl-prop-impr S›
  shows ‹no-smaller-confl T›
```

apply (*rule* cdcl$_W$-o-no-smaller-confl-inv[*of S T*])
subgoal using *assms* **by** (*auto dest*!:cdcl$_W$-o-cdcl$_W$-o)[]
subgoal using *n-s* **by** *auto*
subgoal using *struct-inv* **unfolding** cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv-def
cdcl$_W$-M-level-inv-def cdcl$_W$-restart-mset.cdcl$_W$-M-level-inv-def
by (*auto simp*: abs-state-def cdcl$_W$-restart-mset-state)
subgoal using *lev* **by** *fast*
subgoal using *confl-inv* **unfolding** distinct-cdcl$_W$-state-def
cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv-def cdcl$_W$-restart-mset.distinct-cdcl$_W$-state-def
cdcl$_W$-restart-mset.no-smaller-confl-def
by (*auto simp*: abs-state-def cdcl$_W$-restart-mset-state clauses-def)
done

declare cdcl$_W$-restart-mset.conflict-is-false-with-level-def [*simp del*]

lemma *improve-conflict-is-false-with-level*:
assumes ‹*improvep S T*› **and** ‹*conflict-is-false-with-level S*›
shows ‹*conflict-is-false-with-level T*›
using *assms*
by *induction* (*auto simp*: cdcl$_W$-restart-mset.conflict-is-false-with-level-def
abs-state-def cdcl$_W$-restart-mset-state in-negate-trial-iff Bex-def negate-ann-lits-empty-iff
intro!: *exI*[*of* - ‹−*lit-of* (*hd M*)›])

declare *conflict-is-false-with-level-def*[*simp del*]

lemma cdcl$_W$-M-level-inv-cdcl$_W$-M-level-inv[*iff*]:
‹cdcl$_W$-restart-mset.cdcl$_W$-M-level-inv (*abs-state S*) = cdcl$_W$-M-level-inv *S*›
by (*auto simp*: cdcl$_W$-restart-mset.cdcl$_W$-M-level-inv-def
cdcl$_W$-M-level-inv-def cdcl$_W$-restart-mset-state)

lemma *obacktrack-state-eq-compatible*:
assumes
bt: ‹*obacktrack S T*› **and**
SS′: ‹*S* ∼ *S′*› **and**
TT′: ‹*T* ∼ *T′*›
shows ‹*obacktrack S′ T′*›
proof −
obtain *D L K i M1 M2 D′* **where**
conf: ‹*conflicting S* = *Some* (*add-mset L D*)› **and**
decomp: ‹(*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail S*))› **and**
lev: ‹*get-level* (*trail S*) *L* = *backtrack-lvl S*› **and**
max: ‹*get-level* (*trail S*) *L* = *get-maximum-level* (*trail S*) (*add-mset L D′*)› **and**
max-D: ‹*get-maximum-level* (*trail S*) *D′* ≡ *i*› **and**
lev-K: ‹*get-level* (*trail S*) *K* = *Suc i*› **and**
D′-D: ‹*D′* ⊆# *D*› **and**
NU-DL: ‹*clauses S* + *conflicting-clss S* ⊨pm *add-mset L D′*› **and**
T: *T* ∼ *cons-trail* (*Propagated L* (*add-mset L D′*))
(*reduce-trail-to M1*
(*add-learned-cls* (*add-mset L D′*)
(*update-conflicting None S*)))
using *bt* **by** (*elim obacktrackE*) *force*
let *?D* = ‹*add-mset L D*›
let *?D′* = ‹*add-mset L D′*›
have *D′*: ‹*conflicting S′* = *Some ?D*›
using *SS′ conf* **by** (*cases* ‹*conflicting S′*›) *auto*

```
  have T'-S: T' ∼ cons-trail (Propagated L ?D')
     (reduce-trail-to M1 (add-learned-cls ?D'
     (update-conflicting None S)))
    using T TT' state-eq-sym state-eq-trans by blast
  have T': T' ∼ cons-trail (Propagated L ?D')
     (reduce-trail-to M1 (add-learned-cls ?D'
     (update-conflicting None S')))
    apply (rule state-eq-trans[OF T'-S])
    by (auto simp: cons-trail-state-eq reduce-trail-to-state-eq add-learned-cls-state-eq
        update-conflicting-state-eq SS')
  show ?thesis
    apply (rule obacktrack-rule[of - L D K M1 M2 D' i])
    subgoal by (rule D')
    subgoal using TT' decomp SS' by auto
    subgoal using lev TT'  SS' by auto
    subgoal using max TT' SS' by auto
    subgoal using max-D TT' SS' by auto
    subgoal using lev-K TT' SS' by auto
    subgoal by (rule D'-D)
    subgoal using NU-DL TT' SS' by auto
    subgoal by (rule T')
    done
qed

lemma ocdcl_W-o-no-smaller-confl-inv:
  fixes S S' :: ‹'st›
  assumes
    ‹ocdcl_W-o S S'› and
    n-s: ‹no-step conflict S› and
    lev: ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state S)› and
    max-lev: ‹conflict-is-false-with-level S› and
    smaller: ‹no-smaller-confl S›
  shows ‹no-smaller-confl S'›
  using assms(1,2) unfolding no-smaller-confl-def
proof (induct rule: ocdcl_W-o-induct)
  case (decide L T) note confl = this(1) and undef = this(2) and T = this(4)
  have [simp]: ‹clauses T = clauses S›
    using T undef by auto
  show ?case
  proof (intro allI impI)
    fix M'' K M' Da
    assume ‹trail T = M'' @ Decided K # M'› and D: ‹Da ∈# local.clauses T›
    then have trail S = tl M'' @ Decided K # M'
        ∨ (M'' = [] ∧ Decided K # M' = Decided L # trail S)
      using T undef by (cases M'') auto
    moreover {
      assume ‹trail S = tl M'' @ Decided K # M'›
      then have ‹¬M' ⊨as CNot Da›
        using D T undef confl smaller unfolding no-smaller-confl-def smaller by fastforce
    }
    moreover {
      assume ‹Decided K # M' = Decided L # trail S›
      then have ‹¬M' ⊨as CNot Da› using smaller D confl T n-s by (auto simp: conflict.simps)
    }
    ultimately show ‹¬M' ⊨as CNot Da› by fast
  qed
```

```
next
  case resolve
  then show ?case using smaller max-lev unfolding no-smaller-confl-def by auto
next
  case skip
  then show ?case using smaller max-lev unfolding no-smaller-confl-def by auto
next
  case (backtrack L D K i M1 M2 T D') note confl = this(1) and decomp = this(2) and
    T = this(9)
  obtain c where M: ‹trail S = c @ M2 @ Decided K # M1›
    using decomp by auto

  show ?case
  proof (intro allI impI)
    fix M ia K' M' Da
    assume ‹trail T = M' @ Decided K' # M›
    then have ‹M1 = tl M' @ Decided K' # M›
      using T decomp lev by (cases M') (auto simp: cdcl_W-M-level-inv-decomp)
    let ?D' = ‹add-mset L D'›
    let ?S' = (cons-trail (Propagated L ?D')
                (reduce-trail-to M1 (add-learned-cls ?D' (update-conflicting None S))))
    assume D: ‹Da ∈# clauses T›
    moreover{
      assume ‹Da ∈# clauses S›
      then have ‹¬M ⊨as CNot Da› using ‹M1 = tl M' @ Decided K' # M› M confl smaller
        unfolding no-smaller-confl-def by auto
    }
    moreover {
      assume Da: ‹Da = add-mset L D'›
      have ‹¬M ⊨as CNot Da›
      proof (rule ccontr)
        assume ‹¬ ?thesis›
        then have ‹−L ∈ lits-of-l M›
          unfolding Da by (simp add: in-CNot-implies-uminus(2))
        then have ‹−L ∈ lits-of-l (Propagated L D # M1)›
          using UnI2 ‹M1 = tl M' @ Decided K' # M›
          by auto
        moreover {
          have ‹obacktrack S ?S'›
            using obacktrack-rule[OF backtrack.hyps(1−8) T] obacktrack-state-eq-compatible[of S T S] T
            by force
          then have ‹cdcl-bnb S ?S'›
            by (auto dest!: cdcl-bnb-bj.intros ocdcl_W-o.intros intro: cdcl-bnb.intros)
          then have ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state ?S')›
            using cdcl-bnb-stgy-all-struct-inv[of S, OF - lev] by fast
          then have ‹cdcl_W-restart-mset.cdcl_W-M-level-inv (abs-state ?S')›
            by (auto simp: cdcl_W-restart-mset.cdcl_W-all-struct-inv-def)
          then have ‹no-dup (Propagated L D # M1)›
            using decomp lev unfolding cdcl_W-restart-mset.cdcl_W-M-level-inv-def by auto
        }
        ultimately show False
          using Decided-Propagated-in-iff-in-lits-of-l defined-lit-map
          by (auto simp: no-dup-def)
      qed
    }
    ultimately show ‹¬M ⊨as CNot Da›
```

using *T decomp lev* **unfolding** *cdcl*$_W$*-M-level-inv-def* **by** *fastforce*
qed
qed

lemma *cdcl-bnb-stgy-no-smaller-confl*:
assumes ‹*cdcl-bnb-stgy S T*› **and**
‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)› **and**
‹*no-smaller-confl S*› **and**
‹*conflict-is-false-with-level S*›
shows ‹*no-smaller-confl T*›
using *assms*
proof (*induction rule*: *cdcl-bnb-stgy.cases*)
case (*cdcl-bnb-other′ S′*)
show *?case*
by (*rule ocdcl*$_W$*-o-no-smaller-confl-inv*)
(*use cdcl-bnb-other′* **in** ‹*auto simp*: *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*›)
qed (*auto intro*: *conflict-no-smaller-confl-inv propagate-no-smaller-confl-inv*;
auto simp: *no-smaller-confl-def improvep.simps conflict-opt.simps*)+

lemma *ocdcl*$_W$*-o-conflict-is-false-with-level-inv*:
assumes
‹*ocdcl*$_W$*-o S S′*› **and**
lev: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)› **and**
confl-inv: ‹*conflict-is-false-with-level S*›
shows ‹*conflict-is-false-with-level S′*›
using *assms*(*1*,*2*)
proof (*induct rule*: *ocdcl*$_W$*-o-induct*)
case (*resolve L C M D T*) **note** *tr-S* = *this*(*1*) **and** *confl* = *this*(*4*) **and** *LD* = *this*(*5*) **and** *T* = *this*(*7*)

have ‹*resolve S T*›
using *resolve.intros*[*of S L C D T*] *resolve*
by *auto*
then have ‹*cdcl*$_W$*-restart-mset.resolve* (*abs-state S*) (*abs-state T*)›
by (*simp add*: *resolve-resolve*)
moreover have ‹*cdcl*$_W$*-restart-mset.conflict-is-false-with-level* (*abs-state S*)›
using *confl-inv*
by (*auto simp*: *cdcl*$_W$*-restart-mset.conflict-is-false-with-level-def*
conflict-is-false-with-level-def abs-state-def cdcl$_W$*-restart-mset-state*)
ultimately have ‹*cdcl*$_W$*-restart-mset.conflict-is-false-with-level* (*abs-state T*)›
using *cdcl*$_W$*-restart-mset.cdcl*$_W$*-o-conflict-is-false-with-level-inv*[*of* ‹*abs-state S*› ‹*abs-state T*›]
lev confl-inv **unfolding** *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*
by (*auto dest*!: *cdcl*$_W$*-restart-mset.cdcl*$_W$*-o.intros*
cdcl$_W$*-restart-mset.cdcl*$_W$*-bj.intros*)
then show ‹*?case*›
by (*auto simp*: *cdcl*$_W$*-restart-mset.conflict-is-false-with-level-def*
conflict-is-false-with-level-def abs-state-def cdcl$_W$*-restart-mset-state*)
next
case (*skip L C′ M D T*) **note** *tr-S* = *this*(*1*) **and** *D* = *this*(*2*) **and** *T* = *this*(*5*)
have ‹*cdcl*$_W$*-restart-mset.skip* (*abs-state S*) (*abs-state T*)›
using *skip.intros*[*of S L C′ M D T*] *skip* **by** (*simp add*: *skip-skip*)
moreover have ‹*cdcl*$_W$*-restart-mset.conflict-is-false-with-level* (*abs-state S*)›
using *confl-inv*
by (*auto simp*: *cdcl*$_W$*-restart-mset.conflict-is-false-with-level-def*
conflict-is-false-with-level-def abs-state-def cdcl$_W$*-restart-mset-state*)
ultimately have ‹*cdcl*$_W$*-restart-mset.conflict-is-false-with-level* (*abs-state T*)›

using *cdcl*$_W$*-restart-mset.cdcl*$_W$*-o-conflict-is-false-with-level-inv*[*of* ‹*abs-state S*› ‹*abs-state T*›]
lev confl-inv **unfolding** *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*
by (*auto dest*!: *cdcl*$_W$*-restart-mset.cdcl*$_W$*-o.intros cdcl*$_W$*-restart-mset.cdcl*$_W$*-bj.intros*)
then show ‹*?case*›
by (*auto simp*: *cdcl*$_W$*-restart-mset.conflict-is-false-with-level-def*
conflict-is-false-with-level-def abs-state-def cdcl$_W$*-restart-mset-state*)
next
case *backtrack*
then show *?case*
by (*auto split*: *if-split-asm simp*: *cdcl*$_W$*-M-level-inv-decomp lev conflict-is-false-with-level-def*)
qed (*auto simp*: *conflict-is-false-with-level-def*)

lemma *cdcl-bnb-stgy-conflict-is-false-with-level*:
assumes ‹*cdcl-bnb-stgy S T*› **and**
‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)› **and**
‹*no-smaller-confl S*› **and**
‹*conflict-is-false-with-level S*›
shows ‹*conflict-is-false-with-level T*›
using *assms*
proof (*induction rule*: *cdcl-bnb-stgy.cases*)
case (*cdcl-bnb-conflict S'*)
then show *?case*
using *conflict-conflict-is-false-with-level*
by (*auto simp*: *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*)
next
case (*cdcl-bnb-propagate S'*)
then show *?case*
using *propagate-conflict-is-false-with-level*
by (*auto simp*: *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*)
next
case (*cdcl-bnb-improve S'*)
then show *?case*
using *improve-conflict-is-false-with-level* **by** *blast*
next
case (*cdcl-bnb-conflict-opt S'*)
then show *?case*
using *conflict-opt-no-smaller-conflict*(*2*) **by** *blast*
next
case (*cdcl-bnb-other' S'*)
show *?case*
apply (*rule ocdcl*$_W$*-o-conflict-is-false-with-level-inv*)
using *cdcl-bnb-other'* **by** (*auto simp*: *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*)
qed

lemma *decided-cons-eq-append-decide-cons*: ‹$Decided\ L\ \#\ MM = M'\ @\ Decided\ K\ \#\ M \longleftrightarrow$
$(M' \neq [] \wedge hd\ M' = Decided\ L \wedge MM = tl\ M'\ @\ Decided\ K\ \#\ M) \vee$
$(M' = [] \wedge L = K \wedge MM = M)$›
by (*cases M'*) *auto*

lemma *either-all-false-or-earliest-decomposition*:
shows ‹$(\forall K\ K'.\ L = K'\ @\ K \longrightarrow \neg P\ K) \vee$
$(\exists L'\ L''.\ L = L''\ @\ L' \wedge P\ L' \wedge (\forall K\ K'.\ L' = K'\ @\ K \longrightarrow K' \neq [] \longrightarrow \neg P\ K))$›
apply (*induction L*)
subgoal by *auto*
subgoal for *a*

```
    by (metis append-Cons append-Nil list.sel(3) tl-append2)
  done

lemma trail-is-improving-Ex-improve:
  assumes confl: ‹conflicting S = None› and
    imp: ‹is-improving (trail S) M' S›
  shows ‹Ex (improvep S)›
  using assms
  by (auto simp: improvep.simps intro!: exI)

definition cdcl-bnb-stgy-inv :: ‹'st ⇒ bool› where
  ‹cdcl-bnb-stgy-inv S ⟷ conflict-is-false-with-level S ∧ no-smaller-confl S›

lemma cdcl-bnb-stgy-invD:
  shows ‹cdcl-bnb-stgy-inv S ⟷ cdcl_W-stgy-invariant S›
  unfolding cdcl_W-stgy-invariant-def cdcl-bnb-stgy-inv-def
  by auto

lemma cdcl-bnb-stgy-stgy-inv:
  ‹cdcl-bnb-stgy S T ⟹ cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state S) ⟹
    cdcl-bnb-stgy-inv S ⟹ cdcl-bnb-stgy-inv T›
  using cdcl_W-stgy-cdcl_W-stgy-invariant[of S T]
     cdcl-bnb-stgy-conflict-is-false-with-level cdcl-bnb-stgy-no-smaller-confl
  unfolding cdcl-bnb-stgy-inv-def
  by blast

lemma rtranclp-cdcl-bnb-stgy-stgy-inv:
  ‹cdcl-bnb-stgy** S T ⟹ cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state S) ⟹
    cdcl-bnb-stgy-inv S ⟹ cdcl-bnb-stgy-inv T›
  apply (induction rule: rtranclp-induct)
  subgoal by auto
  subgoal for T U
    using cdcl-bnb-stgy-stgy-inv rtranclp-cdcl-bnb-stgy-all-struct-inv
      rtranclp-cdcl-bnb-stgy-cdcl-bnb by blast
  done

lemma cdcl-bnb-cdcl_W-learned-clauses-entailed-by-init:
  assumes
    ‹cdcl-bnb S T› and
    entailed: ‹cdcl_W-restart-mset.cdcl_W-learned-clauses-entailed-by-init (abs-state S)› and
    all-struct: ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state S)›
  shows ‹cdcl_W-restart-mset.cdcl_W-learned-clauses-entailed-by-init (abs-state T)›
  using assms(1)
proof (induction rule: cdcl-bnb.cases)
  case (cdcl-conflict S')
  then show ?case
    using entailed
    by (auto simp: cdcl_W-restart-mset.cdcl_W-learned-clauses-entailed-by-init-def
        elim!: conflictE)
next
  case (cdcl-propagate S')
  then show ?case
    using entailed
    by (auto simp: cdcl_W-restart-mset.cdcl_W-learned-clauses-entailed-by-init-def
        elim!: propagateE)
next
```

case (*cdcl-improve S′*)
moreover have ‹*set-mset* (*CDCL-W-Abstract-State.init-clss* (*abs-state S*)) ⊆
set-mset (*CDCL-W-Abstract-State.init-clss* (*abs-state* (*update-weight-information M′ S*)))›
if ‹*is-improving M M′ S*› **for** *M M′*
using *that conflicting-clss-update-weight-information-mono*[*OF all-struct*]
by (*auto simp*: *abs-state-def* $cdcl_W$*-restart-mset-state*)
ultimately show *?case*
using *entailed*
by (*fastforce simp*: $cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init-def*
elim!: *improveE intro*: *true-clss-clss-subsetI*)
next
case (*cdcl-other′ S′*) **note** *T* = *this*(*1*) **and** *o* = *this*(*2*)
show *?case*
apply (*rule* $cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed*[*of* ‹*abs-state S*›])
subgoal using *o* **unfolding** *T* **by** (*blast dest*: $cdcl_W$*-o-*$cdcl_W$*-o* $cdcl_W$*-restart-mset.other*)
subgoal using *all-struct* **unfolding** $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def* **by** *fast*
subgoal using *entailed* **by** *fast*
done
next
case (*cdcl-conflict-opt S′*)
then show *?case*
using *entailed*
by (*auto simp*: $cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init-def*
elim!: *conflict-optE*)
qed

lemma *rtranclp-cdcl-bnb-*$cdcl_W$*-learned-clauses-entailed-by-init*:
assumes
‹$cdcl\text{-}bnb^{**}$ *S T*› **and**
entailed: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init* (*abs-state S*)› **and**
all-struct: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state S*)›
shows ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init* (*abs-state T*)›
using *assms* **by** (*induction rule*: *rtranclp-induct*)
(*auto intro*: *cdcl-bnb-*$cdcl_W$*-learned-clauses-entailed-by-init*
rtranclp-cdcl-bnb-stgy-all-struct-inv)

lemma *atms-of-init-clss-conflicting-clss2*[*simp*]:
‹*atms-of-mm* (*init-clss S*) ∪ *atms-of-mm* (*conflicting-clss S*) = *atms-of-mm* (*init-clss S*)›
using *atms-of-conflicting-clss*[*of S*] **by** *blast*

lemma *no-strange-atm-no-strange-atm*[*simp*]:
‹$cdcl_W$*-restart-mset.no-strange-atm* (*abs-state S*) = *no-strange-atm S*›
using *atms-of-conflicting-clss*[*of S*]
unfolding $cdcl_W$*-restart-mset.no-strange-atm-def no-strange-atm-def*
by (*auto simp*: *abs-state-def* $cdcl_W$*-restart-mset-state*)

lemma $cdcl_W$*-conflicting-*$cdcl_W$*-conflicting*[*simp*]:
‹$cdcl_W$*-restart-mset.*$cdcl_W$*-conflicting* (*abs-state S*) = $cdcl_W$*-conflicting S*›
unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-conflicting-def* $cdcl_W$*-conflicting-def*
by (*auto simp*: *abs-state-def* $cdcl_W$*-restart-mset-state*)

lemma *distinct-*$cdcl_W$*-state-distinct-*$cdcl_W$*-state*:
‹$cdcl_W$*-restart-mset.distinct-*$cdcl_W$*-state* (*abs-state S*) ⟹ *distinct-*$cdcl_W$*-state S*›
unfolding $cdcl_W$*-restart-mset.distinct-*$cdcl_W$*-state-def distinct-*$cdcl_W$*-state-def*
by (*auto simp*: *abs-state-def* $cdcl_W$*-restart-mset-state*)

lemma *obacktrack-imp-backtrack*:
‹*obacktrack S T* ⟹ $cdcl_W$*-restart-mset.backtrack* (*abs-state S*) (*abs-state T*)›
by (*elim obacktrackE*, *rule-tac D*=*D* **and** *L*=*L* **and** *K*=*K* **in** $cdcl_W$*-restart-mset.backtrack.intros*)
(*auto elim*!: *obacktrackE simp*: $cdcl_W$*-restart-mset.backtrack.simps sim-abs-state-simp*)

lemma *backtrack-imp-obacktrack*:
‹$cdcl_W$*-restart-mset.backtrack* (*abs-state S*) *T* ⟹ *Ex* (*obacktrack S*)›
by (*elim* $cdcl_W$*-restart-mset.backtrackE*, *rule exI*,
rule-tac D=*D* **and** *L*=*L* **and** *K*=*K* **in** *obacktrack.intros*)
(*auto simp*: $cdcl_W$*-restart-mset.backtrack.simps obacktrack.simps*)

lemma $cdcl_W$*-same-weight*: ‹$cdcl_W$ *S U* ⟹ *weight S* = *weight U*›
by (*induction rule*: $cdcl_W$*.induct*)
(*auto simp*: *improvep.simps* $cdcl_W$*.simps*
propagate.simps sim-abs-state-simp abs-state-def $cdcl_W$*-restart-mset-state*
clauses-def conflict.simps $cdcl_W$*-o.simps decide.simps* $cdcl_W$*-bj.simps*
skip.simps resolve.simps backtrack.simps)

lemma *o*$cdcl_W$*-o-same-weight*: ‹*o*$cdcl_W$*-o S U* ⟹ *weight S* = *weight U*›
by (*induction rule*: *o*$cdcl_W$*-o.induct*)
(*auto simp*: *improvep.simps* $cdcl_W$*.simps cdcl-bnb-bj.simps*
propagate.simps sim-abs-state-simp abs-state-def $cdcl_W$*-restart-mset-state*
clauses-def conflict.simps $cdcl_W$*-o.simps decide.simps* $cdcl_W$*-bj.simps*
skip.simps resolve.simps obacktrack.simps)

This is a proof artefact: it is easier to reason on *improvep* when the set of initial clauses is fixed (here by *N*). The next theorem shows that the conclusion is equivalent to not fixing the set of clauses.

lemma *wf-cdcl-bnb*:
assumes *improve*: ‹⋀*S T*. *improvep S T* ⟹ *init-clss S* = *N* ⟹ (ν (*weight T*), ν (*weight S*)) ∈ *R*›
and
wf-R: ‹*wf R*›
shows ‹*wf* {(*T*, *S*). $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state S*) ∧ *cdcl-bnb S T* ∧
init-clss S = *N*}›
(**is** ‹*wf ?A*›)
proof −
let *?R* = ‹{(*T*, *S*). (ν (*weight T*), ν (*weight S*)) ∈ *R*}›

have ‹*wf* {(*T*, *S*). $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv S* ∧ $cdcl_W$*-restart-mset.*$cdcl_W$ *S T*}›
by (*rule* $cdcl_W$*-restart-mset.wf-*$cdcl_W$)
from *wf-if-measure-f*[*OF this*, *of abs-state*]
have *wf*: ‹*wf* {(*T*, *S*). $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state S*) ∧
$cdcl_W$*-restart-mset.*$cdcl_W$ (*abs-state S*) (*abs-state T*) ∧ *weight S* = *weight T*}›
(**is** ‹*wf ?CDCL*›)
by (*rule wf-subset*) *auto*
have ‹*wf* (*?R* ∪ *?CDCL*)›
apply (*rule wf-union-compatible*)
subgoal by (*rule wf-if-measure-f*[*OF wf-R*, *of* ‹λ*x*. ν (*weight x*)›])
subgoal by (*rule wf*)
subgoal by (*auto simp*: $cdcl_W$*-same-weight*)
done

moreover have ‹*?A* ⊆ *?R* ∪ *?CDCL*›
by (*auto dest*: $cdcl_W$*.intros* $cdcl_W$*-restart-mset.W-propagate* $cdcl_W$*-restart-mset.W-other*
conflict-conflict propagate-propagate decide-decide improve conflict-opt-conflict

$cdcl_W$-o-$cdcl_W$-o $cdcl_W$-restart-mset.W-conflict W-conflict $cdcl_W$-o.intros $cdcl_W$.intros $cdcl_W$-o-$cdcl_W$-o
simp: *$cdcl_W$-same-weight cdcl-bnb.simps o$cdcl_W$-o-same-weight*
elim: *conflict-optE*)
ultimately show *?thesis*
by (*rule wf-subset*)
qed

corollary *wf-cdcl-bnb-fixed-iff*:
shows ‹($\forall N$. *wf* {(*T*, *S*). *$cdcl_W$-restart-mset.$cdcl_W$-all-struct-inv* (*abs-state S*) ∧ *cdcl-bnb S T* ∧ *init-clss S* = *N*}) ⟷
wf {(*T*, *S*). *$cdcl_W$-restart-mset.$cdcl_W$-all-struct-inv* (*abs-state S*) ∧ *cdcl-bnb S T*}›
(**is** ‹($\forall N$. *wf* (*?A N*)) ⟷ *wf ?B*›)
proof
assume ‹*wf ?B*›
then show ‹$\forall N$. *wf* (*?A N*)›
by (*intro allI*, *rule wf-subset*) *auto*
next
assume ‹$\forall N$. *wf* (*?A N*)›
show ‹*wf ?B*›
unfolding *wf-iff-no-infinite-down-chain*
proof
assume ‹$\exists f. \forall i.$ (*f* (*Suc i*), *f i*) ∈ *?B*›
then obtain *f* **where** *f*: ‹(*f* (*Suc i*), *f i*) ∈ *?B*› **for** *i*
by *blast*
then have ‹*$cdcl_W$-restart-mset.$cdcl_W$-all-struct-inv* (*abs-state* (*f n*))› **for** *n*
by (*induction n*) *auto*
with *f* **have** *st*: ‹*cdcl-bnb*** (*f 0*) (*f n*)› **for** *n*
apply (*induction n*)
subgoal by *auto*
subgoal by (*subst rtranclp-unfold*,*subst tranclp-unfold-end*)
auto
done
let *?N* = ‹*init-clss* (*f 0*)›
have *N*: ‹*init-clss* (*f n*) = *?N*› **for** *n*
using *st*[*of n*] **by** (*auto dest*: *rtranclp-cdcl-bnb-no-more-init-clss*)
have ‹(*f* (*Suc i*), *f i*) ∈ *?A ?N*› **for** *i*
using *f N* **by** *auto*
with ‹$\forall N$. *wf* (*?A N*)› **show** *False*
unfolding *wf-iff-no-infinite-down-chain* **by** *blast*
qed
qed

The following is a slightly more restricted version of the theorem, because it makes it possible to add some specific invariant, which can be useful when the proof of the decreasing is complicated.

lemma *wf-cdcl-bnb-with-additional-inv*:
assumes *improve*: ‹$\bigwedge S\ T$. *improvep S T* ⟹ *P S* ⟹ *init-clss S* = *N* ⟹ (ν (*weight T*), ν (*weight S*)) ∈ *R*› **and**
wf-R: ‹*wf R*› **and**
‹$\bigwedge S\ T$. *cdcl-bnb S T* ⟹ *P S* ⟹ *init-clss S* = *N* ⟹ *$cdcl_W$-restart-mset.$cdcl_W$-all-struct-inv* (*abs-state S*) ⟹ *P T*›
shows ‹*wf* {(*T*, *S*). *$cdcl_W$-restart-mset.$cdcl_W$-all-struct-inv* (*abs-state S*) ∧ *cdcl-bnb S T* ∧ *P S* ∧ *init-clss S* = *N*}›
(**is** ‹*wf ?A*›)
proof –
let *?R* = ‹{(*T*, *S*). (ν (*weight T*), ν (*weight S*)) ∈ *R*}›

have ‹*wf* {(*T*, *S*). cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv *S* ∧ cdcl$_W$-restart-mset.cdcl$_W$ *S* *T*}›
by (*rule* cdcl$_W$-restart-mset.wf-cdcl$_W$)
from *wf-if-measure-f*[*OF this*, *of abs-state*]
have *wf*: ‹*wf* {(*T*, *S*). cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv (*abs-state S*) ∧
cdcl$_W$-restart-mset.cdcl$_W$ (*abs-state S*) (*abs-state T*) ∧ *weight S* = *weight T*}›
(**is** ‹*wf ?CDCL*›)
by (*rule wf-subset*) *auto*
have ‹*wf* (*?R* ∪ *?CDCL*)›
apply (*rule wf-union-compatible*)
subgoal by (*rule wf-if-measure-f*[*OF wf-R*, *of* ‹λ*x*. ν (*weight x*)›])
subgoal by (*rule wf*)
subgoal by (*auto simp*: cdcl$_W$-same-weight)
done

moreover have ‹*?A* ⊆ *?R* ∪ *?CDCL*›
using *assms*(*3*) *cdcl-bnb.intros*(*3*)
by (*auto dest*: cdcl$_W$.intros cdcl$_W$-restart-mset.W-propagate cdcl$_W$-restart-mset.W-other
conflict-conflict propagate-propagate decide-decide improve conflict-opt-conflict
cdcl$_W$-o-cdcl$_W$-o cdcl$_W$-restart-mset.W-conflict W-conflict cdcl$_W$-o.intros cdcl$_W$.intros
cdcl$_W$-o-cdcl$_W$-o
simp: cdcl$_W$-same-weight cdcl-bnb.simps ocdcl$_W$-o-same-weight
elim: *conflict-optE*)
ultimately show *?thesis*
by (*rule wf-subset*)
qed

lemma *conflict-is-false-with-level-abs-iff*:
‹cdcl$_W$-restart-mset.conflict-is-false-with-level (*abs-state S*) ⟷
conflict-is-false-with-level S›
by (*auto simp*: cdcl$_W$-restart-mset.conflict-is-false-with-level-def
conflict-is-false-with-level-def)

lemma *decide-abs-state-decide*:
‹cdcl$_W$-restart-mset.decide (*abs-state S*) *T* ⟹ *cdcl-bnb-struct-invs S* ⟹ *Ex*(*decide S*)›
apply (*cases rule*: cdcl$_W$-restart-mset.decide.cases, *assumption*)
subgoal for *L*
apply (*rule exI*)
apply (*rule decide.intros*[*of - L*])
by (*auto simp*: *cdcl-bnb-struct-invs-def abs-state-def* cdcl$_W$-restart-mset-state)
done

lemma cdcl-bnb-no-conflicting-clss-cdcl$_W$:
assumes ‹*cdcl-bnb S T*› **and** ‹*conflicting-clss T* = {#}›
shows ‹cdcl$_W$-restart-mset.cdcl$_W$ (*abs-state S*) (*abs-state T*) ∧ *conflicting-clss S* = {#}›
using *assms*
by (*auto simp*: *cdcl-bnb.simps conflict-opt.simps improvep.simps* ocdcl$_W$-o.simps
cdcl-bnb-bj.simps
dest: *conflict-conflict propagate-propagate decide-decide skip-skip resolve-resolve*
backtrack-backtrack
intro: cdcl$_W$-restart-mset.W-conflict cdcl$_W$-restart-mset.W-propagate cdcl$_W$-restart-mset.W-other
dest: *conflicting-clss-update-weight-information-in*
elim: *conflictE propagateE decideE skipE resolveE improveE obacktrackE*)

lemma rtranclp-cdcl-bnb-no-conflicting-clss-cdcl$_W$:

assumes ‹*cdcl-bnb*** *S T*› **and** ‹*conflicting-clss T* = {#}›
shows ‹$cdcl_W$*-restart-mset.*$cdcl_W$** (*abs-state S*) (*abs-state T*) ∧ *conflicting-clss S* = {#}›
using *assms*
by (*induction rule*: *rtranclp-induct*)
(*fastforce dest*: *cdcl-bnb-no-conflicting-clss-*$cdcl_W$)+

lemma *conflict-abs-ex-conflict-no-conflicting*:
assumes ‹$cdcl_W$*-restart-mset.conflict* (*abs-state S*) *T*› **and** ‹*conflicting-clss S* = {#}›
shows ‹∃ *T*. *conflict S T*›
using *assms* **by** (*auto simp*: *conflict.simps* $cdcl_W$*-restart-mset.conflict.simps abs-state-def*
$cdcl_W$*-restart-mset-state clauses-def* $cdcl_W$*-restart-mset.clauses-def*)

lemma *propagate-abs-ex-propagate-no-conflicting*:
assumes ‹$cdcl_W$*-restart-mset.propagate* (*abs-state S*) *T*› **and** ‹*conflicting-clss S* = {#}›
shows ‹∃ *T*. *propagate S T*›
using *assms* **by** (*auto simp*: *propagate.simps* $cdcl_W$*-restart-mset.propagate.simps abs-state-def*
$cdcl_W$*-restart-mset-state clauses-def* $cdcl_W$*-restart-mset.clauses-def*)

lemma *cdcl-bnb-stgy-no-conflicting-clss-*$cdcl_W$*-stgy*:
assumes ‹*cdcl-bnb-stgy S T*› **and** ‹*conflicting-clss T* = {#}›
shows ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-stgy* (*abs-state S*) (*abs-state T*)›
proof −
have ‹*conflicting-clss S* = {#}›
using *cdcl-bnb-no-conflicting-clss-*$cdcl_W$[*of S T*] *assms*
by (*auto dest*: *cdcl-bnb-stgy-cdcl-bnb*)
then show *?thesis*
using *assms*
by (*auto 7 5 simp*: *cdcl-bnb-stgy.simps conflict-opt.simps o*$cdcl_W$*-o.simps*
cdcl-bnb-bj.simps
dest: *conflict-conflict propagate-propagate decide-decide skip-skip resolve-resolve*
backtrack-backtrack
dest: $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy.intros* $cdcl_W$*-restart-mset.*$cdcl_W$*-o.intros*
dest: *conflicting-clss-update-weight-information-in*
conflict-abs-ex-conflict-no-conflicting
propagate-abs-ex-propagate-no-conflicting
intro: $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy.intros*(*3*)
elim: *improveE*)
qed

lemma *rtranclp-cdcl-bnb-stgy-no-conflicting-clss-*$cdcl_W$*-stgy*:
assumes ‹*cdcl-bnb-stgy*** *S T*› **and** ‹*conflicting-clss T* = {#}›
shows ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-stgy*** (*abs-state S*) (*abs-state T*)›
using *assms* **apply** (*induction rule*: *rtranclp-induct*)
subgoal by *auto*
subgoal for *T U*
using *cdcl-bnb-no-conflicting-clss-*$cdcl_W$[*of T U*, *OF cdcl-bnb-stgy-cdcl-bnb*]
by (*auto dest*: *cdcl-bnb-stgy-no-conflicting-clss-*$cdcl_W$*-stgy*)
done

context
assumes *can-always-improve*:
‹⋀*S*. *trail S* ⊨*asm clauses S* ⟹ *no-step conflict-opt S* ⟹
conflicting S = *None* ⟹
$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state S*) ⟹
total-over-m (*lits-of-l* (*trail S*)) (*set-mset* (*clauses S*)) ⟹ *Ex* (*improvep S*)›

begin

The following theorems states a non-obvious (and slightly subtle) property: The fact that there is no conflicting cannot be shown without additional assumption. However, the assumption that every model leads to an improvements implies that we end up with a conflict.

```
lemma no-step-cdcl-bnb-cdcl_W:
  assumes
    ns: ‹no-step cdcl-bnb S› and
    struct-invs: ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state S)›
  shows ‹no-step cdcl_W-restart-mset.cdcl_W (abs-state S)›
proof -
  have ns-confl: ‹no-step skip S›  ‹no-step resolve S›  ‹no-step obacktrack S› and
    ns-nc: ‹no-step conflict S› ‹no-step propagate S› ‹no-step improvep S› ‹no-step conflict-opt S›
      ‹no-step decide S›
    using ns
    by (auto simp: cdcl-bnb.simps ocdcl_W-o.simps cdcl-bnb-bj.simps)
  have alien: ‹cdcl_W-restart-mset.no-strange-atm (abs-state S)›
    using struct-invs unfolding cdcl_W-restart-mset.cdcl_W-all-struct-inv-def by fast+

  have False if st: ‹∃ T. cdcl_W-restart-mset.cdcl_W (abs-state S) T›
  proof (cases ‹conflicting S = None›)
    case True
    have ‹total-over-m (lits-of-l (trail S)) (set-mset (init-clss S))›
      using ns-nc True apply - apply (rule ccontr)
      by (force simp: decide.simps total-over-m-def total-over-set-def
        Decided-Propagated-in-iff-in-lits-of-l)
    then have tot: ‹total-over-m (lits-of-l (trail S)) (set-mset (clauses S))›
      using alien unfolding cdcl_W-restart-mset.no-strange-atm-def
      by (auto simp: total-over-set-atm-of total-over-m-def clauses-def
        abs-state-def init-clss.simps learned-clss.simps trail.simps)
    then have ‹trail S ⊨asm clauses S›
      using ns-nc True unfolding true-annots-def apply -
      apply clarify
      subgoal for C
        using all-variables-defined-not-imply-cnot[of C ‹trail S›]
        by (fastforce simp: conflict.simps total-over-set-atm-of
        dest: multi-member-split)
      done
    from can-always-improve[OF this] have ‹False›
      using ns-nc True struct-invs tot by blast
    then show ‹?thesis›
      by blast
  next
    case False
    have nss: ‹no-step cdcl_W-restart-mset.skip (abs-state S)›
       ‹no-step cdcl_W-restart-mset.resolve (abs-state S)›
       ‹no-step cdcl_W-restart-mset.backtrack (abs-state S)›
      using ns-confl by (force simp: cdcl_W-restart-mset.skip.simps skip.simps
        cdcl_W-restart-mset.resolve.simps resolve.simps
        dest: backtrack-imp-obacktrack)+
    then show ‹?thesis›
      using that False by (auto simp: cdcl_W-restart-mset.cdcl_W.simps
        cdcl_W-restart-mset.propagate.simps cdcl_W-restart-mset.conflict.simps
        cdcl_W-restart-mset.cdcl_W-o.simps cdcl_W-restart-mset.decide.simps
        cdcl_W-restart-mset.cdcl_W-bj.simps)
```

qed
then show ‹*?thesis*› **by** *blast*
qed

lemma *no-step-cdcl-bnb-stgy*:
assumes
n-s: ‹*no-step cdcl-bnb S*› **and**
all-struct: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state S*)› **and**
stgy-inv: ‹*cdcl-bnb-stgy-inv S*›
shows ‹*conflicting S* = *None* ∨ *conflicting S* = *Some* {#}›
proof (*rule ccontr*)
assume ‹¬ *?thesis*›
then obtain *D* **where** ‹*conflicting S* = *Some D*› **and** ‹*D* ≠ {#}›
by *auto*
moreover have ‹*no-step* $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy* (*abs-state S*)›
using *no-step-cdcl-bnb-*$cdcl_W$[*OF n-s all-struct*]
$cdcl_W$*-restart-mset.*$cdcl_W$*-stgy-*$cdcl_W$ **by** *blast*
moreover have *le*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause* (*abs-state S*)›
using *all-struct* **unfolding** $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def* **by** *fast*
ultimately show *False*
using $cdcl_W$*-restart-mset.conflicting-no-false-can-do-step*[*of* ‹*abs-state S*›] *all-struct stgy-inv le*
unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def cdcl-bnb-stgy-inv-def*
by (*force dest*: *distinct-*$cdcl_W$*-state-distinct-*$cdcl_W$*-state*
simp: *conflict-is-false-with-level-abs-iff*)
qed

lemma *no-step-cdcl-bnb-stgy-empty-conflict*:
assumes
n-s: ‹*no-step cdcl-bnb S*› **and**
all-struct: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state S*)› **and**
stgy-inv: ‹*cdcl-bnb-stgy-inv S*›
shows ‹*conflicting S* = *Some* {#}›
proof (*rule ccontr*)
assume *H*: ‹¬ *?thesis*›
have *all-struct'*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state S*)›
by (*simp add*: *all-struct*)
have *le*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause* (*abs-state S*)›
using *all-struct*
unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def cdcl-bnb-stgy-inv-def*
by *auto*
have ‹*conflicting S* = *None* ∨ *conflicting S* = *Some* {#}›
using *no-step-cdcl-bnb-stgy*[*OF n-s all-struct' stgy-inv*] **.**
then have *confl*: ‹*conflicting S* = *None*›
using *H* **by** *blast*
have ‹*no-step* $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy* (*abs-state S*)›
using *no-step-cdcl-bnb-*$cdcl_W$[*OF n-s all-struct*]
$cdcl_W$*-restart-mset.*$cdcl_W$*-stgy-*$cdcl_W$ **by** *blast*
then have *entail*: ‹*trail S* ⊨*asm clauses S*›
using *confl* $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy-final-state-conclusive2*[*of* ‹*abs-state S*›]
all-struct stgy-inv le
unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def cdcl-bnb-stgy-inv-def*
by (*auto simp*: *conflict-is-false-with-level-abs-iff*)
have ‹*total-over-m* (*lits-of-l* (*trail S*)) (*set-mset* (*clauses S*))›
using $cdcl_W$*-restart-mset.no-step-*$cdcl_W$*-total*[*OF no-step-cdcl-bnb-*$cdcl_W$, *of S*] *all-struct n-s confl*
unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*

by *auto*
with *can-always-improve entail confl all-struct*
show ‹*False*›
using *n-s* **by** (*auto simp*: *cdcl-bnb.simps*)
qed

lemma *full-cdcl-bnb-stgy-no-conflicting-clss-unsat*:
assumes
full: ‹*full cdcl-bnb-stgy S T*› **and**
all-struct: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state S*)› **and**
stgy-inv: ‹*cdcl-bnb-stgy-inv S*› **and**
ent-init: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init* (*abs-state S*)› **and**
[*simp*]: ‹*conflicting-clss T* = {#}›
shows ‹*unsatisfiable* (*set-mset* (*init-clss S*))›
proof −
have *ns*: ‹*no-step cdcl-bnb-stgy T*› **and**
st: ‹$cdcl\text{-}bnb\text{-}stgy^{**}$ *S T*› **and**
st': ‹$cdcl\text{-}bnb^{**}$ *S T*› **and**
ns': ‹*no-step cdcl-bnb T*›
using *full* **unfolding** *full-def* **apply** (*blast dest*: *rtranclp-cdcl-bnb-stgy-cdcl-bnb*)+
using *full* **unfolding** *full-def*
by (*metis cdcl-bnb.simps cdcl-bnb-conflict cdcl-bnb-conflict-opt cdcl-bnb-improve*
cdcl-bnb-other' cdcl-bnb-propagate no-confl-prop-impr.elims(*3*))
have *struct-T*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state T*)›
using *rtranclp-cdcl-bnb-stgy-all-struct-inv*[*OF st' all-struct*] **.**
have [*simp*]: ‹*conflicting-clss S* = {#}›
using *rtranclp-cdcl-bnb-no-conflicting-clss-*$cdcl_W$[*OF st'*] **by** *auto*
have ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-stgy*** (*abs-state S*) (*abs-state T*)›
using *rtranclp-cdcl-bnb-stgy-no-conflicting-clss-*$cdcl_W$*-stgy*[*OF st*] **by** *auto*
then have ‹*full* $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy* (*abs-state S*) (*abs-state T*)›
using *no-step-cdcl-bnb-*$cdcl_W$[*OF ns' struct-T*] **unfolding** *full-def*
by (*auto dest*: $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy-*$cdcl_W$)
moreover have ‹$cdcl_W$*-restart-mset.no-smaller-confl* (*state-butlast S*)›
using *stgy-inv ent-init*
unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def conflict-is-false-with-level-abs-iff*
cdcl-bnb-stgy-inv-def conflict-is-false-with-level-abs-iff
$cdcl_W$*-restart-mset.*$cdcl_W$*-stgy-invariant-def*
by (*auto simp*: *abs-state-def* $cdcl_W$*-restart-mset-state cdcl-bnb-stgy-inv-def*
no-smaller-confl-def $cdcl_W$*-restart-mset.no-smaller-confl-def clauses-def*
$cdcl_W$*-restart-mset.clauses-def*)
ultimately have *conflicting T* = *Some* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss S*))
∨ *conflicting T* = *None* ∧ *trail T* ⊨*asm init-clss S*
using $cdcl_W$*-restart-mset.full-*$cdcl_W$*-stgy-inv-normal-form*[*of* ‹*abs-state S*› ‹*abs-state T*›] *all-struct*
stgy-inv ent-init
unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def conflict-is-false-with-level-abs-iff*
cdcl-bnb-stgy-inv-def conflict-is-false-with-level-abs-iff
$cdcl_W$*-restart-mset.*$cdcl_W$*-stgy-invariant-def*
by (*auto simp*: *abs-state-def* $cdcl_W$*-restart-mset-state cdcl-bnb-stgy-inv-def*)
moreover have ‹*cdcl-bnb-stgy-inv T*›
using *rtranclp-cdcl-bnb-stgy-stgy-inv*[*OF st all-struct stgy-inv*] **.**
ultimately show ‹*?thesis*›
using *no-step-cdcl-bnb-stgy-empty-conflict*[*OF ns' struct-T*] **by** *auto*

qed

```
lemma ocdcl_W-o-no-smaller-propa:
  assumes ‹ocdcl_W-o S T› and
    inv: ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state S)› and
    smaller-propa: ‹no-smaller-propa S› and
    n-s: ‹no-confl-prop-impr S›
  shows ‹no-smaller-propa T›
  using assms(1)
proof cases
  case decide
  show ?thesis
    unfolding no-smaller-propa-def
  proof clarify
    fix M K M' D L
    assume
      tr: ‹trail T = M' @ Decided K # M› and
      D: ‹D+{#L#} ∈# clauses T› and
      undef: ‹undefined-lit M L› and
      M: ‹M ⊨as CNot D›
    then have ‹Ex (propagate S)›
      apply (cases M')
      using propagate-rule[of S ‹D+{#L#}› L ‹cons-trail (Propagated L (D + {#L#})) S›]
        smaller-propa decide
      by (auto simp: no-smaller-propa-def elim!: rulesE)
    then show False
      using n-s unfolding no-confl-prop-impr.simps by blast
  qed
next
  case bj
  then show ?thesis
  proof cases
    case skip
    then show ?thesis
      using assms no-smaller-propa-tl[of S T]
      by (auto simp: cdcl-bnb-bj.simps ocdcl_W-o.simps obacktrack.simps elim!: rulesE)
  next
    case resolve
    then show ?thesis
      using assms no-smaller-propa-tl[of S T]
      by (auto simp: cdcl-bnb-bj.simps ocdcl_W-o.simps obacktrack.simps elim!: rulesE)
  next
    case backtrack
    have inv-T: ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state T)›
      using cdcl_W-restart-mset.cdcl_W-stgy-cdcl_W-all-struct-inv inv assms(1)
      using cdcl-bnb-stgy-all-struct-inv cdcl-other' by blast
    obtain D D' :: ‹'v clause› and K L :: ‹'v literal› and
      M1 M2 :: ‹('v, 'v clause) ann-lit list› and i :: nat where
      ‹conflicting S = Some (add-mset L D)› and
      decomp: ‹(Decided K # M1, M2) ∈ set (get-all-ann-decomposition (trail S))› and
      ‹get-level (trail S) L = backtrack-lvl S› and
      ‹get-level (trail S) L = get-maximum-level (trail S) (add-mset L D')› and
      i: ‹get-maximum-level (trail S) D' ≡ i› and
      lev-K: ‹get-level (trail S) K = i + 1› and
      D-D': ‹D' ⊆# D› and
      T: T ∼ cons-trail (Propagated L (add-mset L D'))
          (reduce-trail-to M1
            (add-learned-cls (add-mset L D')
```

```
      (update-conflicting None S)))
  using backtrack by (auto elim!: obacktrackE)
let ?D' = ‹add-mset L D'›
have [simp]: ‹trail (reduce-trail-to M1 S) = M1›
  using decomp by auto
obtain M'' c where M'': ‹trail S = M'' @ tl (trail T)› and c: ‹M'' = c @ M2 @ [Decided K]›
  using decomp T by auto
have M1: ‹M1 = tl (trail T)› and tr-T: ‹trail T = Propagated L ?D' # M1›
  using decomp T by auto
have lev-inv: ‹cdcl_W-restart-mset.cdcl_W-M-level-inv (abs-state S)›
  using inv unfolding cdcl_W-restart-mset.cdcl_W-all-struct-inv-def by auto
then have lev-inv-T: ‹cdcl_W-restart-mset.cdcl_W-M-level-inv (abs-state T)›
  using inv-T unfolding cdcl_W-restart-mset.cdcl_W-all-struct-inv-def by auto
have n-d: ‹no-dup (trail S)›
  using lev-inv unfolding cdcl_W-restart-mset.cdcl_W-M-level-inv-def
  by (auto simp: abs-state-def trail.simps)
have n-d-T: ‹no-dup (trail T)›
  using lev-inv-T unfolding cdcl_W-restart-mset.cdcl_W-M-level-inv-def
  by (auto simp: abs-state-def trail.simps)

have i-lvl: ‹i = backtrack-lvl T›
  using no-dup-append-in-atm-notin[of ‹c @ M2› ‹Decided K # tl (trail T)› K]
  n-d lev-K unfolding c M'' by (auto simp: image-Un tr-T)

from backtrack show ?thesis
  unfolding no-smaller-propa-def
proof clarify
  fix M K' M' E' L'
  assume
    tr: ‹trail T = M' @ Decided K' # M› and
    E: ‹E'+{#L'#} ∈# clauses T› and
    undef: ‹undefined-lit M L'› and
    M: ‹M ⊨as CNot E'›
  have False if D: ‹add-mset L D' = add-mset L' E'› and M-D: ‹M ⊨as CNot E'›
  proof -
    have ‹i ≠ 0›
      using i-lvl tr T by auto
    moreover {
      have ‹M1 ⊨as CNot D'›
        using inv-T tr-T unfolding cdcl_W-restart-mset.cdcl_W-all-struct-inv-def
        cdcl_W-restart-mset.cdcl_W-conflicting-def
        by (force simp: abs-state-def trail.simps conflicting.simps)
      then have ‹get-maximum-level M1 D' = i›
        using T i n-d D-D' unfolding M'' tr-T
        by (subst (asm) get-maximum-level-skip-beginning)
          (auto dest: defined-lit-no-dupD dest!: true-annots-CNot-definedD) }
    ultimately obtain L-max where
      L-max-in: ‹L-max ∈# D'› and
      lev-L-max: ‹get-level M1 L-max = i›
      using i get-maximum-level-exists-lit-of-max-level[of D' M1]
      by (cases D') auto
    have count-dec-M: ‹count-decided M < i›
      using T i-lvl unfolding tr by auto
    have ‹- L-max ∉ lits-of-l M›
    proof (rule ccontr)
      assume ‹¬ ?thesis›
```

then have ‹*undefined-lit* (*M'* @ [*Decided K'*]) *L-max*›
  using *n-d-T* **unfolding** *tr*
  by (*auto dest*: *in-lits-of-l-defined-litD dest*: *defined-lit-no-dupD simp*: *atm-of-eq-atm-of*)
then have ‹*get-level* (*tl M'* @ *Decided K'* # *M*) *L-max* < *i*›
  apply (*subst get-level-skip*)
   apply (*cases M'*; *auto simp add*: *atm-of-eq-atm-of lits-of-def*; *fail*)
  using *count-dec-M count-decided-ge-get-level*[*of M L-max*] **by** *auto*
then show *False*
  using *lev-L-max tr* **unfolding** *tr-T* **by** (*auto simp*: *propagated-cons-eq-append-decide-cons*)
qed
moreover have ‹− *L* ∉ *lits-of-l M*›
proof (*rule ccontr*)
  define *MM* **where** ‹*MM* = *tl M'*›
  assume ‹¬ *?thesis*›
  then have ‹− *L* ∉ *lits-of-l* (*M'* @ [*Decided K'*])›
    using *n-d-T* **unfolding** *tr* **by** (*auto simp*: *lits-of-def no-dup-def*)
  have ‹*undefined-lit* (*M'* @ [*Decided K'*]) *L*›
    apply (*rule no-dup-uminus-append-in-atm-notin*)
    using *n-d-T* ‹¬ − *L* ∉ *lits-of-l M*› **unfolding** *tr* **by** *auto*
  moreover have ‹*M'* = *Propagated L ?D'* # *MM*›
    using *tr-T MM-def* **by** (*metis hd-Cons-tl propagated-cons-eq-append-decide-cons tr*)
  ultimately show *False*
    by *simp*
qed
moreover have ‹*L-max* ∈# *D'* ∨ *L* ∈# *D'*›
  using *D L-max-in* **by** (*auto split*: *if-splits*)
ultimately show *False*
  using *M-D D* **by** (*auto simp*: *true-annots-true-cls true-clss-def add-mset-eq-add-mset*)
qed
then show *False*
  using *M''* *smaller-propa tr undef M T E*
  by (*cases M'*) (*auto simp*: *no-smaller-propa-def trivial-add-mset-remove-iff elim*!: *rulesE*)
qed
qed
qed

lemma *ocdcl*$_W$*-no-smaller-propa*:
  assumes ‹*cdcl-bnb-stgy S T*› **and**
    inv: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)› **and**
    smaller-propa: ‹*no-smaller-propa S*› **and**
    n-s: ‹*no-confl-prop-impr S*›
  shows ‹*no-smaller-propa T*›
  using *assms*
  apply (*cases*)
  subgoal by (*auto*)
  subgoal by (*auto*)
  subgoal by (*auto elim*!: *improveE simp*: *no-smaller-propa-def*)
  subgoal by (*auto elim*!: *conflict-optE simp*: *no-smaller-propa-def*)
  subgoal using *ocdcl*$_W$*-o-no-smaller-propa* **by** *fast*
  done

Unfortunately, we cannot reuse the proof we have alrealy done.

lemma *ocdcl*$_W$*-no-relearning*:
  assumes ‹*cdcl-bnb-stgy S T*› **and**
    inv: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)› **and**
    smaller-propa: ‹*no-smaller-propa S*› **and**

```
    n-s: ‹no-confl-prop-impr S› and
    dist: ‹distinct-mset (clauses S)›
  shows ‹distinct-mset (clauses T)›
  using assms(1)
proof cases
  case cdcl-bnb-conflict
  then show ?thesis using dist by (auto elim: rulesE)
next
  case cdcl-bnb-propagate
  then show ?thesis using dist by (auto elim: rulesE)
next
  case cdcl-bnb-improve
  then show ?thesis using dist by (auto elim: improveE)
next
  case cdcl-bnb-conflict-opt
  then show ?thesis using dist by (auto elim: conflict-optE)
next
  case cdcl-bnb-other'
  then show ?thesis
  proof cases
    case decide
    then show ?thesis using dist by (auto elim: rulesE)
  next
    case bj
    then show ?thesis
    proof cases
      case skip
      then show ?thesis using dist by (auto elim: rulesE)
    next
      case resolve
      then show ?thesis using dist by (auto elim: rulesE)
    next
      case backtrack
      have smaller-propa: ‹⋀M K M' D L.
        trail S = M' @ Decided K # M ⟹
        D + {#L#} ∈# clauses S ⟹ undefined-lit M L ⟹ ¬ M ⊨as CNot D›
        using smaller-propa unfolding no-smaller-propa-def by fast
      have inv: ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state T)›
        using inv
        using cdcl_W-restart-mset.cdcl_W-stgy-cdcl_W-all-struct-inv inv assms(1)
        using cdcl-bnb-stgy-all-struct-inv cdcl-other' backtrack ocdcl_W-o.intros
        cdcl-bnb-bj.intros
        by blast
      then have n-d: ‹no-dup (trail T)› and
        ent: ‹⋀L mark a b.
          a @ Propagated L mark # b = trail T ⟹
           b ⊨as CNot (remove1-mset L mark) ∧ L ∈# mark›
        unfolding cdcl_W-restart-mset.cdcl_W-M-level-inv-def
          cdcl_W-restart-mset.cdcl_W-all-struct-inv-def
           cdcl_W-restart-mset.cdcl_W-conflicting-def
        by (auto simp: abs-state-def trail.simps)
      show ?thesis
      proof (rule ccontr)
        assume H: ‹¬?thesis›
        obtain D D' :: ‹'v clause› and K L :: ‹'v literal› and
          M1 M2 :: ‹('v, 'v clause) ann-lit list› and i :: nat where
```

```
        ‹conflicting S = Some (add-mset L D)› and
        decomp: ‹(Decided K # M1, M2) ∈ set (get-all-ann-decomposition (trail S))› and
        ‹get-level (trail S) L = backtrack-lvl S› and
        ‹get-level (trail S) L = get-maximum-level (trail S) (add-mset L D')› and
        i: ‹get-maximum-level (trail S) D' ≡ i› and
        lev-K: ‹get-level (trail S) K = i + 1› and
        D-D': ‹D' ⊆# D› and
        T: T ∼ cons-trail (Propagated L (add-mset L D'))
            (reduce-trail-to M1
              (add-learned-cls (add-mset L D')
                (update-conflicting None S)))
        using backtrack by (auto elim!: obacktrackE)
      from H T dist have LD': ‹add-mset L D' ∈# clauses S›
        by auto
      have ‹¬M1 ⊨as CNot D'›
        using get-all-ann-decomposition-exists-prepend[OF decomp] apply (elim exE)
        by (rule smaller-propa[of ‹- @ M2› K M1 D' L])
          (use n-d T decomp LD' in auto)
      moreover have ‹M1 ⊨as CNot D'›
        using ent[of ‹[]› L ‹add-mset L D'› M1] T decomp by auto
      ultimately show False
        ..
    qed
  qed
 qed
qed


lemma full-cdcl-bnb-stgy-unsat:
  assumes
    st: ‹full cdcl-bnb-stgy S T› and
    all-struct: ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state S)› and
    opt-struct: ‹cdcl-bnb-struct-invs S› and
    stgy-inv: ‹cdcl-bnb-stgy-inv S›
  shows
    ‹unsatisfiable (set-mset (clauses T + conflicting-clss T))›
proof –
  have ns: ‹no-step cdcl-bnb-stgy T› and
    st: ‹cdcl-bnb-stgy** S T› and
    st': ‹cdcl-bnb** S T›
    using st unfolding full-def by (auto intro: rtranclp-cdcl-bnb-stgy-cdcl-bnb)
  have ns': ‹no-step cdcl-bnb T›
    by (meson cdcl-bnb.cases cdcl-bnb-stgy.simps no-confl-prop-impr.elims(3) ns)
  have struct-T: ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state T)›
    using rtranclp-cdcl-bnb-stgy-all-struct-inv[OF st' all-struct] .
  have stgy-T: ‹cdcl-bnb-stgy-inv T›
    using rtranclp-cdcl-bnb-stgy-stgy-inv[OF st all-struct stgy-inv] .
  have confl: ‹conflicting T = Some {#}›
    using no-step-cdcl-bnb-stgy-empty-conflict[OF ns' struct-T stgy-T] .

  have ‹cdcl_W-restart-mset.cdcl_W-learned-clause (abs-state T)› and
    alien: ‹cdcl_W-restart-mset.no-strange-atm (abs-state T)›
    using struct-T unfolding cdcl_W-restart-mset.cdcl_W-all-struct-inv-def by fast+
  then have ent': ‹set-mset (clauses T + conflicting-clss T) ⊨p {#}›
    using confl unfolding cdcl_W-restart-mset.cdcl_W-learned-clause-alt-def
    by auto
```

then show ‹*unsatisfiable* (*set-mset* (*clauses* *T* + *conflicting-clss* *T*))›
  unfolding *true-clss-cls-def satisfiable-def* **by** *auto*

qed

end

lemma *cdcl-bnb-reasons-in-clauses*:
‹*cdcl-bnb* *S* *T* ⟹ *reasons-in-clauses* *S* ⟹ *reasons-in-clauses* *T*›
by (*auto simp*: *cdcl-bnb.simps reasons-in-clauses-def ocdcl*$_W$*-o.simps*
  cdcl-bnb-bj.simps get-all-mark-of-propagated-tl-proped
  elim!: *rulesE improveE conflict-optE obacktrackE*
  dest!: *in-set-tlD get-all-ann-decomposition-exists-prepend*)

lemma *cdcl-bnb-pow2-n-learned-clauses*:
assumes ‹*distinct-mset-mset* *N*›
  ‹*cdcl-bnb*** (*init-state* *N*) *T*›
shows ‹*size* (*learned-clss* *T*) ≤ *2* ^ (*card* (*atms-of-mm* *N*))›
proof −
have *H*: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state* (*init-state* *N*))›
  using *assms* **apply** (*auto simp*: *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*
  cdcl$_W$*-restart-mset.distinct-cdcl*$_W$*-state-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-learned-clause-def*
  cdcl$_W$*-restart-mset.reasons-in-clauses-def*)
  using *assms* **by** (*auto simp*: *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*
  distinct-mset-mset-conflicting-clss
  cdcl$_W$*-restart-mset.distinct-cdcl*$_W$*-state-def abs-state-def init-clss.simps*)
then obtain *Na* **where** *Na*: ‹ *cdcl*$_W$*-restart-mset.cdcl*$_W$**
  (*trail* (*init-state* *N*), *init-clss* (*init-state* *N*) + *Na*,
  learned-clss (*init-state* *N*), *conflicting* (*init-state* *N*))
  (*abs-state* *T*) ∧
  CDCL-W-Abstract-State.init-clss (*abs-state* *T*) = *init-clss* (*init-state* *N*) + *Na*›
  using *rtranclp-cdcl-or-improve-cdclD*[*OF H assms*(*2*)] **by** *auto*
moreover have ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* ([], *N* + *Na*, {#}, *None*)›
  using *assms Na rtranclp-cdcl-bnb-no-more-init-clss*[*OF assms*(*2*)]
  apply (*auto simp*: *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*
  cdcl$_W$*-restart-mset.distinct-cdcl*$_W$*-state-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-learned-clause-def*
  cdcl$_W$*-restart-mset.reasons-in-clauses-def*)
  using *assms* **by** (*auto simp*: *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-restart-mset-state*
  distinct-mset-mset-conflicting-clss cdcl$_W$*-restart-mset.no-strange-atm-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-M-level-inv-def*
  cdcl$_W$*-restart-mset.cdcl*$_W$*-conflicting-def*
  cdcl$_W$*-restart-mset.distinct-cdcl*$_W$*-state-def abs-state-def init-clss.simps*)
ultimately show *?thesis*
  using *rtranclp-cdcl-bnb-no-more-init-clss*[*OF assms*(*2*)]
  cdcl$_W$*-restart-mset.cdcl-pow2-n-learned-clauses2*[*of* ‹*N* + *Na*› ‹*abs-state* *T*›]
  by (*auto simp*: *init-state.simps abs-state-def cdcl*$_W$*-restart-mset-state*)
qed
end

end
theory *CDCL-W-Optimal-Model*
  imports *CDCL-W-BnB HOL−Library.Extended-Nat*
begin

OCDCL

The following datatype is equivalent to *'a option*. However, it has the opposite ordering. Therefore, I decided to use a different type instead of have a second order which conflicts with `~~/src/HOL/Library/Option_ord.thy`.

datatype *'a optimal-model* = *Not-Found* | *is-found*: *Found* (*the-optimal*: *'a*)

instantiation *optimal-model* :: (*ord*) *ord*
begin
  fun *less-optimal-model* :: ‹*'a* :: *ord optimal-model* ⇒ *'a optimal-model* ⇒ *bool*› **where**
  ‹*less-optimal-model Not-Found* - = *False*›
| ‹*less-optimal-model* (*Found* -) *Not-Found* ⟷ *True*›
| ‹*less-optimal-model* (*Found a*) (*Found b*) ⟷ *a* < *b*›

fun *less-eq-optimal-model* :: ‹*'a* :: *ord optimal-model* ⇒ *'a optimal-model* ⇒ *bool*› **where**
  ‹*less-eq-optimal-model Not-Found Not-Found* = *True*›
| ‹*less-eq-optimal-model Not-Found* (*Found* -) = *False*›
| ‹*less-eq-optimal-model* (*Found* -) *Not-Found* ⟷ *True*›
| ‹*less-eq-optimal-model* (*Found a*) (*Found b*) ⟷ *a* ≤ *b*›

instance
  by *standard*

end

instance *optimal-model* :: (*preorder*) *preorder*
  apply *standard*
  subgoal for *a b*
    by (*cases a*; *cases b*) (*auto simp*: *less-le-not-le*)
  subgoal for *a*
    by (*cases a*) *auto*
  subgoal for *a b c*
    by (*cases a*; *cases b*; *cases c*) (*auto dest*: *order-trans*)
  done

instance *optimal-model* :: (*order*) *order*
  apply *standard*
  subgoal for *a b*
    by (*cases a*; *cases b*) (*auto simp*: *less-le-not-le*)
  done

instance *optimal-model* :: (*linorder*) *linorder*
  apply *standard*
  subgoal for *a b*
    by (*cases a*; *cases b*) (*auto simp*: *less-le-not-le*)
  done

instantiation *optimal-model* :: (*wellorder*) *wellorder*
begin

lemma *wf-less-optimal-model*: ‹*wf* {(*M* :: *'a optimal-model*, *N*). *M* < *N*}›
proof −
  have *1*: ‹{(*M* :: *'a optimal-model*, *N*). *M* < *N*} =
    map-prod Found Found ‘ {(*M* :: *'a*, *N*). *M* < *N*} ∪
    {(*a*, *b*). *a* ≠ *Not-Found* ∧ *b* = *Not-Found*}› (**is** ‹*?A* = *?B* ∪ *?C*›)

apply (*auto simp*: *image-iff*)
apply (*case-tac a*; *case-tac b*)
apply *auto*
apply (*case-tac a*)
apply *auto*
done
have [*simp*]: ‹*inj Found*›
by (*auto simp*:*inj-on-def*)
have ‹*wf ?B*›
by (*rule wf-map-prod-image*) (*auto intro*: *wf*)
moreover have ‹*wf ?C*›
by (*rule wfI-pf*) *auto*
ultimately show ‹*wf* (*?A*)›
unfolding *1*
by (*rule wf-Un*) (*auto*)
qed

instance by *standard* (*metis CollectI split-conv wf-def wf-less-optimal-model*)

end

This locales includes only the assumption we make on the weight function.

locale *ocdcl-weight* =
fixes
ϱ :: ‹*'v clause* ⇒ *'a* :: {*linorder*}›
assumes
ϱ*-mono*: ‹*distinct-mset B* ⟹ *A* ⊆# *B* ⟹ ϱ *A* ≤ ϱ *B*›
begin

lemma ϱ*-empty-simp*[*simp*]:
assumes ‹*consistent-interp* (*set-mset A*)› ‹*distinct-mset A*›
shows ‹ϱ *A* ≥ ϱ {#}› ‹¬ϱ *A* < ϱ {#}› ‹ϱ *A* ≤ ϱ {#} ⟷ ϱ *A* = ϱ {#}›
using ϱ*-mono*[*of A* ‹{#}›] *assms*
by *auto*

abbreviation ϱ' :: ‹*'v clause option* ⇒ *'a optimal-model*› **where**
‹ϱ' *w* ≡ (*case w of None* ⇒ *Not-Found* | *Some w* ⇒ *Found* (ϱ *w*))›

definition *is-improving-int*
:: (*'v literal*, *'v literal*, *'b*) *annotated-lits* ⇒ (*'v literal*, *'v literal*, *'b*) *annotated-lits* ⇒ *'v clauses* ⇒
'v clause option ⇒ *bool*
where
‹*is-improving-int M M' N w* ⟷ *Found* (ϱ (*lit-of* '# *mset M'*)) < ϱ' *w* ∧
M' ⊨*asm N* ∧ *no-dup M'* ∧
lit-of '# *mset M'* ∈ *simple-clss* (*atms-of-mm N*) ∧
total-over-m (*lits-of-l M'*) (*set-mset N*) ∧
(∀ *M'*. *total-over-m* (*lits-of-l M'*) (*set-mset N*) ⟶ *mset M* ⊆# *mset M'* ⟶
lit-of '# *mset M'* ∈ *simple-clss* (*atms-of-mm N*) ⟶
ϱ (*lit-of* '# *mset M'*) = ϱ (*lit-of* '# *mset M*))›

definition *too-heavy-clauses*
:: ‹*'v clauses* ⇒ *'v clause option* ⇒ *'v clauses*›
where
‹*too-heavy-clauses M w* =
{#*pNeg C* | *C* ∈# *mset-set* (*simple-clss* (*atms-of-mm M*)). ϱ' *w* ≤ *Found* (ϱ *C*)#}›

definition *conflicting-clauses*
:: ‹'v clauses ⇒ 'v clause option ⇒ 'v clauses›
where
‹*conflicting-clauses N w* =
{#*C* ∈# *mset-set* (*simple-clss* (*atms-of-mm N*)). *too-heavy-clauses N w* ⊨*pm C*#}›

lemma *too-heavy-clauses-conflicting-clauses*:
‹*C* ∈# *too-heavy-clauses M w* ⟹ *C* ∈# *conflicting-clauses M w*›
by (*auto simp*: *conflicting-clauses-def too-heavy-clauses-def simple-clss-finite*)

lemma *too-heavy-clauses-contains-itself*:
‹*M* ∈ *simple-clss* (*atms-of-mm N*) ⟹ *pNeg M* ∈# *too-heavy-clauses N* (*Some M*)›
by (*auto simp*: *too-heavy-clauses-def simple-clss-finite*)

lemma *too-heavy-clause-None*[*simp*]: ‹*too-heavy-clauses M None* = {#}›
by (*auto simp*: *too-heavy-clauses-def*)

lemma *atms-of-mm-too-heavy-clauses-le*:
‹*atms-of-mm* (*too-heavy-clauses M I*) ⊆ *atms-of-mm M*›
by (*auto simp*: *too-heavy-clauses-def atms-of-ms-def simple-clss-finite dest*: *simple-clssE*)

lemma
atms-too-heavy-clauses-None:
‹*atms-of-mm* (*too-heavy-clauses M None*) = {}› **and**
atms-too-heavy-clauses-Some:
‹*atms-of w* ⊆ *atms-of-mm M* ⟹ *distinct-mset w* ⟹ ¬*tautology w* ⟹
atms-of-mm (*too-heavy-clauses M* (*Some w*)) = *atms-of-mm M*›
proof −
show ‹*atms-of-mm* (*too-heavy-clauses M None*) = {}›
by (*auto simp*: *too-heavy-clauses-def*)
assume *atms*: ‹*atms-of w* ⊆ *atms-of-mm M*› **and**
dist: ‹*distinct-mset w*› **and**
taut: ‹¬*tautology w*›
have ‹*atms-of-mm* (*too-heavy-clauses M* (*Some w*)) ⊆ *atms-of-mm M*›
by (*auto simp*: *too-heavy-clauses-def atms-of-ms-def simple-clss-finite*)
(*auto simp*: *simple-clss-def*)
let *?w* = ‹*w* + *Neg* '# {#*x* ∈# *mset-set* (*atms-of-mm M*). *x* ∉ *atms-of w*#}›
have [*simp*]: ‹*inj-on Neg A*› **for** *A*
by (*auto simp*: *inj-on-def*)
have *dist*: ‹*distinct-mset ?w*›
using *dist*
by (*auto simp*: *distinct-mset-add distinct-image-mset-inj distinct-mset-mset-set uminus-lit-swap*
disjunct-not-in dest: *multi-member-split*)
moreover have *not-tauto*: ‹¬*tautology ?w*›
by (*auto simp*: *tautology-union taut uminus-lit-swap dest*: *multi-member-split*)
ultimately have ‹*?w* ∈ (*simple-clss* (*atms-of-mm M*))›
using *atms* **by** (*auto simp*: *simple-clss-def*)
moreover have ‹ϱ *?w* ≥ ϱ *w*›
by (*rule* ϱ*-mono*) (*use dist not-tauto* **in** ‹*auto simp*: *consistent-interp-tuatology-mset-set tautology-decomp*›)
ultimately have ‹*pNeg ?w* ∈# *too-heavy-clauses M* (*Some w*)›
by (*auto simp*: *too-heavy-clauses-def simple-clss-finite*)
then have ‹*atms-of-mm M* ⊆ *atms-of-mm* (*too-heavy-clauses M* (*Some w*))›
by (*auto dest*!: *multi-member-split*)
then show ‹*atms-of-mm* (*too-heavy-clauses M* (*Some w*)) = *atms-of-mm M*›
using ‹*atms-of-mm* (*too-heavy-clauses M* (*Some w*)) ⊆ *atms-of-mm M*› **by** *blast*

```
qed

lemma entails-too-heavy-clauses-too-heavy-clauses:
  assumes
    ‹consistent-interp I› and
    tot: ‹total-over-m I (set-mset (too-heavy-clauses M w))› and
    ‹I ⊨m too-heavy-clauses M w› and
    w: ‹w ≠ None ⟹ atms-of (the w) ⊆ atms-of-mm M›
      ‹w ≠ None ⟹ ¬tautology (the w)›
      ‹w ≠ None ⟹ distinct-mset (the w)›
  shows ‹I ⊨m conflicting-clauses M w›
proof (cases w)
  case None
  have [simp]: ‹{x ∈ simple-clss (atms-of-mm M). tautology x} = {}›
    by (auto dest: simple-clssE)
  show ?thesis
    using None by (auto simp: conflicting-clauses-def true-clss-cls-tautology-iff
      simple-clss-finite)
next
  case w′: (Some w′)
  have ‹x ∈# mset-set (simple-clss (atms-of-mm M)) ⟹ total-over-set I (atms-of x)› for x
    using tot w atms-too-heavy-clauses-Some[of w′ M] unfolding w′
    by (auto simp: total-over-m-def simple-clss-finite total-over-set-alt-def
      dest!: simple-clssE)
  then show ?thesis
    using assms
    by (subst true-cls-mset-def)
      (auto simp: conflicting-clauses-def true-clss-cls-def
        dest!: spec[of - I])
qed

lemma not-entailed-too-heavy-clauses-ge:
  ‹C ∈ simple-clss (atms-of-mm N) ⟹ ¬ too-heavy-clauses N w ⊨pm pNeg C ⟹ ¬Found (ρ C) ≥ ρ′
w›
  using true-clss-cls-in[of ‹pNeg C› ‹set-mset (too-heavy-clauses N w)›]
    too-heavy-clauses-contains-itself
  by (auto simp: too-heavy-clauses-def simple-clss-finite
        image-iff)

lemma conflicting-clss-incl-init-clauses:
  ‹atms-of-mm (conflicting-clauses N w) ⊆ atms-of-mm (N)›
  unfolding conflicting-clauses-def
  apply (auto simp: simple-clss-finite)
  by (auto simp: simple-clss-def atms-of-ms-def split: if-splits)

lemma distinct-mset-mset-conflicting-clss2: ‹distinct-mset-mset (conflicting-clauses N w)›
  unfolding conflicting-clauses-def distinct-mset-set-def
  apply (auto simp: simple-clss-finite)
  by (auto simp: simple-clss-def)

lemma too-heavy-clauses-mono:
  ‹ρ a > ρ (lit-of ‘# mset M) ⟹ too-heavy-clauses N (Some a) ⊆#
      too-heavy-clauses N (Some (lit-of ‘# mset M))›
  by (auto simp: too-heavy-clauses-def multiset-filter-mono2
    intro!: multiset-filter-mono image-mset-subseteq-mono)
```

lemma *is-improving-conflicting-clss-update-weight-information*: ‹*is-improving-int M M′ N w* $\Longrightarrow$
conflicting-clauses N w ⊆# *conflicting-clauses N* (*Some* (*lit-of* '# *mset M′*))›
using *too-heavy-clauses-mono*[*of M′* ‹*the w*› ‹*N*›]
by (*cases* ‹*w*›)
(*auto simp*: *is-improving-int-def conflicting-clauses-def multiset-filter-mono2*
intro!: *image-mset-subseteq-mono*
intro: *true-clss-cls-subset*
dest: *simple-clssE*)

lemma *conflicting-clss-update-weight-information-in2*:
assumes ‹*is-improving-int M M′ N w*›
shows ‹*negate-ann-lits M′* ∈# *conflicting-clauses N* (*Some* (*lit-of* '# *mset M′*))›
using *assms* **apply** (*auto simp*: *simple-clss-finite*
conflicting-clauses-def is-improving-int-def)
by (*auto simp*: *is-improving-int-def conflicting-clauses-def multiset-filter-mono2 simple-clss-def*
lits-of-def negate-ann-lits-pNeg-lit-of image-iff dest: *total-over-m-atms-incl*
intro!: *true-clss-cls-in too-heavy-clauses-contains-itself*)

lemma *atms-of-init-clss-conflicting-clauses′*[*simp*]:
‹*atms-of-mm N* ∪ *atms-of-mm* (*conflicting-clauses N S*) = *atms-of-mm N*›
using *conflicting-clss-incl-init-clauses*[*of N*] **by** *blast*

lemma *entails-too-heavy-clauses-if-le*:
assumes
dist: ‹*distinct-mset I*› **and**
cons: ‹*consistent-interp* (*set-mset I*)› **and**
tot: ‹*atms-of I* = *atms-of-mm N*› **and**
le: ‹*Found* (ϱ *I*) < ϱ′ (*Some M′*)›
shows
‹*set-mset I* ⊨*m too-heavy-clauses N* (*Some M′*)›
proof −
show ‹*set-mset I* ⊨*m too-heavy-clauses N* (*Some M′*)›
unfolding *true-cls-mset-def*
proof
fix *C*
assume ‹*C* ∈# *too-heavy-clauses N* (*Some M′*)›
then obtain *x* **where**
[*simp*]: ‹*C* = *pNeg x*› **and**
x: ‹*x* ∈ *simple-clss* (*atms-of-mm N*)› **and**
we: ‹ϱ *M′* ≤ ϱ *x*›
unfolding *too-heavy-clauses-def*
by (*auto simp*: *simple-clss-finite*)
then have ‹*x* ≠ *I*›
using *le* **by** *auto*
then have ‹*set-mset x* ≠ *set-mset I*›
using *distinct-set-mset-eq-iff*[*of x I*] *x dist*
by (*auto simp*: *simple-clss-def*)
then have ‹∃ *a*. ((*a* ∈# *x* ∧ *a* ∉# *I*) ∨ (*a* ∈# *I* ∧ *a* ∉# *x*))›
by *auto*
moreover have *not-incl*: ‹¬*set-mset x* ⊆ *set-mset I*›
using ϱ*-mono*[*of I* ‹*x*›] *we le distinct-set-mset-eq-iff*[*of x I*] *simple-clssE*[*OF x*]
dist cons
by *auto*
moreover have ‹*x* ≠ {#}›
using *we le cons dist not-incl* **by** *auto*
ultimately obtain *L* **where**

L-x: ‹*L* ∈# *x*› **and**
‹*L* ∉# *I*›
by *auto*
moreover have ‹*atms-of x* ⊆ *atms-of I*›
using *simple-clssE*[*OF x*] *tot atm-iff-pos-or-neg-lit*[*of a I*] *atm-iff-pos-or-neg-lit*[*of a x*]
by (*auto dest*!: *multi-member-split*)
ultimately have ‹−*L* ∈# *I*›
using *tot simple-clssE*[*OF x*] *atm-of-notin-atms-of-iff* **by** *auto*
then show ‹*set-mset I* ⊨ *C*›
using *L-x* **by** (*auto simp*: *simple-clss-finite pNeg-def dest*!: *multi-member-split*)
qed
qed

lemma *entails-conflicting-clauses-if-le*:
fixes *M''*
defines ‹*M'* ≡ *lit-of* '# *mset M''*›
assumes
dist: ‹*distinct-mset I*› **and**
cons: ‹*consistent-interp* (*set-mset I*)› **and**
tot: ‹*atms-of I* = *atms-of-mm N*› **and**
le: ‹*Found* (ϱ *I*) < ϱ' (*Some M'*)› **and**
‹*is-improving-int M M'' N w*›
shows
‹*set-mset I* ⊨*m conflicting-clauses N* (*Some* (*lit-of* '# *mset M''*))›
apply (*rule entails-too-heavy-clauses-too-heavy-clauses*[*OF cons*])
subgoal
using *assms* **unfolding** *is-improving-int-def*
by (*auto simp*: *total-over-m-alt-def M'-def atms-of-def lit-in-set-iff-atm*
atms-too-heavy-clauses-Some eq-commute[*of - ‹atms-of-mm N›*]
dest: *multi-member-split dest*!: *simple-clssE*)
by (*use assms entails-too-heavy-clauses-if-le*[*OF assms*(*2−5*)] **in**
‹*auto simp*: *M'-def lits-of-def image-image is-improving-int-def dest*!: *simple-clssE*›)

end

locale *conflict-driven-clause-learning*$_W$*-optimal-weight* =
conflict-driven-clause-learning$_W$
state-eq
state
— functions for the state:
— access functions:
trail init-clss learned-clss conflicting
— changing state:
cons-trail tl-trail add-learned-cls remove-cls
update-conflicting
— get state:
init-state +
ocdcl-weight ϱ
for
state-eq :: ‹*'st* ⇒ *'st* ⇒ *bool*› (**infix** ‹∼› *50*) **and**
state :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* × *'v clauses* × *'v clauses* × *'v clause option* ×
'v clause option × *'b* **and**
trail :: ‹*'st* ⇒ (*'v*, *'v clause*) *ann-lits*› **and**
init-clss :: ‹*'st* ⇒ *'v clauses*› **and**
learned-clss :: ‹*'st* ⇒ *'v clauses*› **and**

conflicting :: ‹′*st* ⇒ ′*v clause option*› **and**

cons-trail :: ‹(′*v*, ′*v clause*) *ann-lit* ⇒ ′*st* ⇒ ′*st*› **and**
tl-trail :: ‹′*st* ⇒ ′*st*› **and**
add-learned-cls :: ‹′*v clause* ⇒ ′*st* ⇒ ′*st*› **and**
remove-cls :: ‹′*v clause* ⇒ ′*st* ⇒ ′*st*› **and**
update-conflicting :: ‹′*v clause option* ⇒ ′*st* ⇒ ′*st*› **and**
init-state :: ‹′*v clauses* ⇒ ′*st*› **and**
ϱ :: ‹′*v clause* ⇒ ′*a* :: {*linorder*}› +
fixes
update-additional-info :: ‹′*v clause option* × ′*b* ⇒ ′*st* ⇒ ′*st*›
assumes
update-additional-info:
‹*state S* = (*M*, *N*, *U*, *C*, *K*) ⟹ *state* (*update-additional-info K′ S*) = (*M*, *N*, *U*, *C*, *K′*)› **and**
weight-init-state:
‹⋀*N* :: ′*v clauses*. *fst* (*additional-info* (*init-state N*)) = *None*›
begin

definition *update-weight-information* :: ‹(′*v*, ′*v clause*) *ann-lits* ⇒ ′*st* ⇒ ′*st*› **where**
‹*update-weight-information M S* =
update-additional-info (*Some* (*lit-of* ‘# *mset M*), *snd* (*additional-info S*)) *S*›

lemma
trail-update-additional-info[*simp*]: ‹*trail* (*update-additional-info w S*) = *trail S*› **and**
init-clss-update-additional-info[*simp*]:
‹*init-clss* (*update-additional-info w S*) = *init-clss S*› **and**
learned-clss-update-additional-info[*simp*]:
‹*learned-clss* (*update-additional-info w S*) = *learned-clss S*› **and**
backtrack-lvl-update-additional-info[*simp*]:
‹*backtrack-lvl* (*update-additional-info w S*) = *backtrack-lvl S*› **and**
conflicting-update-additional-info[*simp*]:
‹*conflicting* (*update-additional-info w S*) = *conflicting S*› **and**
clauses-update-additional-info[*simp*]:
‹*clauses* (*update-additional-info w S*) = *clauses S*›
using *update-additional-info*[*of S*] **unfolding** *clauses-def*
by (*subst* (*asm*) *state-prop*; *subst* (*asm*) *state-prop*; *auto*; *fail*)+

lemma
trail-update-weight-information[*simp*]:
‹*trail* (*update-weight-information w S*) = *trail S*› **and**
init-clss-update-weight-information[*simp*]:
‹*init-clss* (*update-weight-information w S*) = *init-clss S*› **and**
learned-clss-update-weight-information[*simp*]:
‹*learned-clss* (*update-weight-information w S*) = *learned-clss S*› **and**
backtrack-lvl-update-weight-information[*simp*]:
‹*backtrack-lvl* (*update-weight-information w S*) = *backtrack-lvl S*› **and**
conflicting-update-weight-information[*simp*]:
‹*conflicting* (*update-weight-information w S*) = *conflicting S*› **and**
clauses-update-weight-information[*simp*]:
‹*clauses* (*update-weight-information w S*) = *clauses S*›
using *update-additional-info*[*of S*] **unfolding** *update-weight-information-def* **by** *auto*

definition *weight* :: ‹′*st* ⇒ ′*v clause option*› **where**
‹*weight S* = *fst* (*additional-info S*)›

lemma

additional-info-update-additional-info[*simp*]:
‹*additional-info* (*update-additional-info w S*) = *w*›
unfolding *additional-info-def* **using** *update-additional-info*[*of S*]
by (*cases* ‹*state S*›; *auto*; *fail*)+

lemma
weight-cons-trail2[*simp*]: ‹*weight* (*cons-trail L S*) = *weight S*› **and**
clss-tl-trail2[*simp*]: ‹*weight* (*tl-trail S*) = *weight S*› **and**
weight-add-learned-cls-unfolded:
‹*weight* (*add-learned-cls U S*) = *weight S*›
and
weight-update-conflicting2[*simp*]: ‹*weight* (*update-conflicting D S*) = *weight S*› **and**
weight-remove-cls2[*simp*]:
‹*weight* (*remove-cls C S*) = *weight S*› **and**
weight-add-learned-cls2[*simp*]:
‹*weight* (*add-learned-cls C S*) = *weight S*› **and**
weight-update-weight-information2[*simp*]:
‹*weight* (*update-weight-information M S*) = *Some* (*lit-of* ‘# *mset M*)›
by (*auto simp*: *update-weight-information-def weight-def*)

sublocale *conflict-driven-clause-learning-with-adding-init-clause-bnb*$_W$*-no-state*
where
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state* **and**
weight = *weight* **and**
update-weight-information = *update-weight-information* **and**
is-improving-int = *is-improving-int* **and**
conflicting-clauses = *conflicting-clauses*
by *unfold-locales*

lemma *state-additional-info'*:
‹*state S* = (*trail S*, *init-clss S*, *learned-clss S*, *conflicting S*, *weight S*, *additional-info' S*)›
unfolding *additional-info'-def* **by** (*cases* ‹*state S*›; *auto simp*: *state-prop weight-def*)

lemma *state-update-weight-information*:
‹*state S* = (*M*, *N*, *U*, *C*, *w*, *other*) ⟹
∃ *w'*. *state* (*update-weight-information T S*) = (*M*, *N*, *U*, *C*, *w'*, *other*)›
unfolding *update-weight-information-def* **by** (*cases* ‹*state S*›; *auto simp*: *state-prop weight-def*)

lemma *atms-of-init-clss-conflicting-clauses*[*simp*]:
‹*atms-of-mm* (*init-clss S*) ∪ *atms-of-mm* (*conflicting-clss S*) = *atms-of-mm* (*init-clss S*)›
using *conflicting-clss-incl-init-clauses*[*of* ‹(*init-clss S*)›] **unfolding** *conflicting-clss-def* **by** *blast*

lemma *lit-of-trail-in-simple-clss*: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*) ⟹
lit-of ‘# *mset* (*trail S*) ∈ *simple-clss* (*atms-of-mm* (*init-clss S*))›
unfolding *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def abs-state-def*

$cdcl_W$-restart-mset.cdcl$_W$-M-level-inv-def cdcl$_W$-restart-mset.no-strange-atm-def
by (*auto simp*: *simple-clss-def cdcl$_W$-restart-mset-state atms-of-def pNeg-def lits-of-def*
dest: *no-dup-not-tautology no-dup-distinct*)

lemma *pNeg-lit-of-trail-in-simple-clss*: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*) $\Longrightarrow$
pNeg (*lit-of* '# *mset* (*trail S*)) $\in$ *simple-clss* (*atms-of-mm* (*init-clss S*))›
unfolding *cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv-def abs-state-def*
cdcl$_W$-restart-mset.cdcl$_W$-M-level-inv-def cdcl$_W$-restart-mset.no-strange-atm-def
by (*auto simp*: *simple-clss-def cdcl$_W$-restart-mset-state atms-of-def pNeg-def lits-of-def*
dest: *no-dup-not-tautology-uminus no-dup-distinct-uminus*)

lemma *conflict-clss-update-weight-no-alien*:
‹*atms-of-mm* (*conflicting-clss* (*update-weight-information M S*))
$\subseteq$ *atms-of-mm* (*init-clss S*)›
by (*auto simp*: *conflicting-clss-def conflicting-clauses-def atms-of-ms-def*
cdcl$_W$-restart-mset-state simple-clss-finite
dest: *simple-clssE*)

sublocale *state$_W$-no-state*
where
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
by *unfold-locales*

sublocale *state$_W$-no-state*
where
state-eq = *state-eq* **and**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
by *unfold-locales*

sublocale *conflict-driven-clause-learning$_W$*
where
state-eq = *state-eq* **and**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**

conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
by *unfold-locales*

lemma *is-improving-conflicting-clss-update-weight-information'*: ‹*is-improving M M' S* ⟹
conflicting-clss S ⊆# *conflicting-clss* (*update-weight-information M' S*)›
using *is-improving-conflicting-clss-update-weight-information*[*of M M'* ‹*init-clss S*› ‹*weight S*›]
unfolding *conflicting-clss-def*
by *auto*

lemma *conflicting-clss-update-weight-information-in2'*:
assumes ‹*is-improving M M' S*›
shows ‹*negate-ann-lits M'* ∈# *conflicting-clss* (*update-weight-information M' S*)›
using *conflicting-clss-update-weight-information-in2*[*of M M'* ‹*init-clss S*› ‹*weight S*›] *assms*
unfolding *conflicting-clss-def*
by *auto*

sublocale *conflict-driven-clause-learning-with-adding-init-clause-bnb*$_W$*-ops*
where
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state* **and**
weight = *weight* **and**
update-weight-information = *update-weight-information* **and**
is-improving-int = *is-improving-int* **and**
conflicting-clauses = *conflicting-clauses*
apply *unfold-locales*
subgoal by (*rule state-additional-info'*)
subgoal by (*rule state-update-weight-information*)
subgoal unfolding *conflicting-clss-def* **by** (*rule conflicting-clss-incl-init-clauses*)
subgoal unfolding *conflicting-clss-def* **by** (*rule distinct-mset-mset-conflicting-clss2*)
subgoal by (*rule is-improving-conflicting-clss-update-weight-information'*)
subgoal by (*rule conflicting-clss-update-weight-information-in2'*; *assumption*)
done

lemma *wf-cdcl-bnb-fixed*:
‹*wf* {(*T*, *S*). *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*) ∧ *cdcl-bnb S T*
∧ *init-clss S* = *N*}›
apply (*rule wf-cdcl-bnb*[*of N id* ‹{(*I'*, *I*). *I'* ≠ *None* ∧
(*the I'*) ∈ *simple-clss* (*atms-of-mm N*) ∧ (*ϱ' I'*, *ϱ' I*) ∈ {(*j*, *i*). *j* < *i*}}›])
subgoal for *S T*
by (*cases* ‹*weight S*›; *cases* ‹*weight T*›)
(*auto simp*: *improvep.simps is-improving-int-def split*: *enat.splits*)

```
  subgoal
    apply (rule wf-finite-segments)
    subgoal by (auto simp: irrefl-def)
    subgoal
      apply (auto simp: irrefl-def trans-def intro: less-trans[of ‹Found -› ‹Found -›])
      apply (rule less-trans[of ‹Found -› ‹Found -›])
      apply auto
      done
    subgoal for x
      by (subgoal-tac ‹{y. (y, x)
         ∈ {(I', I). I' ≠ None ∧ the I' ∈ simple-clss (atms-of-mm N) ∧
             (ϱ' I', ϱ' I) ∈ {(j, i). j < i}}} =
        Some ‘ {y. (y, x) ∈ {(I', I).
            I' ∈ simple-clss (atms-of-mm N) ∧
             (ϱ' (Some I'), ϱ' I) ∈ {(j, i). j < i}}}›)
       (auto simp: finite-image-iff intro: finite-subset[OF - simple-clss-finite[of ‹atms-of-mm N›]])
    done
  done

lemma wf-cdcl-bnb2:
  ‹wf {(T, S). cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state S)
     ∧ cdcl-bnb S T}›
  by (subst wf-cdcl-bnb-fixed-iff[symmetric]) (intro allI, rule wf-cdcl-bnb-fixed)

lemma can-always-improve:
  assumes
    ent: ‹trail S ⊨asm clauses S› and
    total: ‹total-over-m (lits-of-l (trail S)) (set-mset (clauses S))› and
    n-s: ‹no-step conflict-opt S› and
    confl[simp]: ‹conflicting S = None› and
    all-struct: ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state S)›
   shows ‹Ex (improvep S)›
proof –
  have H: ‹(lit-of ‘# mset (trail S)) ∈# mset-set (simple-clss (atms-of-mm (init-clss S)))›
    ‹(lit-of ‘# mset (trail S)) ∈ simple-clss (atms-of-mm (init-clss S))›
    ‹no-dup (trail S)›
    apply (subst finite-set-mset-mset-set[OF simple-clss-finite])
    using all-struct by (auto simp: simple-clss-def cdcl_W-restart-mset.cdcl_W-all-struct-inv-def
        no-strange-atm-def atms-of-def lits-of-def image-image
        cdcl_W-M-level-inv-def clauses-def
      dest: no-dup-not-tautology no-dup-distinct)
  then have le: ‹Found (ϱ (lit-of ‘# mset (trail S))) < ϱ' (weight S)›
    using n-s total
    by (auto simp: conflict-opt.simps conflicting-clss-def lits-of-def
         conflicting-clauses-def clauses-def negate-ann-lits-pNeg-lit-of simple-clss-finite
       dest: not-entailed-too-heavy-clauses-ge)
  have tr: ‹trail S ⊨asm init-clss S›
    using ent by (auto simp: clauses-def)
  have tot': ‹total-over-m (lits-of-l (trail S)) (set-mset (init-clss S))›
    using total all-struct by (auto simp: total-over-m-def total-over-set-def
       cdcl_W-all-struct-inv-def clauses-def no-strange-atm-def)
  have M': ‹ϱ (lit-of ‘# mset M') = ϱ (lit-of ‘# mset (trail S))›
      if ‹total-over-m (lits-of-l M') (set-mset (init-clss S))› and
        incl: ‹mset (trail S) ⊆# mset M'› and
        ‹lit-of ‘# mset M' ∈ simple-clss (atms-of-mm (init-clss S))›
      for M'
```

proof −
have [*simp*]: ‹*lits-of-l M′* = *set-mset* (*lit-of* ‘# *mset M′*)›
by (*auto simp*: *lits-of-def*)
obtain *A* **where** *A*: ‹*mset M′* = *A* + *mset* (*trail S*)›
using *incl* **by** (*auto simp*: *mset-subset-eq-exists-conv*)
have *M′*: ‹*lits-of-l M′* = *lit-of* ‘ *set-mset A* ∪ *lits-of-l* (*trail S*)›
unfolding *lits-of-def*
by (*metis A image-Un set-mset-mset set-mset-union*)
have ‹*mset M′* = *mset* (*trail S*)›
using *that tot′ total* **unfolding** *A total-over-m-alt-def*
apply (*case-tac A*)
apply (*auto simp*: *A simple-clss-def distinct-mset-add M′ image-Un tautology-union mset-inter-empty-set-mset atms-of-def atms-of-s-def atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set image-image tautology-add-mset*)
by (*metis* (*no-types*, *lifting*) *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set subsetCE lits-of-def*)
then show *?thesis*
using *total* **by** *auto*
qed
have ‹*is-improving* (*trail S*) (*trail S*) *S*›
if ‹*Found* (*ϱ* (*lit-of* ‘# *mset* (*trail S*))) < *ϱ′* (*weight S*)›
using *that total H confl tr tot′ M′* **unfolding** *is-improving-int-def lits-of-def* **by** *fast*
then show ‹*Ex* (*improvep S*)›
using *improvep.intros*[*of S* ‹*trail S*› ‹*update-weight-information* (*trail S*) *S*›] *le confl* **by** *fast*
qed

lemma *no-step-cdcl-bnb-stgy-empty-conflict2*:
assumes
n-s: ‹*no-step cdcl-bnb S*› **and**
all-struct: ‹$cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv* (*abs-state S*)› **and**
stgy-inv: ‹*cdcl-bnb-stgy-inv S*›
shows ‹*conflicting S* = *Some* {#}›
by (*rule no-step-cdcl-bnb-stgy-empty-conflict*[*OF can-always-improve assms*])

lemma *cdcl-bnb-larger-still-larger*:
assumes
‹*cdcl-bnb S T*›
shows ‹*ϱ′* (*weight S*) ≥ *ϱ′* (*weight T*)›
using *assms* **apply** (*cases rule*: *cdcl-bnb.cases*)
by (*auto simp*: *improvep.simps is-improving-int-def conflict-opt.simps* $ocdcl_W$*-o.simps cdcl-bnb-bj.simps skip.simps resolve.simps obacktrack.simps elim*: *rulesE*)

lemma *obacktrack-model-still-model*:
assumes
‹*obacktrack S T*› **and**
all-struct: ‹$cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv* (*abs-state S*)› **and**
ent: ‹*set-mset I* ⊨*sm clauses S*› ‹*set-mset I* ⊨*sm conflicting-clss S*› **and**
dist: ‹*distinct-mset I*› **and**
cons: ‹*consistent-interp* (*set-mset I*)› **and**
tot: ‹*atms-of I* = *atms-of-mm* (*init-clss S*)› **and**
opt-struct: ‹*cdcl-bnb-struct-invs S*› **and**
le: ‹*Found* (*ϱ I*) < *ϱ′* (*weight T*)›
shows
‹*set-mset I* ⊨*sm clauses T* ∧ *set-mset I* ⊨*sm conflicting-clss T*›

using *assms*(*1*)
proof (*cases rule*: *obacktrack.cases*)
case (*obacktrack-rule L D K M1 M2 D′ i*) **note** *confl* = *this*(*1*) **and** *DD′* = *this*(*7*) **and** *clss-L-D′* = *this*(*8*) **and** *T* = *this*(*9*)
have *H*: ‹*total-over-m I* (*set-mset* (*clauses S* + *conflicting-clss S*) ∪ {*add-mset L D′*}) ⟹
consistent-interp I ⟹
I ⊨*sm clauses S* + *conflicting-clss S* ⟹ *I* ⊨ *add-mset L D′*› **for** *I*
using *clss-L-D′*
unfolding *true-clss-cls-def*
by *blast*
have *alien*: ‹$cdcl_W$*-restart-mset.no-strange-atm* (*abs-state S*)›
using *all-struct* **unfolding** $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*
by *fast+*
have ‹*total-over-m* (*set-mset I*) (*set-mset* (*init-clss S*))›
using *tot*[*symmetric*]
by (*auto simp*: *total-over-m-def total-over-set-def atm-iff-pos-or-neg-lit*)

then have *1*: ‹*total-over-m* (*set-mset I*) (*set-mset* (*clauses S* + *conflicting-clss S*) ∪ {*add-mset L D′*})›
using *alien T confl tot DD′ opt-struct*
unfolding $cdcl_W$*-restart-mset.no-strange-atm-def total-over-m-def total-over-set-def*
apply (*auto simp*: $cdcl_W$*-restart-mset-state abs-state-def atms-of-def clauses-def cdcl-bnb-struct-invs-def dest*: *multi-member-split*)
by *blast*
have *2*: ‹*set-mset I* ⊨*sm conflicting-clss S*›
using *tot cons ent*(*2*) **by** *auto*
have ‹*set-mset I* ⊨ *add-mset L D′*›
using *H*[*OF 1 cons*] *2 ent* **by** *auto*
then show *?thesis*
using *ent obacktrack-rule 2* **by** *auto*
qed

lemma *entails-conflicting-clauses-if-le′*:
fixes *M″*
defines ‹*M′* ≡ *lit-of* ‘# *mset M″*›
assumes
dist: ‹*distinct-mset I*› **and**
cons: ‹*consistent-interp* (*set-mset I*)› **and**
tot: ‹*atms-of I* = *atms-of-mm* (*init-clss S*)› **and**
le: ‹*Found* (ϱ *I*) < ϱ′ (*Some M′*)› **and**
‹*is-improving M M″ S*› **and**
‹*N* = *init-clss S*›
shows
‹*set-mset I* ⊨*m conflicting-clauses N* (*weight* (*update-weight-information M″ S*))›
using *entails-conflicting-clauses-if-le*[*OF assms*(*2*−*6*)[*unfolded M′-def*]] *assms*(*7*)
unfolding *conflicting-clss-def* **by** *auto*

lemma *improve-model-still-model*:
assumes
‹*improvep S T*› **and**
all-struct: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state S*)› **and**
ent: ‹*set-mset I* ⊨*sm clauses S*› ‹*set-mset I* ⊨*sm conflicting-clss S*› **and**
dist: ‹*distinct-mset I*› **and**
cons: ‹*consistent-interp* (*set-mset I*)› **and**
tot: ‹*atms-of I* = *atms-of-mm* (*init-clss S*)› **and**

```
    opt-struct: ‹cdcl-bnb-struct-invs S› and
    le: ‹Found (ϱ I) < ϱ' (weight T)›
  shows
    ‹set-mset I ⊨sm clauses T ∧ set-mset I ⊨sm conflicting-clss T›
  using assms(1)
proof (cases rule: improvep.cases)
  case (improve-rule M') note imp = this(1) and confl = this(2) and T = this(3)
  have alien: ‹cdcl_W-restart-mset.no-strange-atm (abs-state S)› and
    lev: ‹cdcl_W-restart-mset.cdcl_W-M-level-inv (abs-state S)›
    using all-struct unfolding cdcl_W-restart-mset.cdcl_W-all-struct-inv-def
    by fast+
  then have atm-trail: ‹atms-of (lit-of `# mset (trail S)) ⊆ atms-of-mm (init-clss S)›
    using alien by (auto simp: no-strange-atm-def lits-of-def atms-of-def)
  have dist2: ‹distinct-mset (lit-of `# mset (trail S))› and
    taut2: ‹¬ tautology (lit-of `# mset (trail S))›
    using lev unfolding cdcl_W-restart-mset.cdcl_W-M-level-inv-def
    by (auto dest: no-dup-distinct no-dup-not-tautology)
  have tot2: ‹total-over-m (set-mset I) (set-mset (init-clss S))›
    using tot[symmetric]
    by (auto simp: total-over-m-def total-over-set-def atm-iff-pos-or-neg-lit)
  have atm-trail: ‹atms-of (lit-of `# mset M') ⊆ atms-of-mm (init-clss S)› and
    dist2: ‹distinct-mset (lit-of `# mset M')› and
    taut2: ‹¬ tautology (lit-of `# mset M')›
    using imp by (auto simp: no-strange-atm-def lits-of-def atms-of-def is-improving-int-def
      simple-clss-def)

  have tot2: ‹total-over-m (set-mset I) (set-mset (init-clss S))›
    using tot[symmetric]
    by (auto simp: total-over-m-def total-over-set-def atm-iff-pos-or-neg-lit)
  have
      ‹set-mset I ⊨m conflicting-clauses (init-clss S) (weight (update-weight-information M' S))›
    apply (rule entails-conflicting-clauses-if-le'[unfolded conflicting-clss-def])
    using T dist cons tot le imp by (auto intro!: )
  then have ‹set-mset I ⊨m conflicting-clss (update-weight-information M' S)›
    by (auto simp: update-weight-information-def conflicting-clss-def)
  then show ?thesis
    using ent improve-rule T by auto
qed

lemma cdcl-bnb-still-model:
  assumes
    ‹cdcl-bnb S T› and
    all-struct: ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state S)› and
    ent: ‹set-mset I ⊨sm clauses S› ‹set-mset I ⊨sm conflicting-clss S› and
    dist: ‹distinct-mset I› and
    cons: ‹consistent-interp (set-mset I)› and
    tot: ‹atms-of I = atms-of-mm (init-clss S)› and
    opt-struct: ‹cdcl-bnb-struct-invs S›
  shows
    ‹(set-mset I ⊨sm clauses T ∧ set-mset I ⊨sm conflicting-clss T) ∨ Found (ϱ I) ≥ ϱ' (weight T)›
  using assms
proof (cases rule: cdcl-bnb.cases)
  case cdcl-improve
  from improve-model-still-model[OF this all-struct ent dist cons tot opt-struct]
  show ?thesis
    by (auto simp: improvep.simps)
```

next
  case *cdcl-other'*
  then show *?thesis*
  proof (*induction rule*: *ocdcl$_W$-o-all-rules-induct*)
    case (*backtrack T*)
    from *obacktrack-model-still-model*[*OF this all-struct ent dist cons tot opt-struct*]
    show *?case*
      by *auto*
  qed (*use ent* **in** ‹*auto elim*: *rulesE*›)
qed (*auto simp*: *conflict-opt.simps elim*: *rulesE*)

lemma *rtranclp-cdcl-bnb-still-model*:
  assumes
    st: ‹*cdcl-bnb*** *S T*› **and**
    all-struct: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)› **and**
    ent: ‹(*set-mset I* $\models$*sm clauses S* $\wedge$ *set-mset I* $\models$*sm conflicting-clss S*) $\vee$ *Found* (ϱ *I*) $\geq$ ϱ' (*weight S*)› **and**
    dist: ‹*distinct-mset I*› **and**
    cons: ‹*consistent-interp* (*set-mset I*)› **and**
    tot: ‹*atms-of I* = *atms-of-mm* (*init-clss S*)› **and**
    opt-struct: ‹*cdcl-bnb-struct-invs S*›
  shows
    ‹(*set-mset I* $\models$*sm clauses T* $\wedge$ *set-mset I* $\models$*sm conflicting-clss T*) $\vee$ *Found* (ϱ *I*) $\geq$ ϱ' (*weight T*)›
  using *st*
proof (*induction rule*: *rtranclp-induct*)
  case *base*
  then show *?case*
    using *ent* **by** *auto*
next
  case (*step T U*) **note** *star* = *this*(*1*) **and** *st* = *this*(*2*) **and** *IH* = *this*(*3*)
  have *1*: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state T*)›
    using *rtranclp-cdcl-bnb-stgy-all-struct-inv*[*OF star all-struct*] **.**

  have *2*: ‹*cdcl-bnb-struct-invs T*›
    using *rtranclp-cdcl-bnb-cdcl-bnb-struct-invs*[*OF star opt-struct*] **.**
  have *3*: ‹*atms-of I* = *atms-of-mm* (*init-clss T*)›
    using *tot rtranclp-cdcl-bnb-no-more-init-clss*[*OF star*] **by** *auto*
  show *?case*
    using *cdcl-bnb-still-model*[*OF st 1 - - dist cons 3 2*] *IH*
      cdcl-bnb-larger-still-larger[*OF st*]
    using *dual-order.trans* **by** *blast*
qed

lemma *full-cdcl-bnb-stgy-larger-or-equal-weight*:
  assumes
    st: ‹*full cdcl-bnb-stgy S T*› **and**
    all-struct: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)› **and**
    ent: ‹(*set-mset I* $\models$*sm clauses S* $\wedge$ *set-mset I* $\models$*sm conflicting-clss S*) $\vee$ *Found* (ϱ *I*) $\geq$ ϱ' (*weight S*)› **and**
    dist: ‹*distinct-mset I*› **and**
    cons: ‹*consistent-interp* (*set-mset I*)› **and**
    tot: ‹*atms-of I* = *atms-of-mm* (*init-clss S*)› **and**
    opt-struct: ‹*cdcl-bnb-struct-invs S*› **and**
    stgy-inv: ‹*cdcl-bnb-stgy-inv S*›
  shows
    ‹*Found* (ϱ *I*) $\geq$ ϱ' (*weight T*)› **and**

‹*unsatisfiable* (*set-mset* (*clauses* *T* + *conflicting-clss* *T*))›
proof −
have *ns*: ‹*no-step cdcl-bnb-stgy T*› **and**
st: ‹*cdcl-bnb-stgy*** *S T*› **and**
st': ‹*cdcl-bnb*** *S T*›
using *st* **unfolding** *full-def* **by** (*auto intro*: *rtranclp-cdcl-bnb-stgy-cdcl-bnb*)
have *ns'*: ‹*no-step cdcl-bnb T*›
by (*meson cdcl-bnb.cases cdcl-bnb-stgy.simps no-confl-prop-impr.elims*(*3*) *ns*)
have *struct-T*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state T*)›
using *rtranclp-cdcl-bnb-stgy-all-struct-inv*[*OF st' all-struct*] **.**
have *stgy-T*: ‹*cdcl-bnb-stgy-inv T*›
using *rtranclp-cdcl-bnb-stgy-stgy-inv*[*OF st all-struct stgy-inv*] **.**
have *confl*: ‹*conflicting T* = *Some* {#}›
using *no-step-cdcl-bnb-stgy-empty-conflict2*[*OF ns' struct-T stgy-T*] **.**

have ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause* (*abs-state T*)› **and**
alien: ‹$cdcl_W$*-restart-mset.no-strange-atm* (*abs-state T*)›
using *struct-T* **unfolding** $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def* **by** *fast*+
then have *ent'*: ‹*set-mset* (*clauses T* + *conflicting-clss T*) ⊨*p* {#}›
using *confl* **unfolding** $cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause-alt-def*
by *auto*
have *atms-eq*: ‹*atms-of I* ∪ *atms-of-mm* (*conflicting-clss T*) = *atms-of-mm* (*init-clss T*)›
using *tot*[*symmetric*] *atms-of-conflicting-clss*[*of T*] *alien*
unfolding *rtranclp-cdcl-bnb-no-more-init-clss*[*OF st'*] $cdcl_W$*-restart-mset.no-strange-atm-def*
by (*auto simp*: *clauses-def total-over-m-def total-over-set-def atm-iff-pos-or-neg-lit*
abs-state-def $cdcl_W$*-restart-mset-state*)

have ‹¬ (*set-mset I* ⊨*sm clauses T* + *conflicting-clss T*)›
proof
assume *ent''*: ‹*set-mset I* ⊨*sm clauses T* + *conflicting-clss T*›
moreover have ‹*total-over-m* (*set-mset I*) (*set-mset* (*clauses T* + *conflicting-clss T*))›
using *tot*[*symmetric*] *atms-of-conflicting-clss*[*of T*] *alien*
unfolding *rtranclp-cdcl-bnb-no-more-init-clss*[*OF st'*] $cdcl_W$*-restart-mset.no-strange-atm-def*
by (*auto simp*: *clauses-def total-over-m-def total-over-set-def atm-iff-pos-or-neg-lit*
abs-state-def $cdcl_W$*-restart-mset-state atms-eq*)
then show *False*
using *ent' cons ent''* **unfolding** *true-clss-cls-def* **by** *auto*
qed
then show ‹ϱ' (*weight T*) ≤ *Found* (ϱ *I*)›
using *rtranclp-cdcl-bnb-still-model*[*OF st' all-struct ent dist cons tot opt-struct*]
by *auto*

show ‹*unsatisfiable* (*set-mset* (*clauses T* + *conflicting-clss T*))›
proof
assume ‹*satisfiable* (*set-mset* (*clauses T* + *conflicting-clss T*))›
then obtain *I* **where**
ent'': ‹*I* ⊨*sm clauses T* + *conflicting-clss T*› **and**
tot: ‹*total-over-m I* (*set-mset* (*clauses T* + *conflicting-clss T*))› **and**
‹*consistent-interp I*›
unfolding *satisfiable-def*
by *blast*
then show ‹*False*›
using *ent' cons* **unfolding** *true-clss-cls-def* **by** *auto*
qed
qed

lemma *full-cdcl-bnb-stgy-unsat2*:
assumes
st: ‹*full cdcl-bnb-stgy S T*› **and**
all-struct: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)› **and**
opt-struct: ‹*cdcl-bnb-struct-invs S*› **and**
stgy-inv: ‹*cdcl-bnb-stgy-inv S*›
shows
‹*unsatisfiable* (*set-mset* (*clauses T* + *conflicting-clss T*))›
proof −
have *ns*: ‹*no-step cdcl-bnb-stgy T*› **and**
st: ‹*cdcl-bnb-stgy*** *S T*› **and**
st′: ‹*cdcl-bnb*** *S T*›
using *st* **unfolding** *full-def* **by** (*auto intro*: *rtranclp-cdcl-bnb-stgy-cdcl-bnb*)
have *ns*′: ‹*no-step cdcl-bnb T*›
by (*meson cdcl-bnb.cases cdcl-bnb-stgy.simps no-confl-prop-impr.elims*(*3*) *ns*)
have *struct-T*: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state T*)›
using *rtranclp-cdcl-bnb-stgy-all-struct-inv*[*OF st*′ *all-struct*] **.**
have *stgy-T*: ‹*cdcl-bnb-stgy-inv T*›
using *rtranclp-cdcl-bnb-stgy-stgy-inv*[*OF st all-struct stgy-inv*] **.**
have *confl*: ‹*conflicting T* = *Some* {#}›
using *no-step-cdcl-bnb-stgy-empty-conflict2*[*OF ns*′ *struct-T stgy-T*] **.**

have ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-learned-clause* (*abs-state T*)› **and**
alien: ‹*cdcl*$_W$*-restart-mset.no-strange-atm* (*abs-state T*)›
using *struct-T* **unfolding** *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def* **by** *fast*+
then have *ent*′: ‹*set-mset* (*clauses T* + *conflicting-clss T*) ⊨*p* {#}›
using *confl* **unfolding** *cdcl*$_W$*-restart-mset.cdcl*$_W$*-learned-clause-alt-def*
by *auto*

show ‹*unsatisfiable* (*set-mset* (*clauses T* + *conflicting-clss T*))›
proof
assume ‹*satisfiable* (*set-mset* (*clauses T* + *conflicting-clss T*))›
then obtain *I* **where**
ent′′: ‹*I* ⊨*sm clauses T* + *conflicting-clss T*› **and**
tot: ‹*total-over-m I* (*set-mset* (*clauses T* + *conflicting-clss T*))› **and**
‹*consistent-interp I*›
unfolding *satisfiable-def*
by *blast*
then show ‹*False*›
using *ent*′ **unfolding** *true-clss-cls-def* **by** *auto*
qed
qed

lemma *weight-init-state2*[*simp*]: ‹*weight* (*init-state S*) = *None*› **and**
conflicting-clss-init-state[*simp*]:
‹*conflicting-clss* (*init-state N*) = {#}›
unfolding *weight-def conflicting-clss-def conflicting-clauses-def*
by (*auto simp*: *weight-init-state true-clss-cls-tautology-iff simple-clss-finite*
filter-mset-empty-conv mset-set-empty-iff dest: *simple-clssE*)

First part of Theorem 2.15.6 of Weidenbach's book

lemma *full-cdcl-bnb-stgy-no-conflicting-clause-unsat*:
assumes
st: ‹*full cdcl-bnb-stgy S T*› **and**
all-struct: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)› **and**

opt-struct: ‹*cdcl-bnb-struct-invs S*› **and**
stgy-inv: ‹*cdcl-bnb-stgy-inv S*› **and**
[*simp*]: ‹*weight T = None*› **and**
ent: ‹*cdcl*$_W$*-learned-clauses-entailed-by-init S*›
shows ‹*unsatisfiable* (*set-mset* (*init-clss S*))›
proof −
have ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-learned-clauses-entailed-by-init* (*abs-state S*)› **and**
‹*conflicting-clss T* = {#}›
using *ent* **by** (*auto simp*: *cdcl*$_W$*-restart-mset.cdcl*$_W$*-learned-clauses-entailed-by-init-def*
cdcl$_W$*-learned-clauses-entailed-by-init-def abs-state-def cdcl*$_W$*-restart-mset-state*
conflicting-clss-def conflicting-clauses-def true-clss-cls-tautology-iff simple-clss-finite
filter-mset-empty-conv mset-set-empty-iff dest: *simple-clssE*)
then show *?thesis*
using *full-cdcl-bnb-stgy-no-conflicting-clss-unsat*[*OF - st all-struct*
stgy-inv] **by** (*auto simp*: *can-always-improve*)
qed

definition *annotation-is-model* **where**
‹*annotation-is-model S* ⟷
(*weight S* ≠ *None* ⟶ (*set-mset* (*the* (*weight S*)) ⊨*sm init-clss S* ∧
consistent-interp (*set-mset* (*the* (*weight S*))) ∧
atms-of (*the* (*weight S*)) ⊆ *atms-of-mm* (*init-clss S*) ∧
total-over-m (*set-mset* (*the* (*weight S*))) (*set-mset* (*init-clss S*)) ∧
distinct-mset (*the* (*weight S*))))›

lemma *cdcl-bnb-annotation-is-model*:
assumes
‹*cdcl-bnb S T*› **and**
‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*abs-state S*)› **and**
‹*annotation-is-model S*›
shows ‹*annotation-is-model T*›
proof −
have [*simp*]: ‹*atms-of* (*lit-of* '# *mset M*) = *atm-of* ' *lit-of* ' *set M*› **for** *M*
by (*auto simp*: *atms-of-def*)
have ‹*consistent-interp* (*lits-of-l* (*trail S*)) ∧
atm-of ' (*lits-of-l* (*trail S*)) ⊆ *atms-of-mm* (*init-clss S*) ∧
distinct-mset (*lit-of* '# *mset* (*trail S*))›
using *assms*(*2*) **by** (*auto simp*: *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def*
abs-state-def cdcl$_W$*-restart-mset-state cdcl*$_W$*-restart-mset.no-strange-atm-def*
cdcl$_W$*-restart-mset.cdcl*$_W$*-M-level-inv-def*
dest: *no-dup-distinct*)
with *assms*(*1*,*3*)
show *?thesis*
apply (*cases rule*: *cdcl-bnb.cases*)
subgoal
by (*auto simp*: *conflict.simps annotation-is-model-def*)
subgoal
by (*auto simp*: *propagate.simps annotation-is-model-def*)
subgoal
by (*force simp*: *annotation-is-model-def true-annots-true-cls lits-of-def*
improvep.simps is-improving-int-def image-Un image-image simple-clss-def
consistent-interp-tuatology-mset-set
dest!: *consistent-interp-unionD intro*: *distinct-mset-union2*)
subgoal
by (*auto simp*: *annotation-is-model-def conflict-opt.simps*)
subgoal

by (*auto simp*: *annotation-is-model-def*
$ocdcl_W$-o.simps cdcl-bnb-bj.simps obacktrack.simps
skip.simps resolve.simps decide.simps)
done
qed

lemma *rtranclp-cdcl-bnb-annotation-is-model*:
‹*cdcl-bnb*** *S T* ⟹ *$cdcl_W$-restart-mset.$cdcl_W$-all-struct-inv* (*abs-state S*) ⟹
annotation-is-model S ⟹ *annotation-is-model T*›
by (*induction rule*: *rtranclp-induct*)
(*auto simp*: *cdcl-bnb-annotation-is-model rtranclp-cdcl-bnb-stgy-all-struct-inv*)

Theorem 2.15.6 of Weidenbach's book

theorem *full-cdcl-bnb-stgy-no-conflicting-clause-from-init-state*:
assumes
st: ‹*full cdcl-bnb-stgy* (*init-state N*) *T*› **and**
dist: ‹*distinct-mset-mset N*›
shows
‹*weight T* = *None* ⟹ *unsatisfiable* (*set-mset N*)› (**is** ‹*?B* ⟹ *?A*›) **and**
‹*weight T* ≠ *None* ⟹ *consistent-interp* (*set-mset* (*the* (*weight T*))) ∧
atms-of (*the* (*weight T*)) ⊆ *atms-of-mm N* ∧ *set-mset* (*the* (*weight T*)) ⊨*sm N* ∧
total-over-m (*set-mset* (*the* (*weight T*))) (*set-mset N*) ∧
distinct-mset (*the* (*weight T*))› **and**
‹*distinct-mset I* ⟹ *consistent-interp* (*set-mset I*) ⟹ *atms-of I* = *atms-of-mm N* ⟹
set-mset I ⊨*sm N* ⟹ *Found* (ϱ *I*) ≥ ϱ′ (*weight T*)›
proof −
let *?S* = ‹*init-state N*›
have ‹*distinct-mset C*› **if** ‹*C* ∈# *N*› **for** *C*
using *dist that* **by** (*auto simp*: *distinct-mset-set-def dest*: *multi-member-split*)
then have *dist*: ‹*distinct-mset-mset N*›
by (*auto simp*: *distinct-mset-set-def*)
then have [*simp*]: ‹*$cdcl_W$-restart-mset.$cdcl_W$-all-struct-inv* ([], *N*, {#}, *None*)›
unfolding *init-state.simps*[*symmetric*]
by (*auto simp*: *$cdcl_W$-restart-mset.$cdcl_W$-all-struct-inv-def*)
moreover have [*iff*]: ‹*cdcl-bnb-struct-invs ?S*› **and** [*simp*]: ‹*cdcl-bnb-stgy-inv ?S*›
by (*auto simp*: *cdcl-bnb-struct-invs-def conflict-is-false-with-level-def cdcl-bnb-stgy-inv-def*)
moreover have *ent*: ‹*$cdcl_W$-learned-clauses-entailed-by-init ?S*›
by (*auto simp*: *$cdcl_W$-learned-clauses-entailed-by-init-def*)
moreover have [*simp*]: ‹*$cdcl_W$-restart-mset.$cdcl_W$-all-struct-inv* (*abs-state* (*init-state N*))›
unfolding *CDCL-W-Abstract-State.init-state.simps abs-state-def*
by *auto*
ultimately show ‹*weight T* = *None* ⟹ *unsatisfiable* (*set-mset N*)›
using *full-cdcl-bnb-stgy-no-conflicting-clause-unsat*[*OF st*]
by *auto*
have ‹*annotation-is-model ?S*›
by (*auto simp*: *annotation-is-model-def*)
then have ‹*annotation-is-model T*›
using *rtranclp-cdcl-bnb-annotation-is-model*[*of ?S T*] *st*
unfolding *full-def* **by** (*auto dest*: *rtranclp-cdcl-bnb-stgy-cdcl-bnb*)
moreover have ‹*init-clss T* = *N*›
using *rtranclp-cdcl-bnb-no-more-init-clss*[*of ?S T*] *st*
unfolding *full-def* **by** (*auto dest*: *rtranclp-cdcl-bnb-stgy-cdcl-bnb*)
ultimately show ‹*weight T* ≠ *None* ⟹ *consistent-interp* (*set-mset* (*the* (*weight T*))) ∧
atms-of (*the* (*weight T*)) ⊆ *atms-of-mm N* ∧ *set-mset* (*the* (*weight T*)) ⊨*sm N* ∧
total-over-m (*set-mset* (*the* (*weight T*))) (*set-mset N*) ∧
distinct-mset (*the* (*weight T*))›

```
    by (auto simp: annotation-is-model-def)

  show ‹distinct-mset I ⟹ consistent-interp (set-mset I) ⟹ atms-of I = atms-of-mm N ⟹
      set-mset I ⊨sm N ⟹ Found (ϱ I) ≥ ϱ' (weight T)›
    using full-cdcl-bnb-stgy-larger-or-equal-weight[of ?S T I] st unfolding full-def
    by auto
qed

lemma pruned-clause-in-conflicting-clss:
  assumes
    ge: ‹⋀M'. total-over-m (set-mset (mset (M @ M'))) (set-mset (init-clss S)) ⟹
      distinct-mset (atm-of `# mset (M @ M')) ⟹
      consistent-interp (set-mset (mset (M @ M'))) ⟹
      Found (ϱ (mset (M @ M'))) ≥ ϱ' (weight S)› and
    atm: ‹atms-of (mset M) ⊆ atms-of-mm (init-clss S)› and
    dist: ‹distinct M› and
    cons: ‹consistent-interp (set M)›
  shows ‹pNeg (mset M) ∈# conflicting-clss S›
proof -
  have 0: ‹(pNeg o mset o ((@) M))` {M'.
      distinct-mset (atm-of `# mset (M @ M')) ∧ consistent-interp (set-mset (mset (M @ M'))) ∧
      atms-of-s (set (M @ M')) ⊆ (atms-of-mm (init-clss S)) ∧
      card (atms-of-mm (init-clss S)) = n + card (atms-of (mset (M @ M')))} ⊆
    set-mset (conflicting-clss S)› (is ‹- ` ?A n ⊆ ?H›)for n
  proof (induction n)
    case 0
    show ?case
    proof clarify
      fix x :: ‹'v literal multiset› and xa :: ‹'v literal multiset› and
        xb :: ‹'v literal list› and xc :: ‹'v literal list›
      assume
        dist: ‹distinct-mset (atm-of `# mset (M @ xc))› and
        cons: ‹consistent-interp (set-mset (mset (M @ xc)))› and
        atm': ‹atms-of-s (set (M @ xc)) ⊆ atms-of-mm (init-clss S)› and
        0: ‹card (atms-of-mm (init-clss S)) = 0 + card (atms-of (mset (M @ xc)))›
      have D[dest]:
        ‹A ∈ set M ⟹ A ∉ set xc› ‹A ∈ set M ⟹ −A ∉ set xc›
        for A
        using dist multi-member-split[of A ‹mset M›] multi-member-split[of ‹−A› ‹mset xc›]
          multi-member-split[of ‹−A› ‹mset M›] multi-member-split[of ‹A› ‹mset xc›]
        by (auto simp: atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set)
      have dist2: ‹distinct xc› ‹distinct-mset (atm-of `# mset xc)›
        ‹distinct-mset (mset M + mset xc)›
        using dist distinct-mset-atm-ofD[OF dist]
        unfolding mset-append[symmetric] distinct-mset-mset-distinct
        by (auto dest: distinct-mset-union2 distinct-mset-atm-ofD)
      have eq: ‹card (atms-of-s (set M) ∪ atms-of-s (set xc)) =
        card (atms-of-s (set M)) + card (atms-of-s (set xc))›
            by (subst card-Un-Int) auto
      let ?M = ‹M @ xc›

      have H1: ‹atms-of-s (set ?M) = atms-of-mm (init-clss S)›
        using eq atm card-mono[OF - atm'] card-subset-eq[OF - atm'] 0
        by (auto simp: atms-of-s-def image-Un)
      moreover have tot2: ‹total-over-m (set ?M) (set-mset (init-clss S))›
        using H1 by (auto simp: total-over-m-def total-over-set-def lit-in-set-iff-atm)
```

moreover have ‹¬*tautology* (*mset ?M*)›
using *cons* **unfolding** *consistent-interp-tautology*[*symmetric*]
by *auto*
ultimately have ‹*mset ?M* ∈ *simple-clss* (*atms-of-mm* (*init-clss S*))›
using *dist atm cons H1 dist2*
by (*auto simp*: *simple-clss-def consistent-interp-tautology atms-of-s-def*)
moreover have *tot2*: ‹*total-over-m* (*set ?M*) (*set-mset* (*init-clss S*))›
using *H1* **by** (*auto simp*: *total-over-m-def total-over-set-def lit-in-set-iff-atm*)
ultimately show ‹(*pNeg* ∘ *mset* ∘ (@) *M*) *xc* ∈# *conflicting-clss S*›
using *ge*[*of* ‹*xc*›] *dist 0 cons card-mono*[*OF - atm*] *tot2 cons*
by (*auto simp*: *conflicting-clss-def too-heavy-clauses-def simple-clss-finite*
intro!: *too-heavy-clauses-conflicting-clauses imageI*)
qed
next
case (*Suc n*) **note** *IH* = *this*(*1*)
let *?H* = ‹*?A n*›
show *?case*
proof *clarify*
fix *x* :: ‹′*v literal multiset*› **and** *xa* :: ‹′*v literal multiset*› **and**
xb :: ‹′*v literal list*› **and** *xc* :: ‹′*v literal list*›
assume
dist: ‹*distinct-mset* (*atm-of* ‘# *mset* (*M* @ *xc*))› **and**
cons: ‹*consistent-interp* (*set-mset* (*mset* (*M* @ *xc*)))› **and**
atm′: ‹*atms-of-s* (*set* (*M* @ *xc*)) ⊆ *atms-of-mm* (*init-clss S*)› **and**
0: ‹*card* (*atms-of-mm* (*init-clss S*)) = *Suc n* + *card* (*atms-of* (*mset* (*M* @ *xc*)))›
then obtain *a* **where**
a: ‹*a* ∈ *atms-of-mm* (*init-clss S*)› **and**
a-notin: ‹*a* ∉ *atms-of-s* (*set* (*M* @ *xc*))›
by (*metis Suc-n-not-le-n add-Suc-shift atms-of-mmltiset atms-of-s-def le-add2*
subsetI subset-antisym)
have *dist2*: ‹*distinct xc*› ‹*distinct-mset* (*atm-of* ‘# *mset xc*)›
‹*distinct-mset* (*mset M* + *mset xc*)›
using *dist distinct-mset-atm-ofD*[*OF dist*]
unfolding *mset-append*[*symmetric*] *distinct-mset-mset-distinct*
by (*auto dest*: *distinct-mset-union2 distinct-mset-atm-ofD*)
let *?xc1* = ‹*Pos a* # *xc*›
let *?xc2* = ‹*Neg a* # *xc*›
have ‹*?xc1* ∈ *?H*›
using *dist cons atm*′ *0 dist2 a-notin a*
by (*auto simp*: *distinct-mset-add mset-inter-empty-set-mset*
lit-in-set-iff-atm card-insert-if)
from *set-mp*[*OF IH imageI*[*OF this*]]
have *1*: ‹*too-heavy-clauses* (*init-clss S*) (*weight S*) ⊨*pm add-mset* (−(*Pos a*)) (*pNeg* (*mset* (*M* @
xc)))›
unfolding *conflicting-clss-def* **unfolding** *conflicting-clauses-def*
by (*auto simp*: *pNeg-simps*)
have ‹*?xc2* ∈ *?H*›
using *dist cons atm*′ *0 dist2 a-notin a*
by (*auto simp*: *distinct-mset-add mset-inter-empty-set-mset*
lit-in-set-iff-atm card-insert-if)
from *set-mp*[*OF IH imageI*[*OF this*]]
have *2*: ‹*too-heavy-clauses* (*init-clss S*) (*weight S*) ⊨*pm add-mset* (*Pos a*) (*pNeg* (*mset* (*M* @ *xc*)))›
unfolding *conflicting-clss-def* **unfolding** *conflicting-clauses-def*
by (*auto simp*: *pNeg-simps*)

have ‹¬*tautology* (*mset* (*M* @ *xc*))›

using *cons* **unfolding** *consistent-interp-tautology*[*symmetric*]
by *auto*
then have ‹¬*tautology* (*pNeg* (*mset M*) + *pNeg* (*mset xc*))›
unfolding *mset-append*[*symmetric*] *pNeg-simps*[*symmetric*]
by (*auto simp del*: *mset-append*)
then have ‹*pNeg* (*mset M*) + *pNeg* (*mset xc*) ∈ *simple-clss* (*atms-of-mm* (*init-clss S*))›
using *atm′ dist2* **by** (*auto simp*: *simple-clss-def atms-of-s-def simp flip*: *pNeg-simps*)
then show ‹(*pNeg* ∘ *mset* ∘ (@) *M*) *xc* ∈# *conflicting-clss S*›
using *true-clss-cls-or-true-clss-cls-or-not-true-clss-cls-or*[*OF 1 2*] **apply** −
unfolding *conflicting-clss-def conflicting-clauses-def*
by (*subst* (*asm*) *true-clss-cls-remdups-mset*[*symmetric*])
(*auto simp*: *simple-clss-finite pNeg-simps intro*: *true-clss-cls-cong-set-mset*
simp del: *true-clss-cls-remdups-mset*)
qed
qed
have ‹[] ∈ {*M′*.
distinct-mset (*atm-of* ‘# *mset* (*M* @ *M′*)) ∧
consistent-interp (*set-mset* (*mset* (*M* @ *M′*))) ∧
atms-of-s (*set* (*M* @ *M′*)) ⊆ *atms-of-mm* (*init-clss S*) ∧
card (*atms-of-mm* (*init-clss S*)) =
card (*atms-of-mm* (*init-clss S*)) − *card* (*atms-of* (*mset M*)) +
card (*atms-of* (*mset* (*M* @ *M′*)))}›
using *card-mono*[*OF - assms*(*2*)] *assms* **by** (*auto dest*: *card-mono distinct-consistent-distinct-atm*)

from *set-mp*[*OF 0 imageI*[*OF this*]]
show ‹*pNeg* (*mset M*) ∈# *conflicting-clss S*›
by *auto*
qed

end

end
theory *OCDCL*
imports *CDCL-W-Optimal-Model*
begin

Alternative versions

We instantiate our more general rules with exactly the rule from Christoph's OCDCL with either versions of improve.

Weights

This one is the version of the weight functions used by Christoph Weidenbach. However, we have decided to not instantiate tho calculus with this weight function, because it only a slight restriction.

locale *ocdcl-weight-WB* =
fixes
ν :: ‹*′v literal* ⇒ *nat*›
begin

definition *ϱ* :: ‹*′v clause* ⇒ *nat*› **where**
‹*ϱ M* = (∑ *A* ∈# *M*. *ν A*)›

sublocale *ocdcl-weight* ϱ
  by (*unfold-locales*)
    (*auto simp*: ϱ*-def sum-image-mset-mono*)

end

Calculus with simple Improve rule

context *conflict-driven-clause-learning*$_W$*-optimal-weight*
begin

To make sure that the paper version of the correct, we restrict the previous calculus to exactly the rules that are on paper.

inductive *pruning* :: ‹'*st* ⇒ '*st* ⇒ *bool*› **where**
pruning-rule:
  ‹*pruning S T*›
  if
    ‹⋀*M'*. *total-over-m* (*set-mset* (*mset* (*map lit-of* (*trail S*) @ *M'*))) (*set-mset* (*init-clss S*)) ⟹
      distinct-mset (*atm-of* '# *mset* (*map lit-of* (*trail S*) @ *M'*)) ⟹
      consistent-interp (*set-mset* (*mset* (*map lit-of* (*trail S*) @ *M'*))) ⟹
      ϱ' (*weight S*) ≤ *Found* (ϱ (*mset* (*map lit-of* (*trail S*) @ *M'*)))›
    ‹*conflicting S* = *None*›
    ‹*T* ∼ *update-conflicting* (*Some* (*negate-ann-lits* (*trail S*))) *S*›

inductive *oconflict-opt* :: ‹'*st* ⇒ '*st* ⇒ *bool*› **for** *S T* :: '*st* **where**
oconflict-opt-rule:
  ‹*oconflict-opt S T*›
  if
    ‹*Found* (ϱ (*lit-of* '# *mset* (*trail S*))) ≥ ϱ' (*weight S*)›
    ‹*conflicting S* = *None*›
    ‹*T* ∼ *update-conflicting* (*Some* (*negate-ann-lits* (*trail S*))) *S*›

inductive *improve* :: ‹'*st* ⇒ '*st* ⇒ *bool*› **for** *S T* :: '*st* **where**
improve-rule:
  ‹*improve S T*›
  if
    ‹*total-over-m* (*lits-of-l* (*trail S*)) (*set-mset* (*init-clss S*))›
    ‹*Found* (ϱ (*lit-of* '# *mset* (*trail S*))) < ϱ' (*weight S*)›
    ‹*trail S* ⊨*asm init-clss S*›
    ‹*conflicting S* = *None*›
    ‹*T* ∼ *update-weight-information* (*trail S*) *S*›

This is the basic version of the calculus:

inductive *ocdcl*$_w$ :: ‹'*st* ⇒ '*st* ⇒ *bool*› **for** *S* :: '*st* **where**
ocdcl-conflict: ‹*conflict S S'* ⟹ *ocdcl*$_w$ *S S'*› |
ocdcl-propagate: ‹*propagate S S'* ⟹ *ocdcl*$_w$ *S S'*› |
ocdcl-improve: ‹*improve S S'* ⟹ *ocdcl*$_w$ *S S'*› |
ocdcl-conflict-opt: ‹*oconflict-opt S S'* ⟹ *ocdcl*$_w$ *S S'*› |
ocdcl-other': ‹*ocdcl*$_W$*-o S S'* ⟹ *ocdcl*$_w$ *S S'*› |
ocdcl-pruning: ‹*pruning S S'* ⟹ *ocdcl*$_w$ *S S'*›

inductive *ocdcl*$_w$*-stgy* :: ‹'*st* ⇒ '*st* ⇒ *bool*› **for** *S* :: '*st* **where**
ocdcl$_w$*-conflict*: ‹*conflict S S'* ⟹ *ocdcl*$_w$*-stgy S S'*› |
ocdcl$_w$*-propagate*: ‹*propagate S S'* ⟹ *ocdcl*$_w$*-stgy S S'*› |
ocdcl$_w$*-improve*: ‹*improve S S'* ⟹ *ocdcl*$_w$*-stgy S S'*› |

```
ocdcl_w-conflict-opt: ‹conflict-opt S S' ⟹ ocdcl_w-stgy S S'› |
ocdcl_w-other': ‹ocdcl_W-o S S' ⟹ no-confl-prop-impr S ⟹ ocdcl_w-stgy S S'›

lemma pruning-conflict-opt:
  assumes ocdcl-pruning: ‹pruning S T› and
    inv: ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state S)›
  shows ‹conflict-opt S T›
proof -
  have le:
    ‹⋀M'. total-over-m (set-mset (mset (map lit-of (trail S) @ M')))
        (set-mset (init-clss S)) ⟹
       distinct-mset (atm-of `# mset (map lit-of (trail S) @ M')) ⟹
       consistent-interp (set-mset (mset (map lit-of (trail S) @ M'))) ⟹
       ϱ' (weight S) ≤ Found (ϱ (mset (map lit-of (trail S) @ M')))›
    using ocdcl-pruning by (auto simp: pruning.simps)
  have alien: ‹cdcl_W-restart-mset.no-strange-atm (abs-state S)› and
    lev: ‹cdcl_W-restart-mset.cdcl_W-M-level-inv (abs-state S)›
    using inv unfolding cdcl_W-restart-mset.cdcl_W-all-struct-inv-def
    by fast+
  have incl: ‹atms-of (mset (map lit-of (trail S))) ⊆ atms-of-mm (init-clss S)›
    using alien unfolding cdcl_W-restart-mset.no-strange-atm-def
    by (auto simp: abs-state-def cdcl_W-restart-mset-state lits-of-def image-image atms-of-def)
  have dist: ‹distinct (map lit-of (trail S))› and
    cons: ‹consistent-interp (set (map lit-of (trail S)))›
    using lev unfolding cdcl_W-restart-mset.cdcl_W-M-level-inv-def
    by (auto simp: abs-state-def cdcl_W-restart-mset-state lits-of-def image-image atms-of-def
      dest: no-dup-map-lit-of)
  have ‹negate-ann-lits (trail S) ∈# conflicting-clss S›
    unfolding negate-ann-lits-pNeg-lit-of comp-def mset-map[symmetric]
    by (rule pruned-clause-in-conflicting-clss[OF le incl dist cons]) fast+
  then show ‹conflict-opt S T›
    by (rule conflict-opt.intros) (use ocdcl-pruning in ‹auto simp: pruning.simps›)
qed

lemma ocdcl-conflict-opt-conflict-opt:
  assumes ocdcl-pruning: ‹oconflict-opt S T› and
    inv: ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (abs-state S)›
  shows ‹conflict-opt S T›
proof -
  have alien: ‹cdcl_W-restart-mset.no-strange-atm (abs-state S)› and
    lev: ‹cdcl_W-restart-mset.cdcl_W-M-level-inv (abs-state S)›
    using inv unfolding cdcl_W-restart-mset.cdcl_W-all-struct-inv-def
    by fast+
  have incl: ‹atms-of (lit-of `# mset (trail S)) ⊆ atms-of-mm (init-clss S)›
    using alien unfolding cdcl_W-restart-mset.no-strange-atm-def
    by (auto simp: abs-state-def cdcl_W-restart-mset-state lits-of-def image-image atms-of-def)
  have dist: ‹distinct-mset (lit-of `# mset (trail S))› and
    cons: ‹consistent-interp (set (map lit-of (trail S)))› and
    tauto: ‹¬tautology (lit-of `# mset (trail S))›
    using lev unfolding cdcl_W-restart-mset.cdcl_W-M-level-inv-def
    by (auto simp: abs-state-def cdcl_W-restart-mset-state lits-of-def image-image atms-of-def
      dest: no-dup-map-lit-of no-dup-distinct no-dup-not-tautology)
  have ‹lit-of `# mset (trail S) ∈ simple-clss (atms-of-mm (init-clss S))›
    using dist incl tauto by (auto simp: simple-clss-def)
  then have simple: ‹(lit-of `# mset (trail S))
    ∈ {a. a ∈# mset-set (simple-clss (atms-of-mm (init-clss S))) ∧
```

ϱ' (*weight S*) ≤ *Found* (ϱ *a*)}›
using *ocdcl-pruning* **by** (*auto simp*: *simple-clss-finite oconflict-opt.simps*)
have ‹*negate-ann-lits* (*trail S*) ∈# *conflicting-clss S*›
unfolding *negate-ann-lits-pNeg-lit-of comp-def conflicting-clss-def*
by (*rule too-heavy-clauses-conflicting-clauses*)
(*use simple* **in** ‹*auto simp*: *too-heavy-clauses-def oconflict-opt.simps*›)
then show ‹*conflict-opt S T*›
apply (*rule conflict-opt.intros*)
subgoal using *ocdcl-pruning* **by** (*auto simp*: *oconflict-opt.simps*)
subgoal using *ocdcl-pruning* **by** (*auto simp*: *oconflict-opt.simps*)
done
qed

lemma *improve-improvep*:
assumes *imp*: ‹*improve S T*› **and**
inv: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state S*)›
shows ‹*improvep S T*›
proof −
have *alien*: ‹$cdcl_W$*-restart-mset.no-strange-atm* (*abs-state S*)› **and**
lev: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv* (*abs-state S*)›
using *inv* **unfolding** $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*
by *fast+*
have *incl*: ‹*atms-of* (*lit-of* ‘# *mset* (*trail S*)) ⊆ *atms-of-mm* (*init-clss S*)›
using *alien* **unfolding** $cdcl_W$*-restart-mset.no-strange-atm-def*
by (*auto simp*: *abs-state-def* $cdcl_W$*-restart-mset-state lits-of-def image-image atms-of-def*)
have *dist*: ‹*distinct-mset* (*lit-of* ‘# *mset* (*trail S*))› **and**
cons: ‹*consistent-interp* (*set* (*map lit-of* (*trail S*)))› **and**
tauto: ‹¬*tautology* (*lit-of* ‘# *mset* (*trail S*))› **and**
nd: ‹*no-dup* (*trail S*)›
using *lev* **unfolding** $cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv-def*
by (*auto simp*: *abs-state-def* $cdcl_W$*-restart-mset-state lits-of-def image-image atms-of-def*
dest: *no-dup-map-lit-of no-dup-distinct no-dup-not-tautology*)
have ‹*lit-of* ‘# *mset* (*trail S*) ∈ *simple-clss* (*atms-of-mm* (*init-clss S*))›
using *dist incl tauto* **by** (*auto simp*: *simple-clss-def*)
have *tot'*: ‹*total-over-m* (*lits-of-l* (*trail S*)) (*set-mset* (*init-clss S*))› **and**
confl: ‹*conflicting S* = *None*› **and**
T: ‹*T* ∼ *update-weight-information* (*trail S*) *S*›
using *imp nd* **by** (*auto simp*: *is-improving-int-def improve.simps*)
have *M'*: ‹ϱ (*lit-of* ‘# *mset M'*) = ϱ (*lit-of* ‘# *mset* (*trail S*))›
if ‹*total-over-m* (*lits-of-l M'*) (*set-mset* (*init-clss S*))› **and**
incl: ‹*mset* (*trail S*) ⊆# *mset M'*› **and**
‹*lit-of* ‘# *mset M'* ∈ *simple-clss* (*atms-of-mm* (*init-clss S*))›
for *M'*
proof −
have [*simp*]: ‹*lits-of-l M'* = *set-mset* (*lit-of* ‘# *mset M'*)›
by (*auto simp*: *lits-of-def*)
obtain *A* **where** *A*: ‹*mset M'* = *A* + *mset* (*trail S*)›
using *incl* **by** (*auto simp*: *mset-subset-eq-exists-conv*)
have *M'*: ‹*lits-of-l M'* = *lit-of* ‘ *set-mset A* ∪ *lits-of-l* (*trail S*)›
unfolding *lits-of-def*
by (*metis A image-Un set-mset-mset set-mset-union*)
have ‹*mset M'* = *mset* (*trail S*)›
using *that tot'* **unfolding** *A total-over-m-alt-def*
apply (*case-tac A*)
apply (*auto simp*: *A simple-clss-def distinct-mset-add M' image-Un*

tautology-union mset-inter-empty-set-mset atms-of-def atms-of-s-def
atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set image-image
tautology-add-mset)
by (*metis* (*no-types*, *lifting*) *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set*
lits-of-def subsetCE)
then show *?thesis*
by *auto*
qed

have ‹*lit-of* ‘# *mset* (*trail S*) ∈ *simple-clss* (*atms-of-mm* (*init-clss S*))›
using *tauto dist incl* **by** (*auto simp*: *simple-clss-def*)
then have *improving*: ‹*is-improving* (*trail S*) (*trail S*) *S*› **and**
‹*total-over-m* (*lits-of-l* (*trail S*)) (*set-mset* (*init-clss S*))›
using *imp nd* **by** (*auto simp*: *is-improving-int-def improve.simps intro*: *M′*)

show ‹*improvep S T*›
by (*rule improvep.intros*[*OF improving confl T*])
qed

lemma $ocdcl_w$*-cdcl-bnb*:
assumes ‹$ocdcl_w$ *S T*› **and**
inv: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state S*)›
shows ‹*cdcl-bnb S T*›
using *assms* **by** (*cases*) (*auto intro*: *cdcl-bnb.intros dest*: *pruning-conflict-opt*
ocdcl-conflict-opt-conflict-opt improve-improvep)

lemma $ocdcl_w$*-stgy-cdcl-bnb-stgy*:
assumes ‹$ocdcl_w$*-stgy S T*› **and**
inv: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state S*)›
shows ‹*cdcl-bnb-stgy S T*›
using *assms* **by** (*cases*)
(*auto intro*: *cdcl-bnb-stgy.intros dest*: *pruning-conflict-opt improve-improvep*)

lemma *rtranclp-*$ocdcl_w$*-stgy-rtranclp-cdcl-bnb-stgy*:
assumes ‹$ocdcl_w$*-stgy*** *S T*› **and**
inv: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state S*)›
shows ‹*cdcl-bnb-stgy*** *S T*›
using *assms*
by (*induction rule*: *rtranclp-induct*)
(*auto dest*: *rtranclp-cdcl-bnb-stgy-all-struct-inv*[*OF rtranclp-cdcl-bnb-stgy-cdcl-bnb*]
$ocdcl_w$*-stgy-cdcl-bnb-stgy*)

lemma *no-step-*$ocdcl_w$*-no-step-cdcl-bnb*:
assumes ‹*no-step* $ocdcl_w$ *S*› **and**
inv: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state S*)›
shows ‹*no-step cdcl-bnb S*›
proof −
have
nsc: ‹*no-step conflict S*› **and**
nsp: ‹*no-step propagate S*› **and**
nsi: ‹*no-step improve S*› **and**
nsco: ‹*no-step oconflict-opt S*› **and**
nso: ‹*no-step* $ocdcl_W$*-o S*›**and**
nspr: ‹*no-step pruning S*›
using *assms*(*1*) **by** (*auto simp*: *cdcl-bnb.simps* $ocdcl_w$*.simps*)

```
have alien: ‹cdcl_W-restart-mset.no-strange-atm (abs-state S)› and
  lev: ‹cdcl_W-restart-mset.cdcl_W-M-level-inv (abs-state S)›
  using inv unfolding cdcl_W-restart-mset.cdcl_W-all-struct-inv-def
  by fast+
have incl: ‹atms-of (lit-of ‘# mset (trail S)) ⊆ atms-of-mm (init-clss S)›
  using alien unfolding cdcl_W-restart-mset.no-strange-atm-def
  by (auto simp: abs-state-def cdcl_W-restart-mset-state lits-of-def image-image atms-of-def)
have dist: ‹distinct-mset (lit-of ‘# mset (trail S))› and
  cons: ‹consistent-interp (set (map lit-of (trail S)))› and
  tauto: ‹¬tautology (lit-of ‘# mset (trail S))› and
  n-d: ‹no-dup (trail S)›
  using lev unfolding cdcl_W-restart-mset.cdcl_W-M-level-inv-def
  by (auto simp: abs-state-def cdcl_W-restart-mset-state lits-of-def image-image atms-of-def
    dest: no-dup-map-lit-of no-dup-distinct no-dup-not-tautology)

have nsip: False if imp: ‹improvep S S'› for S'
proof −
  obtain M' where
    [simp]: ‹conflicting S = None› and
    is-improving:
      ‹⋀M'. total-over-m (lits-of-l M') (set-mset (init-clss S)) ⟶
          mset (trail S) ⊆# mset M' ⟶
          lit-of ‘# mset M' ∈ simple-clss (atms-of-mm (init-clss S)) ⟶
          ϱ (lit-of ‘# mset M') = ϱ (lit-of ‘# mset (trail S))› and
    S': ‹S' ∼ update-weight-information M' S›
    using imp by (auto simp: improvep.simps is-improving-int-def)
  have 1: ‹¬ ϱ' (weight S) ≤ Found (ϱ (lit-of ‘# mset (trail S)))›
    using nsco
    by (auto simp: is-improving-int-def oconflict-opt.simps)
  have 2: ‹total-over-m (lits-of-l (trail S)) (set-mset (init-clss S))›
  proof (rule ccontr)
    assume ‹¬ ?thesis›
    then obtain A where
      ‹A ∈ atms-of-mm (init-clss S)› and
      ‹A ∉ atms-of-s (lits-of-l (trail S))›
      by (auto simp: total-over-m-def total-over-set-def)
    then show ‹False›
      using decide-rule[of S ‹Pos A›, OF - - - state-eq-ref] nso
      by (auto simp: Decided-Propagated-in-iff-in-lits-of-l ocdcl_W-o.simps)
  qed
  have 3: ‹trail S ⊨asm init-clss S›
    unfolding true-annots-def
  proof clarify
    fix C
    assume C: ‹C ∈# init-clss S›
    have ‹total-over-m (lits-of-l (trail S)) {C}›
      using 2 C by (auto dest!: multi-member-split)
    moreover have ‹¬ trail S ⊨as CNot C›
      using C nsc conflict-rule[of S C, OF - - - state-eq-ref]
      by (auto simp: clauses-def dest!: multi-member-split)
    ultimately show ‹trail S ⊨a C›
      using total-not-CNot[of ‹lits-of-l (trail S)› C] unfolding true-annots-true-cls true-annot-def
      by auto
  qed
  have 4: ‹lit-of ‘# mset (trail S) ∈ simple-clss (atms-of-mm (init-clss S))›
    using tauto cons incl dist by (auto simp: simple-clss-def)
```

have ‹*improve S* (*update-weight-information* (*trail S*) *S*)›
by (*rule improve.intros*[*OF 2 - 3*]) (*use 1 2* **in** *auto*)
then show *False*
using *nsi* **by** *auto*
qed
moreover have *False* **if** ‹*conflict-opt S S'*› **for** *S'*
proof −
have [*simp*]: ‹*conflicting S* = *None*›
using *that* **by** (*auto simp*: *conflict-opt.simps*)
have *1*: ‹¬ *ϱ'* (*weight S*) ≤ *Found* (*ϱ* (*lit-of* '# *mset* (*trail S*)))›
using *nsco*
by (*auto simp*: *is-improving-int-def oconflict-opt.simps*)
have *2*: ‹*total-over-m* (*lits-of-l* (*trail S*)) (*set-mset* (*init-clss S*))›
proof (*rule ccontr*)
assume ‹¬ *?thesis*›
then obtain *A* **where**
‹*A* ∈ *atms-of-mm* (*init-clss S*)› **and**
‹*A* ∉ *atms-of-s* (*lits-of-l* (*trail S*))›
by (*auto simp*: *total-over-m-def total-over-set-def*)
then show ‹*False*›
using *decide-rule*[*of S* ‹*Pos A*›, *OF - - - state-eq-ref*] *nso*
by (*auto simp*: *Decided-Propagated-in-iff-in-lits-of-l* $ocdcl_W$*-o.simps*)
qed
have *3*: ‹*trail S* ⊨*asm init-clss S*›
unfolding *true-annots-def*
proof *clarify*
fix *C*
assume *C*: ‹*C* ∈# *init-clss S*›
have ‹*total-over-m* (*lits-of-l* (*trail S*)) {*C*}›
using *2 C* **by** (*auto dest*!: *multi-member-split*)
moreover have ‹¬ *trail S* ⊨*as CNot C*›
using *C nsc conflict-rule*[*of S C*, *OF - - - state-eq-ref*]
by (*auto simp*: *clauses-def dest*!: *multi-member-split*)
ultimately show ‹*trail S* ⊨*a C*›
using *total-not-CNot*[*of* ‹*lits-of-l* (*trail S*)› *C*] **unfolding** *true-annots-true-cls true-annot-def*
by *auto*
qed
have *4*: ‹*lit-of* '# *mset* (*trail S*) ∈ *simple-clss* (*atms-of-mm* (*init-clss S*))›
using *tauto cons incl dist* **by** (*auto simp*: *simple-clss-def*)

have [*intro*]: ‹*ϱ* (*lit-of* '# *mset M'*) = *ϱ* (*lit-of* '# *mset* (*trail S*))›
if
‹*lit-of* '# *mset* (*trail S*) ∈ *simple-clss* (*atms-of-mm* (*init-clss S*))› **and**
‹*atms-of* (*lit-of* '# *mset* (*trail S*)) ⊆ *atms-of-mm* (*init-clss S*)› **and**
‹*no-dup* (*trail S*)› **and**
‹*total-over-m* (*lits-of-l M'*) (*set-mset* (*init-clss S*))› **and**
incl: ‹*mset* (*trail S*) ⊆# *mset M'*› **and**
‹*lit-of* '# *mset M'* ∈ *simple-clss* (*atms-of-mm* (*init-clss S*))›
for *M'* :: ‹('*v literal*, '*v literal*, '*v literal multiset*) *annotated-lit list*›
proof −
have [*simp*]: ‹*lits-of-l M'* = *set-mset* (*lit-of* '# *mset M'*)›
by (*auto simp*: *lits-of-def*)
obtain *A* **where** *A*: ‹*mset M'* = *A* + *mset* (*trail S*)›
using *incl* **by** (*auto simp*: *mset-subset-eq-exists-conv*)
have *M'*: ‹*lits-of-l M'* = *lit-of* ' *set-mset A* ∪ *lits-of-l* (*trail S*)›
unfolding *lits-of-def*

by (*metis A image-Un set-mset-mset set-mset-union*)
have ‹*mset M′* = *mset* (*trail S*)›
using *that 2* **unfolding** *A total-over-m-alt-def*
apply (*case-tac A*)
apply (*auto simp*: *A simple-clss-def distinct-mset-add M′ image-Un tautology-union mset-inter-empty-set-mset atms-of-def atms-of-s-def atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set image-image tautology-add-mset*)
by (*metis* (*no-types*, *lifting*) *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set lits-of-def subsetCE*)
then show *?thesis*
using *2* **by** *auto*
qed
have *imp*: ‹*is-improving* (*trail S*) (*trail S*) *S*›
using *1 2 3 4 incl n-d* **unfolding** *is-improving-int-def*
by (*auto simp*: *oconflict-opt.simps*)

show ‹*False*›
using *trail-is-improving-Ex-improve*[*of S*, *OF - imp*] *nsip*
by *auto*
qed
ultimately show *?thesis*
using *nsc nsp nsi nsco nso nsp nspr*
by (*auto simp*: *cdcl-bnb.simps*)
qed

lemma *all-struct-init-state-distinct-iff*:
‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state* (*init-state N*)) ⟷ *distinct-mset-mset N*›
unfolding *init-state.simps*[*symmetric*]
by (*auto simp*: $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*
$cdcl_W$*-restart-mset.distinct-*$cdcl_W$*-state-def*
$cdcl_W$*-restart-mset.no-strange-atm-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-conflicting-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause-alt-def*
abs-state-def $cdcl_W$*-restart-mset-state*)

lemma *no-step-ocdcl*$_w$*-stgy-no-step-cdcl-bnb-stgy*:
assumes ‹*no-step ocdcl*$_w$*-stgy S*› **and**
inv: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state S*)›
shows ‹*no-step cdcl-bnb-stgy S*›
using *assms no-step-ocdcl*$_w$*-no-step-cdcl-bnb*[*of S*]
by (*auto simp*: *ocdcl*$_w$*-stgy.simps ocdcl*$_w$*.simps cdcl-bnb.simps cdcl-bnb-stgy.simps*
dest: *ocdcl-conflict-opt-conflict-opt pruning-conflict-opt*)

lemma *full-ocdcl*$_w$*-stgy-full-cdcl-bnb-stgy*:
assumes ‹*full ocdcl*$_w$*-stgy S T*› **and**
inv: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state S*)›
shows ‹*full cdcl-bnb-stgy S T*›
using *assms rtranclp-ocdcl*$_w$*-stgy-rtranclp-cdcl-bnb-stgy*[*of S T*]
no-step-ocdcl$_w$*-stgy-no-step-cdcl-bnb-stgy*[*of T*]
unfolding *full-def*
by (*auto dest*: *rtranclp-cdcl-bnb-stgy-all-struct-inv*[*OF rtranclp-cdcl-bnb-stgy-cdcl-bnb*])

corollary *full-ocdcl*$_w$*-stgy-no-conflicting-clause-from-init-state*:

assumes
st: ‹*full* $ocdcl_w$*-stgy* (*init-state N*) *T*› **and**
dist: ‹*distinct-mset-mset N*›
shows
‹*weight T* = *None* ⟹ *unsatisfiable* (*set-mset N*)› **and**
‹*weight T* ≠ *None* ⟹ *model-on* (*set-mset* (*the* (*weight T*))) *N* ∧ *set-mset* (*the* (*weight T*)) ⊨*sm N* ∧
distinct-mset (*the* (*weight T*))› **and**
‹*distinct-mset I* ⟹ *consistent-interp* (*set-mset I*) ⟹ *atms-of I* = *atms-of-mm N* ⟹
set-mset I ⊨*sm N* ⟹ *Found* (ϱ *I*) ≥ ϱ′ (*weight T*)›
using *full-cdcl-bnb-stgy-no-conflicting-clause-from-init-state*[*of N T*,
OF full-$ocdcl_w$*-stgy-full-cdcl-bnb-stgy*[*OF st*] *dist*] *dist*
by (*auto simp*: *all-struct-init-state-distinct-iff model-on-def*
dest: *multi-member-split*)

lemma *wf-*$ocdcl_w$:
‹*wf* {(*T*, *S*). $cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv* (*abs-state S*)
∧ $ocdcl_w$ *S T*}›
by (*rule wf-subset*[*OF wf-cdcl-bnb2*]) (*auto dest*: $ocdcl_w$*-cdcl-bnb*)

Calculus with generalised Improve rule

Now a version with the more general improve rule:

inductive $ocdcl_w$*-p* :: ‹′*st* ⇒ ′*st* ⇒ *bool*› **for** *S* :: ′*st* **where**
ocdcl-conflict: ‹*conflict S S*′ ⟹ $ocdcl_w$*-p S S*′› |
ocdcl-propagate: ‹*propagate S S*′ ⟹ $ocdcl_w$*-p S S*′› |
ocdcl-improve: ‹*improvep S S*′ ⟹ $ocdcl_w$*-p S S*′› |
ocdcl-conflict-opt: ‹*oconflict-opt S S*′ ⟹ $ocdcl_w$*-p S S*′› |
ocdcl-other′: ‹$ocdcl_W$*-o S S*′ ⟹ $ocdcl_w$*-p S S*′› |
ocdcl-pruning: ‹*pruning S S*′ ⟹ $ocdcl_w$*-p S S*′›

inductive $ocdcl_w$*-p-stgy* :: ‹′*st* ⇒ ′*st* ⇒ *bool*› **for** *S* :: ′*st* **where**
$ocdcl_w$*-p-conflict*: ‹*conflict S S*′ ⟹ $ocdcl_w$*-p-stgy S S*′› |
$ocdcl_w$*-p-propagate*: ‹*propagate S S*′ ⟹ $ocdcl_w$*-p-stgy S S*′› |
$ocdcl_w$*-p-improve*: ‹*improvep S S*′ ⟹ $ocdcl_w$*-p-stgy S S*′› |
$ocdcl_w$*-p-conflict-opt*: ‹*conflict-opt S S*′ ⟹ $ocdcl_w$*-p-stgy S S*′›|
$ocdcl_w$*-p-pruning*: ‹*pruning S S*′ ⟹ $ocdcl_w$*-p-stgy S S*′› |
$ocdcl_w$*-p-other*′: ‹$ocdcl_W$*-o S S*′ ⟹ *no-confl-prop-impr S* ⟹ $ocdcl_w$*-p-stgy S S*′›

lemma $ocdcl_w$*-p-cdcl-bnb*:
assumes ‹$ocdcl_w$*-p S T*› **and**
inv: ‹$cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv* (*abs-state S*)›
shows ‹*cdcl-bnb S T*›
using *assms* **by** (*cases*) (*auto intro*: *cdcl-bnb.intros dest*: *pruning-conflict-opt*
ocdcl-conflict-opt-conflict-opt)

lemma $ocdcl_w$*-p-stgy-cdcl-bnb-stgy*:
assumes ‹$ocdcl_w$*-p-stgy S T*› **and**
inv: ‹$cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv* (*abs-state S*)›
shows ‹*cdcl-bnb-stgy S T*›
using *assms* **by** (*cases*) (*auto intro*: *cdcl-bnb-stgy.intros dest*: *pruning-conflict-opt*)

lemma *rtranclp-*$ocdcl_w$*-p-stgy-rtranclp-cdcl-bnb-stgy*:

assumes ‹$ocdcl_w$-*p-stgy*** S T› **and**
inv: ‹$cdcl_W$-*restart-mset*.$cdcl_W$-*all-struct-inv* (*abs-state* S)›
shows ‹*cdcl-bnb-stgy*** S T›
using *assms*
by (*induction rule*: *rtranclp-induct*)
(*auto dest*: *rtranclp-cdcl-bnb-stgy-all-struct-inv*[*OF rtranclp-cdcl-bnb-stgy-cdcl-bnb*]
$ocdcl_w$-*p-stgy-cdcl-bnb-stgy*)

lemma *no-step-*$ocdcl_w$*-p-no-step-cdcl-bnb*:
assumes ‹*no-step* $ocdcl_w$-*p* S› **and**
inv: ‹$cdcl_W$-*restart-mset*.$cdcl_W$-*all-struct-inv* (*abs-state* S)›
shows ‹*no-step cdcl-bnb* S›
proof –
have
nsc: ‹*no-step conflict* S› **and**
nsp: ‹*no-step propagate* S› **and**
nsi: ‹*no-step improvep* S› **and**
nsco: ‹*no-step oconflict-opt* S› **and**
nso: ‹*no-step* $ocdcl_W$-*o* S›**and**
nspr: ‹*no-step pruning* S›
using *assms*(*1*) **by** (*auto simp*: *cdcl-bnb.simps* $ocdcl_w$-*p.simps*)
have *alien*: ‹$cdcl_W$-*restart-mset.no-strange-atm* (*abs-state* S)› **and**
lev: ‹$cdcl_W$-*restart-mset*.$cdcl_W$-*M-level-inv* (*abs-state* S)›
using *inv* **unfolding** $cdcl_W$-*restart-mset*.$cdcl_W$-*all-struct-inv-def*
by *fast+*
have *incl*: ‹*atms-of* (*lit-of* ‘# *mset* (*trail* S)) ⊆ *atms-of-mm* (*init-clss* S)›
using *alien* **unfolding** $cdcl_W$-*restart-mset.no-strange-atm-def*
by (*auto simp*: *abs-state-def* $cdcl_W$-*restart-mset-state lits-of-def image-image atms-of-def*)
have *dist*: ‹*distinct-mset* (*lit-of* ‘# *mset* (*trail* S))› **and**
cons: ‹*consistent-interp* (*set* (*map lit-of* (*trail* S)))› **and**
tauto: ‹¬*tautology* (*lit-of* ‘# *mset* (*trail* S))› **and**
n-d: ‹*no-dup* (*trail* S)›
using *lev* **unfolding** $cdcl_W$-*restart-mset*.$cdcl_W$-*M-level-inv-def*
by (*auto simp*: *abs-state-def* $cdcl_W$-*restart-mset-state lits-of-def image-image atms-of-def*
dest: *no-dup-map-lit-of no-dup-distinct no-dup-not-tautology*)

have *False* **if** ‹*conflict-opt* S S'› **for** S'
proof –
have [*simp*]: ‹*conflicting* S = *None*›
using *that* **by** (*auto simp*: *conflict-opt.simps*)
have *1*: ‹¬ ϱ' (*weight* S) ≤ *Found* (ϱ (*lit-of* ‘# *mset* (*trail* S)))›
using *nsco*
by (*auto simp*: *is-improving-int-def oconflict-opt.simps*)
have *2*: ‹*total-over-m* (*lits-of-l* (*trail* S)) (*set-mset* (*init-clss* S))›
proof (*rule ccontr*)
assume ‹¬ *?thesis*›
then obtain A **where**
‹A ∈ *atms-of-mm* (*init-clss* S)› **and**
‹A ∉ *atms-of-s* (*lits-of-l* (*trail* S))›
by (*auto simp*: *total-over-m-def total-over-set-def*)
then show ‹*False*›
using *decide-rule*[*of* S ‹*Pos* A›, *OF* - - - *state-eq-ref*] *nso*
by (*auto simp*: *Decided-Propagated-in-iff-in-lits-of-l* $ocdcl_W$-*o.simps*)
qed
have *3*: ‹*trail* S ⊨*asm init-clss* S›
unfolding *true-annots-def*

```
    proof clarify
      fix C
      assume C: ‹C ∈# init-clss S›
      have ‹total-over-m (lits-of-l (trail S)) {C}›
        using 2 C by (auto dest!: multi-member-split)
      moreover have ‹¬ trail S ⊨as CNot C›
        using C nsc conflict-rule[of S C, OF - - - state-eq-ref]
        by (auto simp: clauses-def dest!: multi-member-split)
      ultimately show ‹trail S ⊨a C›
        using total-not-CNot[of ‹lits-of-l (trail S)› C] unfolding true-annots-true-cls true-annot-def
        by auto
    qed
    have 4: ‹lit-of `# mset (trail S) ∈ simple-clss (atms-of-mm (init-clss S))›
      using tauto cons incl dist by (auto simp: simple-clss-def)

    have [intro]: ‹ϱ (lit-of `# mset M') = ϱ (lit-of `# mset (trail S))›
      if
        ‹lit-of `# mset (trail S) ∈ simple-clss (atms-of-mm (init-clss S))› and
        ‹atms-of (lit-of `# mset (trail S)) ⊆ atms-of-mm (init-clss S)› and
        ‹no-dup (trail S)› and
        ‹total-over-m (lits-of-l M') (set-mset (init-clss S))› and
        incl: ‹mset (trail S) ⊆# mset M'› and
        ‹lit-of `# mset M' ∈ simple-clss (atms-of-mm (init-clss S))›
      for M' :: ‹('v literal, 'v literal, 'v literal multiset) annotated-lit list›
    proof −
      have [simp]: ‹lits-of-l M' = set-mset (lit-of `# mset M')›
        by (auto simp: lits-of-def)
      obtain A where A: ‹mset M' = A + mset (trail S)›
        using incl by (auto simp: mset-subset-eq-exists-conv)
      have M': ‹lits-of-l M' = lit-of ` set-mset A ∪ lits-of-l (trail S)›
        unfolding lits-of-def
        by (metis A image-Un set-mset-mset set-mset-union)
      have ‹mset M' = mset (trail S)›
        using that 2 unfolding A total-over-m-alt-def
          apply (case-tac A)
        apply (auto simp: A simple-clss-def distinct-mset-add M' image-Un
            tautology-union mset-inter-empty-set-mset atms-of-def atms-of-s-def
            atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set image-image
            tautology-add-mset)
          by (metis (no-types, lifting) atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set
          lits-of-def subsetCE)
      then show ?thesis
        using 2 by auto
    qed
    have imp: ‹is-improving (trail S) (trail S) S›
      using 1 2 3 4 incl n-d unfolding is-improving-int-def
      by (auto simp:  oconflict-opt.simps)

    show ‹False›
      using trail-is-improving-Ex-improve[of S, OF - imp] nsi by auto
  qed
  then show ?thesis
    using nsc nsp nsi nsco nso nsp nspr
    by (auto simp: cdcl-bnb.simps)
qed
```

lemma *no-step-ocdcl$_w$-p-stgy-no-step-cdcl-bnb-stgy*:
assumes ‹*no-step ocdcl$_w$-p-stgy S*› **and**
inv: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)›
shows ‹*no-step cdcl-bnb-stgy S*›
using *assms no-step-ocdcl$_w$-p-no-step-cdcl-bnb*[*of S*]
by (*auto simp*: *ocdcl$_w$-p-stgy.simps ocdcl$_w$-p.simps*
cdcl-bnb.simps cdcl-bnb-stgy.simps)

lemma *full-ocdcl$_w$-p-stgy-full-cdcl-bnb-stgy*:
assumes ‹*full ocdcl$_w$-p-stgy S T*› **and**
inv: ‹*cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*)›
shows ‹*full cdcl-bnb-stgy S T*›
using *assms rtranclp-ocdcl$_w$-p-stgy-rtranclp-cdcl-bnb-stgy*[*of S T*]
no-step-ocdcl$_w$-p-stgy-no-step-cdcl-bnb-stgy[*of T*]
unfolding *full-def*
by (*auto dest*: *rtranclp-cdcl-bnb-stgy-all-struct-inv*[*OF rtranclp-cdcl-bnb-stgy-cdcl-bnb*])

corollary *full-ocdcl$_w$-p-stgy-no-conflicting-clause-from-init-state*:
assumes
st: ‹*full ocdcl$_w$-p-stgy* (*init-state N*) *T*› **and**
dist: ‹*distinct-mset-mset N*›
shows
‹*weight T* = *None* $\Longrightarrow$ *unsatisfiable* (*set-mset N*)› **and**
‹*weight T* $\neq$ *None* $\Longrightarrow$ *model-on* (*set-mset* (*the* (*weight T*))) *N* $\wedge$ *set-mset* (*the* (*weight T*)) $\models$*sm N*
$\wedge$
distinct-mset (*the* (*weight T*))› **and**
‹*distinct-mset I* $\Longrightarrow$ *consistent-interp* (*set-mset I*) $\Longrightarrow$ *atms-of I* = *atms-of-mm N* $\Longrightarrow$
set-mset I $\models$*sm N* $\Longrightarrow$ *Found* (ϱ *I*) $\geq$ ϱ' (*weight T*)›
using *full-cdcl-bnb-stgy-no-conflicting-clause-from-init-state*[*of N T*,
OF full-ocdcl$_w$-p-stgy-full-cdcl-bnb-stgy[*OF st*] *dist*] *dist*
by (*auto simp*: *all-struct-init-state-distinct-iff model-on-def*
dest: *multi-member-split*)

lemma *cdcl-bnb-stgy-no-smaller-propa*:
‹*cdcl-bnb-stgy S T* $\Longrightarrow$ *cdcl$_W$-restart-mset.cdcl$_W$-all-struct-inv* (*abs-state S*) $\Longrightarrow$
no-smaller-propa S $\Longrightarrow$ *no-smaller-propa T*›
apply (*induction rule*: *cdcl-bnb-stgy.induct*)
subgoal
by (*auto simp*: *no-smaller-propa-def propagated-cons-eq-append-decide-cons conflict.simps*)
subgoal
by (*auto simp*: *no-smaller-propa-def propagated-cons-eq-append-decide-cons*
propagate.simps no-smaller-propa-tl elim!: *rulesE*)
subgoal
by (*auto simp*: *no-smaller-propa-def propagated-cons-eq-append-decide-cons*
improvep.simps elim!: *rulesE*)
subgoal
by (*auto simp*: *no-smaller-propa-def propagated-cons-eq-append-decide-cons*
conflict-opt.simps no-smaller-propa-tl elim!: *rulesE*)
subgoal for *T*
apply (*cases rule*: *ocdcl$_W$-o.cases*, *assumption*; *thin-tac* ‹*ocdcl$_W$-o S T*›)
subgoal
using *decide-no-smaller-step*[*of S T*] **unfolding** *no-confl-prop-impr.simps* **by** *auto*
subgoal
apply (*cases rule*: *cdcl-bnb-bj.cases*, *assumption*; *thin-tac* ‹*cdcl-bnb-bj S T*›)
subgoal

by (*use no-smaller-propa-tl*[*of S T*] **in** ‹*auto elim*: *rulesE*›)
subgoal
by (*use no-smaller-propa-tl*[*of S T*] **in** ‹*auto elim*: *rulesE*›)
subgoal
using *backtrackg-no-smaller-propa*[*OF obacktrack-backtrackg*, *of S T*]
unfolding $cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv-def*
$cdcl_W$*-restart-mset*.$cdcl_W$*-M-level-inv-def* $cdcl_W$*-restart-mset*.$cdcl_W$*-conflicting-def*
by (*auto elim*: *obacktrackE*)
done
done
done

lemma *rtranclp-cdcl-bnb-stgy-no-smaller-propa*:
‹*cdcl-bnb-stgy*** *S T* $\Longrightarrow$ $cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv* (*abs-state S*) $\Longrightarrow$
no-smaller-propa S $\Longrightarrow$ *no-smaller-propa T*›
by (*induction rule*: *rtranclp-induct*)
(*use rtranclp-cdcl-bnb-stgy-all-struct-inv*
rtranclp-cdcl-bnb-stgy-cdcl-bnb **in** ‹*force intro*: *cdcl-bnb-stgy-no-smaller-propa*›)+

lemma *wf-ocdcl*$_w$*-p*:
‹*wf* {(*T*, *S*). $cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv* (*abs-state S*)
$\wedge$ *ocdcl*$_w$*-p S T*}›
by (*rule wf-subset*[*OF wf-cdcl-bnb2*]) (*auto dest*: *ocdcl*$_w$*-p-cdcl-bnb*)

end

end
theory *CDCL-W-Partial-Encoding*
imports *CDCL-W-Optimal-Model*
begin

lemma *consistent-interp-unionI*:
‹*consistent-interp A* $\Longrightarrow$ *consistent-interp B* $\Longrightarrow$ ($\bigwedge a.\ a \in A \Longrightarrow -a \notin B$) $\Longrightarrow$ ($\bigwedge a.\ a \in B \Longrightarrow -a$
$\notin A$) $\Longrightarrow$
consistent-interp ($A \cup B$)›
by (*auto simp*: *consistent-interp-def*)

lemma *consistent-interp-poss*: ‹*consistent-interp* (*Pos* ‘ *A*)› **and**
consistent-interp-negs: ‹*consistent-interp* (*Neg* ‘ *A*)›
by (*auto simp*: *consistent-interp-def*)

lemma *Neg-in-lits-of-l-definedD*:
‹*Neg A* $\in$ *lits-of-l M* $\Longrightarrow$ *defined-lit M* (*Pos A*)›
by (*simp add*: *Decided-Propagated-in-iff-in-lits-of-l*)

0.1.2 Encoding of partial SAT into total SAT

As a way to make sure we don't reuse theorems names:

interpretation *test*: *conflict-driven-clause-learning*$_W$*-optimal-weight* **where**
state-eq = ‹(=)› **and**
state = *id* **and**
trail = ‹λ(*M*, *N*, *U*, *D*, *W*). *M*› **and**
init-clss = ‹λ(*M*, *N*, *U*, *D*, *W*). *N*› **and**
learned-clss = ‹λ(*M*, *N*, *U*, *D*, *W*). *U*› **and**

conflicting = ‹λ(*M*, *N*, *U*, *D*, *W*). *D*› **and**
cons-trail = ‹λ*K* (*M*, *N*, *U*, *D*, *W*). (*K* # *M*, *N*, *U*, *D*, *W*)› **and**
tl-trail = ‹λ(*M*, *N*, *U*, *D*, *W*). (*tl M*, *N*, *U*, *D*, *W*)› **and**
add-learned-cls = ‹λ*C* (*M*, *N*, *U*, *D*, *W*). (*M*, *N*, *add-mset C U*, *D*, *W*)› **and**
remove-cls = ‹λ*C* (*M*, *N*, *U*, *D*, *W*). (*M*, *removeAll-mset C N*, *removeAll-mset C U*, *D*, *W*)› **and**
update-conflicting = ‹λ*C* (*M*, *N*, *U*, -, *W*). (*M*, *N*, *U*, *C*, *W*)› **and**
init-state = ‹λ*N*. ([], *N*, {#}, *None*, *None*, ())› **and**
ϱ = ‹λ-. *0*› **and**
update-additional-info = ‹λ*W* (*M*, *N*, *U*, *D*, -, -). (*M*, *N*, *U*, *D*, *W*)›
by *unfold-locales* (*auto simp*: $state_W$*-ops.additional-info-def*)

We here formalise the encoding from a formula to another formula from which we will use to derive the optimal partial model.

While the proofs are still inspired by Dominic Zimmer's upcoming bachelor thesis, we now use the dual rail encoding, which is more elegant that the solution found by Christoph to solve the problem.

The intended meaning is the following:

- Σ is the set of all variables
- ΔΣ is the set of all variables with a (possibly non-zero) weight: These are the variable that needs to be replaced during encoding, but it does not matter if the weight 0.

locale *optimal-encoding-opt-ops* =
fixes Σ ΔΣ :: ‹*'v set*› **and**
new-vars :: ‹*'v* ⇒ *'v* × *'v*›
begin

abbreviation *replacement-pos* :: ‹*'v* ⇒ *'v*› (‹$(\text{-})^{\mapsto 1}$› *100*) **where**
‹*replacement-pos A* ≡ *fst* (*new-vars A*)›

abbreviation *replacement-neg* :: ‹*'v* ⇒ *'v*› (‹$(\text{-})^{\mapsto 0}$› *100*) **where**
‹*replacement-neg A* ≡ *snd* (*new-vars A*)›

fun *encode-lit* **where**
‹*encode-lit* (*Pos A*) = (*if A* ∈ ΔΣ *then Pos* (*replacement-pos A*) *else Pos A*)› |
‹*encode-lit* (*Neg A*) = (*if A* ∈ ΔΣ *then Pos* (*replacement-neg A*) *else Neg A*)›

lemma *encode-lit-alt-def*:
‹*encode-lit A* = (*if atm-of A* ∈ ΔΣ
then Pos (*if is-pos A then replacement-pos* (*atm-of A*) *else replacement-neg* (*atm-of A*))
else A)›
by (*cases A*) *auto*

definition *encode-clause* :: ‹*'v clause* ⇒ *'v clause*› **where**
‹*encode-clause C* = *encode-lit* '# *C*›

lemma *encode-clause-simp*[*simp*]:
‹*encode-clause* {#} = {#}›
‹*encode-clause* (*add-mset A C*) = *add-mset* (*encode-lit A*) (*encode-clause C*)›
‹*encode-clause* (*C* + *D*) = *encode-clause C* + *encode-clause D*›
by (*auto simp*: *encode-clause-def*)

definition *encode-clauses* :: ‹'*v clauses* ⇒ '*v clauses*› **where**
‹*encode-clauses C* = *encode-clause* '# *C*›

lemma *encode-clauses-simp*[*simp*]:
‹*encode-clauses* {#} = {#}›
‹*encode-clauses* (*add-mset A C*) = *add-mset* (*encode-clause A*) (*encode-clauses C*)›
‹*encode-clauses* (*C* + *D*) = *encode-clauses C* + *encode-clauses D*›
by (*auto simp*: *encode-clauses-def*)

definition *additional-constraint* :: ‹'*v* ⇒ '*v clauses*› **where**
‹*additional-constraint A* =
{#{#*Neg* ($A^{\mapsto 1}$), *Neg* ($A^{\mapsto 0}$)#}#}›

definition *additional-constraints* :: ‹'*v clauses*› **where**
‹*additional-constraints* = $\sum_{\#}$(*additional-constraint* '# (*mset-set* ΔΣ))›

definition *penc* :: ‹'*v clauses* ⇒ '*v clauses*› **where**
‹*penc N* = *encode-clauses N* + *additional-constraints*›

lemma *size-encode-clauses*[*simp*]: ‹*size* (*encode-clauses N*) = *size N*›
by (*auto simp*: *encode-clauses-def*)

lemma *size-penc*:
‹*size* (*penc N*) = *size N* + *card* ΔΣ›
by (*auto simp*: *penc-def additional-constraints-def*
additional-constraint-def size-Union-mset-image-mset)

lemma *atms-of-mm-additional-constraints*: ‹*finite* ΔΣ ⟹
atms-of-mm additional-constraints = *replacement-pos* ' ΔΣ ∪ *replacement-neg* ' ΔΣ›
by (*auto simp*: *additional-constraints-def additional-constraint-def atms-of-ms-def*)

lemma *atms-of-mm-encode-clause-subset*:
‹*atms-of-mm* (*encode-clauses N*) ⊆ (*atms-of-mm N* − ΔΣ) ∪ *replacement-pos* ' {*A* ∈ ΔΣ. *A* ∈ *atms-of-mm N*}
∪ *replacement-neg* ' {*A* ∈ ΔΣ. *A* ∈ *atms-of-mm N*}›
by (*auto simp*: *encode-clauses-def encode-lit-alt-def atms-of-ms-def atms-of-def*
encode-clause-def split: *if-splits*
dest!: *multi-member-split*[*of* - *N*])

In every meaningful application of the theorem below, we have ΔΣ ⊆ *atms-of-mm N*.

lemma *atms-of-mm-penc-subset*: ‹*finite* ΔΣ ⟹
atms-of-mm (*penc N*) ⊆ *atms-of-mm N* ∪ *replacement-pos* ' ΔΣ
∪ *replacement-neg* ' ΔΣ ∪ ΔΣ›
using *atms-of-mm-encode-clause-subset*[*of N*]
by (*auto simp*: *penc-def atms-of-mm-additional-constraints*)

lemma *atms-of-mm-encode-clause-subset2*: ‹*finite* ΔΣ ⟹ ΔΣ ⊆ *atms-of-mm N* ⟹
atms-of-mm N ⊆ *atms-of-mm* (*encode-clauses N*) ∪ ΔΣ›
by (*auto simp*: *encode-clauses-def encode-lit-alt-def atms-of-ms-def atms-of-def*
encode-clause-def split: *if-splits*
dest!: *multi-member-split*[*of* - *N*])

lemma *atms-of-mm-penc-subset2*: ‹*finite* ΔΣ ⟹ ΔΣ ⊆ *atms-of-mm N* ⟹
atms-of-mm (*penc N*) = (*atms-of-mm N* − ΔΣ) ∪ *replacement-pos* ' ΔΣ ∪ *replacement-neg* ' ΔΣ›
using *atms-of-mm-encode-clause-subset*[*of N*] *atms-of-mm-encode-clause-subset2*[*of N*]
by (*auto simp*: *penc-def atms-of-mm-additional-constraints*)

```
theorem card-atms-of-mm-penc:
  assumes ‹finite ΔΣ› and ‹ΔΣ ⊆ atms-of-mm N›
  shows ‹card (atms-of-mm (penc N)) ≤ card (atms-of-mm N − ΔΣ) + 2 ∗ card ΔΣ› (is ‹?A ≤ ?B›)
proof −
  have ‹?A = card
     ((atms-of-mm N − ΔΣ) ∪ replacement-pos ‘ ΔΣ ∪
      replacement-neg ‘ ΔΣ)› (is ‹- = card (?W ∪ ?X ∪ ?Y)›)
    using arg-cong[OF atms-of-mm-penc-subset2[of N], of card] assms card-Un-le
    by auto
  also have ‹... ≤ card (?W ∪ ?X) + card ?Y›
    using card-Un-le[of ‹?W ∪ ?X› ?Y] by auto
  also have ‹... ≤ card ?W + card ?X + card ?Y›
    using card-Un-le[of ‹?W› ?X] by auto
  also have ‹... ≤  card (atms-of-mm N − ΔΣ) + 2 ∗ card ΔΣ›
    using card-mono[of ‹atms-of-mm N› ‹ΔΣ›] assms
      card-image-le[of ΔΣ replacement-pos] card-image-le[of ΔΣ replacement-neg]
    by auto
  finally show ?thesis .
qed

definition postp :: ‹'v partial-interp ⇒ 'v partial-interp› where
  ‹postp I =
     {A ∈ I. atm-of A ∉ ΔΣ ∧ atm-of A ∈ Σ} ∪ Pos ‘ {A. A ∈ ΔΣ ∧ Pos (replacement-pos A) ∈ I}
      ∪ Neg ‘ {A. A ∈ ΔΣ ∧ Pos (replacement-neg A) ∈ I ∧ Pos (replacement-pos A) ∉ I}›

lemma preprocess-clss-model-additional-variables2:
  assumes
    ‹atm-of A ∈ Σ − ΔΣ›
  shows
    ‹A ∈ postp I ⟷ A ∈ I› (is ?A)
proof −
  show ?A
    using assms
    by (auto simp: postp-def)
qed

lemma encode-clause-iff:
  assumes
    ‹⋀A. A ∈ ΔΣ ⟹ Pos A ∈ I ⟷ Pos (replacement-pos A) ∈ I›
    ‹⋀A. A ∈ ΔΣ ⟹ Neg A ∈ I ⟷ Pos (replacement-neg A) ∈ I›
  shows ‹I ⊨ encode-clause C ⟷ I ⊨ C›
  using assms
  apply (induction C)
  subgoal by auto
  subgoal for A C
    by (cases A)
      (auto simp: encode-clause-def encode-lit-alt-def split: if-splits)
  done

lemma encode-clauses-iff:
  assumes
    ‹⋀A. A ∈ ΔΣ ⟹ Pos A ∈ I ⟷ Pos (replacement-pos A) ∈ I›
    ‹⋀A. A ∈ ΔΣ ⟹ Neg A ∈ I ⟷ Pos (replacement-neg A) ∈ I›
  shows ‹I ⊨m encode-clauses C ⟷ I ⊨m C›
  using encode-clause-iff[OF assms]
```

by (*auto simp*: *encode-clauses-def true-cls-mset-def*)

definition Σ_{add} **where**
‹Σ_{add} = *replacement-pos* ‘ $\Delta\Sigma$ ∪ *replacement-neg* ‘ $\Delta\Sigma$›

definition *upostp* :: ‹′*v partial-interp* ⇒ ′*v partial-interp*› **where**
‹*upostp I* =
Neg ‘ $\{A \in \Sigma.\ A \notin \Delta\Sigma \wedge Pos\ A \notin I \wedge Neg\ A \notin I\}$
$\cup \{A \in I.\ atm\text{-}of\ A \in \Sigma \wedge atm\text{-}of\ A \notin \Delta\Sigma\}$
∪ *Pos* ‘ *replacement-pos* ‘ $\{A \in \Delta\Sigma.\ Pos\ A \in I\}$
∪ *Neg* ‘ *replacement-pos* ‘ $\{A \in \Delta\Sigma.\ Pos\ A \notin I\}$
∪ *Pos* ‘ *replacement-neg* ‘ $\{A \in \Delta\Sigma.\ Neg\ A \in I\}$
∪ *Neg* ‘ *replacement-neg* ‘ $\{A \in \Delta\Sigma.\ Neg\ A \notin I\}$›

lemma *atm-of-upostp-subset*:
‹*atm-of* ‘ (*upostp I*) ⊆
(*atm-of* ‘ $I - \Delta\Sigma$) ∪ *replacement-pos* ‘ $\Delta\Sigma$ ∪
replacement-neg ‘ $\Delta\Sigma \cup \Sigma$›
by (*auto simp*: *upostp-def image-Un*)

end

locale *optimal-encoding-opt* = *conflict-driven-clause-learning*$_W$*-optimal-weight*
state-eq
state
— functions for the state:
— access functions:
trail init-clss learned-clss conflicting
— changing state:
cons-trail tl-trail add-learned-cls remove-cls
update-conflicting

— get state:
init-state ϱ
update-additional-info +
optimal-encoding-opt-ops Σ $\Delta\Sigma$ *new-vars*
for
state-eq :: ‹′*st* ⇒ ′*st* ⇒ *bool*› (**infix** ‹∼› *50*) **and**
state :: ′*st* ⇒ (′*v*, ′*v clause*) *ann-lits* × ′*v clauses* × ′*v clauses* × ′*v clause option* ×
′*v clause option* × ′*b* **and**
trail :: ‹′*st* ⇒ (′*v*, ′*v clause*) *ann-lits*› **and**
init-clss :: ‹′*st* ⇒ ′*v clauses*› **and**
learned-clss :: ‹′*st* ⇒ ′*v clauses*› **and**
conflicting :: ‹′*st* ⇒ ′*v clause option*› **and**

cons-trail :: ‹(′*v*, ′*v clause*) *ann-lit* ⇒ ′*st* ⇒ ′*st*› **and**
tl-trail :: ‹′*st* ⇒ ′*st*› **and**
add-learned-cls :: ‹′*v clause* ⇒ ′*st* ⇒ ′*st*› **and**
remove-cls :: ‹′*v clause* ⇒ ′*st* ⇒ ′*st*› **and**
update-conflicting :: ‹′*v clause option* ⇒ ′*st* ⇒ ′*st*› **and**

init-state :: ‹′*v clauses* ⇒ ′*st*› **and**
update-additional-info :: ‹′*v clause option* × ′*b* ⇒ ′*st* ⇒ ′*st*› **and**

Σ ΔΣ :: ‹'v set› **and**
ϱ :: ‹'v clause ⇒ 'a :: {linorder}› **and**
new-vars :: ‹'v ⇒ 'v × 'v›
begin

inductive *odecide* :: ‹'st ⇒ 'st ⇒ bool› **where**
odecide-noweight: ‹*odecide S T*›
if
‹*conflicting S* = *None*› **and**
‹*undefined-lit* (*trail S*) *L*› **and**
‹*atm-of L* ∈ *atms-of-mm* (*init-clss S*)› **and**
‹*T* ∼ *cons-trail* (*Decided L*) *S*› **and**
‹*atm-of L* ∈ Σ − ΔΣ› |
odecide-replacement-pos: ‹*odecide S T*›
if
‹*conflicting S* = *None*› **and**
‹*undefined-lit* (*trail S*) (*Pos* (*replacement-pos L*))› **and**
‹*T* ∼ *cons-trail* (*Decided* (*Pos* (*replacement-pos L*))) *S*› **and**
‹*L* ∈ ΔΣ› |
odecide-replacement-neg: ‹*odecide S T*›
if
‹*conflicting S* = *None*› **and**
‹*undefined-lit* (*trail S*) (*Pos* (*replacement-neg L*))› **and**
‹*T* ∼ *cons-trail* (*Decided* (*Pos* (*replacement-neg L*))) *S*› **and**
‹*L* ∈ ΔΣ›

inductive-cases *odecideE*: ‹*odecide S T*›

definition *no-new-lonely-clause* :: ‹'v clause ⇒ bool› **where**
‹*no-new-lonely-clause C* ⟷
(∀ *L* ∈ ΔΣ. *L* ∈ *atms-of C* ⟶
Neg (*replacement-pos L*) ∈# *C* ∨ *Neg* (*replacement-neg L*) ∈# *C* ∨ *C* ∈# *additional-constraint* *L*)›

definition *lonely-weighted-lit-decided* **where**
‹*lonely-weighted-lit-decided S* ⟷
(∀ *L* ∈ ΔΣ. *Decided* (*Pos L*) ∉ *set* (*trail S*) ∧ *Decided* (*Neg L*) ∉ *set* (*trail S*))›

end

locale *optimal-encoding-ops* = *optimal-encoding-opt-ops*
Σ ΔΣ
new-vars +
ocdcl-weight ϱ
for
Σ ΔΣ :: ‹'v set› **and**
new-vars :: ‹'v ⇒ 'v × 'v› **and**
ϱ :: ‹'v clause ⇒ 'a :: {linorder}› +
assumes
finite-Σ:
‹*finite* ΔΣ› **and**
ΔΣ-Σ:
‹ΔΣ ⊆ Σ› **and**
new-vars-pos:
‹*A* ∈ ΔΣ ⟹ *replacement-pos A* ∉ Σ› **and**

new-vars-neg:
‹$A \in \Delta\Sigma \Longrightarrow$ *replacement-neg* $A \notin \Sigma$› **and**
new-vars-dist:
‹*inj-on replacement-pos* $\Delta\Sigma$›
‹*inj-on replacement-neg* $\Delta\Sigma$›
‹*replacement-pos* ‘ $\Delta\Sigma \cap$ *replacement-neg* ‘ $\Delta\Sigma = \{\}$› **and**
Σ-*no-weight*:
‹*atm-of* $C \in \Sigma - \Delta\Sigma \Longrightarrow \varrho$ (*add-mset* C M) $= \varrho$ M›
begin

lemma *new-vars-dist2*:
‹$A \in \Delta\Sigma \Longrightarrow B \in \Delta\Sigma \Longrightarrow A \neq B \Longrightarrow$ *replacement-pos* $A \neq$ *replacement-pos* B›
‹$A \in \Delta\Sigma \Longrightarrow B \in \Delta\Sigma \Longrightarrow A \neq B \Longrightarrow$ *replacement-neg* $A \neq$ *replacement-neg* B›
‹$A \in \Delta\Sigma \Longrightarrow B \in \Delta\Sigma \Longrightarrow$ *replacement-neg* $A \neq$ *replacement-pos* B›
using *new-vars-dist* **unfolding** *inj-on-def* **apply** *blast*
using *new-vars-dist* **unfolding** *inj-on-def* **apply** *blast*
using *new-vars-dist* **unfolding** *inj-on-def* **apply** *blast*
done

lemma *consistent-interp-postp*:
‹*consistent-interp* $I \Longrightarrow$ *consistent-interp* (*postp* I)›
by (*auto simp*: *consistent-interp-def postp-def uminus-lit-swap*)

The reverse of the previous theorem does not hold due to the filtering on the variables of $\Delta\Sigma$. One example of version that holds:

lemma
assumes ‹$A \in \Delta\Sigma$›
shows ‹*consistent-interp* (*postp* {*Pos A* , *Neg A*})› **and**
‹¬*consistent-interp* {*Pos A*, *Neg A*}›
using *assms* $\Delta\Sigma$-Σ
by (*auto simp*: *consistent-interp-def postp-def uminus-lit-swap*)

Some more restricted version of the reverse hold, like:

lemma *consistent-interp-postp-iff*:
‹*atm-of* ‘ $I \subseteq \Sigma - \Delta\Sigma \Longrightarrow$ *consistent-interp* $I \longleftrightarrow$ *consistent-interp* (*postp* I)›
by (*auto simp*: *consistent-interp-def postp-def uminus-lit-swap*)

lemma *new-vars-different-iff* [*simp*]:
‹$A \neq x^{\mapsto 1}$›
‹$A \neq x^{\mapsto 0}$›
‹$x^{\mapsto 1} \neq A$›
‹$x^{\mapsto 0} \neq A$›
‹$A^{\mapsto 0} \neq x^{\mapsto 1}$›
‹$A^{\mapsto 1} \neq x^{\mapsto 0}$›
‹$A^{\mapsto 0} = x^{\mapsto 0} \longleftrightarrow A = x$›
‹$A^{\mapsto 1} = x^{\mapsto 1} \longleftrightarrow A = x$›
‹$(A^{\mapsto 1}) \notin \Sigma$›
‹$(A^{\mapsto 0}) \notin \Sigma$›
‹$(A^{\mapsto 1}) \notin \Delta\Sigma$›
‹$(A^{\mapsto 0}) \notin \Delta\Sigma$›**if** ‹$A \in \Delta\Sigma$› ‹$x \in \Delta\Sigma$› **for** A x
using $\Delta\Sigma$-Σ *new-vars-pos*[*of x*] *new-vars-pos*[*of A*] *new-vars-neg*[*of x*] *new-vars-neg*[*of A*]
new-vars-neg new-vars-dist2[*of A x*] *new-vars-dist2*[*of x A*] *that*
by (*cases* ‹$A = x$›; *fastforce simp*: *comp-def*; *fail*)+

lemma *consistent-interp-upostp*:

‹*consistent-interp* $I \Longrightarrow$ *consistent-interp* (*upostp* I)›
using $\Delta\Sigma$-Σ
by (*auto simp*: *consistent-interp-def upostp-def uminus-lit-swap*)

lemma *atm-of-upostp-subset2*:
‹*atm-of* ‘ $I \subseteq \Sigma \Longrightarrow$ *replacement-pos* ‘ $\Delta\Sigma$ ∪
replacement-neg ‘ $\Delta\Sigma \cup (\Sigma - \Delta\Sigma) \subseteq$ *atm-of* ‘ (*upostp* I)›
apply (*auto simp*: *upostp-def image-Un image-image*)
apply (*metis* (*mono-tags*, *lifting*) *imageI literal.sel*(*1*) *mem-Collect-eq*)
apply (*metis* (*mono-tags*, *lifting*) *imageI literal.sel*(*2*) *mem-Collect-eq*)
done

lemma $\Delta\Sigma$-*notin-upost*[*simp*]:
‹$y \in \Delta\Sigma \Longrightarrow$ *Neg* $y \notin$ *upostp* I›
‹$y \in \Delta\Sigma \Longrightarrow$ *Pos* $y \notin$ *upostp* I›
using $\Delta\Sigma$-Σ **by** (*auto simp*: *upostp-def*)

lemma *penc-ent-upostp*:
assumes Σ: ‹*atms-of-mm* $N = \Sigma$› **and**
sat: ‹$I \models sm$ N› **and**
cons: ‹*consistent-interp* I› **and**
atm: ‹*atm-of* ‘ $I \subseteq$ *atms-of-mm* N›
shows ‹*upostp* $I \models m$ *penc* N›
proof −
have [*iff*]: ‹*Pos* ($A^{\mapsto 0}$) $\notin I$› ‹*Pos* ($A^{\mapsto 1}$) $\notin I$›
‹*Neg* ($A^{\mapsto 0}$) $\notin I$› ‹*Neg* ($A^{\mapsto 1}$) $\notin I$› **if** ‹$A \in \Delta\Sigma$› **for** A
using *atm new-vars-neg*[*of A*] *new-vars-pos*[*of A*] *that*
unfolding Σ **by** *force+*
have *enc*: ‹*upostp* $I \models m$ *encode-clauses* N›
unfolding *true-cls-mset-def*
proof
fix C
assume ‹$C \in\#$ *encode-clauses* N›
then obtain C' **where**
‹$C' \in\# N$› **and**
‹$C =$ *encode-clause* C'›
by (*auto simp*: *encode-clauses-def*)
then obtain A **where**
‹$A \in\# C'$› **and**
‹$A \in I$›
using *sat*
by (*auto simp*: *true-cls-def*
dest!: *multi-member-split*[*of - N*])
moreover have ‹*atm-of* $A \in \Sigma - \Delta\Sigma \lor$ *atm-of* $A \in \Delta\Sigma$›
using *atm* ‹$A \in I$› **unfolding** Σ **by** *blast*
ultimately have ‹*encode-lit* $A \in$ *upostp* I›
by (*auto simp*: *encode-lit-alt-def upostp-def*)
then show ‹*upostp* $I \models C$›
using ‹$A \in\# C'$›
unfolding ‹$C =$ *encode-clause* C'›
by (*auto simp*: *encode-clause-def dest*: *multi-member-split*)
qed
have [*iff*]: ‹*Pos* ($y^{\mapsto 1}$) $\notin$ *upostp* $I \longleftrightarrow$ *Neg* ($y^{\mapsto 1}$) $\in$ *upostp* I›
‹*Pos* ($y^{\mapsto 0}$) $\notin$ *upostp* $I \longleftrightarrow$ *Neg* ($y^{\mapsto 0}$) $\in$ *upostp* I›

```
    if ‹y ∈ ΔΣ› for y
    using that
    by (cases ‹Pos y ∈ I›; auto simp: upostp-def image-image; fail)+
  have H:
    ‹Neg (y^↦0) ∉ upostp I ⟹ Neg (y^↦1) ∈ upostp I›
    if ‹y ∈ ΔΣ› for y
    using that cons ΔΣ-Σ unfolding upostp-def consistent-interp-def
    by (cases ‹Pos y ∈ I›) (auto simp: image-image)
  have [dest]: ‹Neg A ∈ upostp I ⟹ Pos A ∉ upostp I›
    ‹Pos A ∈ upostp I ⟹ Neg A ∉ upostp I› for A
    using consistent-interp-upostp[OF cons]
    by (auto simp: consistent-interp-def)

  have add: ‹upostp I ⊨m additional-constraints›
    using finite-Σ H
    by (auto simp: additional-constraints-def true-cls-mset-def additional-constraint-def)

  show ‹upostp I ⊨m penc N›
    using enc add unfolding penc-def by auto
qed

lemma penc-ent-postp:
  assumes Σ: ‹atms-of-mm N = Σ› and
    sat: ‹I ⊨sm penc N› and
    cons: ‹consistent-interp I›
  shows ‹postp I ⊨m N›
proof -
  have enc: ‹I ⊨m encode-clauses N› and ‹I ⊨m additional-constraints›
    using sat unfolding penc-def
    by auto
  have [dest]: ‹Pos (x2^↦0) ∈ I ⟹ Neg (x2^↦1) ∈ I› if ‹x2 ∈ ΔΣ› for x2
    using ‹I ⊨m additional-constraints› that cons
    multi-member-split[of x2 ‹mset-set ΔΣ›] finite-Σ
    unfolding additional-constraints-def additional-constraint-def
      consistent-interp-def
    by (auto simp: true-cls-mset-def)
  have [dest]: ‹Pos (x2^↦0) ∈ I ⟹ Pos (x2^↦1) ∉ I› if ‹x2 ∈ ΔΣ› for x2
    using that cons
    unfolding consistent-interp-def
    by auto

  show ‹postp I ⊨m N›
    unfolding true-cls-mset-def
  proof
    fix C
    assume ‹C ∈# N›
    then have ‹I ⊨ encode-clause C›
      using enc by (auto dest!: multi-member-split)
    then show ‹postp I ⊨ C›
      unfolding true-cls-def
      using cons finite-Σ sat
        preprocess-clss-model-additional-variables2[of - I]
        Σ ‹C ∈# N› in-m-in-literals
      apply (auto simp: encode-clause-def postp-def encode-lit-alt-def
          split: if-splits
          dest!: multi-member-split[of - C])
```

using *image-iff* **apply** *fastforce*
apply (*case-tac xa*; *auto*)
apply *auto*
done

qed
qed

lemma *satisfiable-penc-satisfiable*:
assumes Σ: ‹*atms-of-mm N* = Σ› **and**
sat: ‹*satisfiable* (*set-mset* (*penc N*))›
shows ‹*satisfiable* (*set-mset N*)›
using *assms* **apply** (*subst* (*asm*) *satisfiable-def*)
apply *clarify*
subgoal for *I*
using *penc-ent-postp*[*OF* Σ, *of I*] *consistent-interp-postp*[*of I*]
by *auto*
done

lemma *satisfiable-penc*:
assumes Σ: ‹*atms-of-mm N* = Σ› **and**
sat: ‹*satisfiable* (*set-mset N*)›
shows ‹*satisfiable* (*set-mset* (*penc N*))›
using *assms*
apply (*subst* (*asm*) *satisfiable-def-min*)
apply *clarify*
subgoal for *I*
using *penc-ent-upostp*[*of N I*] *consistent-interp-upostp*[*of I*]
by *auto*
done

lemma *satisfiable-penc-iff*:
assumes Σ: ‹*atms-of-mm N* = Σ›
shows ‹*satisfiable* (*set-mset* (*penc N*)) ⟷ *satisfiable* (*set-mset N*)›
using *assms satisfiable-penc satisfiable-penc-satisfiable* **by** *blast*

abbreviation ϱ_e*-filter* :: ‹*'v literal multiset* ⇒ *'v literal multiset*› **where**
‹ϱ_e*-filter M* ≡ {#*L* ∈# *poss* (*mset-set* ΔΣ). *Pos* (*atm-of* $L^{\mapsto 1}$) ∈# *M*#} +
{#*L* ∈# *negs* (*mset-set* ΔΣ). *Pos* (*atm-of* $L^{\mapsto 0}$) ∈# *M*#}›

lemma *finite-upostp*: ‹*finite I* ⟹ *finite* Σ ⟹ *finite* (*upostp I*)›
using *finite*-Σ ΔΣ-Σ
by (*auto simp*: *upostp-def*)

declare *finite*-Σ[*simp*]

lemma *encode-lit-eq-iff*:
‹*atm-of x* ∈ Σ ⟹ *atm-of y* ∈ Σ ⟹ *encode-lit x* = *encode-lit y* ⟷ *x* = *y*›
by (*cases x*; *cases y*) (*auto simp*: *encode-lit-alt-def atm-of-eq-atm-of*)

lemma *distinct-mset-encode-clause-iff*:
‹*atms-of N* ⊆ Σ ⟹ *distinct-mset* (*encode-clause N*) ⟷ *distinct-mset N*›
by (*induction N*)
(*auto simp*: *encode-clause-def encode-lit-eq-iff*
dest!: *multi-member-split*)

lemma *distinct-mset-encodes-clause-iff*:
‹*atms-of-mm N* ⊆ Σ ⟹ *distinct-mset-mset* (*encode-clauses N*) ⟷ *distinct-mset-mset N*›
by (*induction N*)
(*auto simp*: *encode-clauses-def distinct-mset-encode-clause-iff*)

lemma *distinct-additional-constraints*[*simp*]:
‹*distinct-mset-mset additional-constraints*›
by (*auto simp*: *additional-constraints-def additional-constraint-def*
distinct-mset-set-def)

lemma *distinct-mset-penc*:
‹*atms-of-mm N* ⊆ Σ ⟹ *distinct-mset-mset* (*penc N*) ⟷ *distinct-mset-mset N*›
by (*auto simp*: *penc-def*
distinct-mset-encodes-clause-iff)

lemma *finite-postp*: ‹*finite I* ⟹ *finite* (*postp I*)›
by (*auto simp*: *postp-def*)

lemma *total-entails-iff-no-conflict*:
assumes ‹*atms-of-mm N* ⊆ *atm-of* ‘ *I*› **and** ‹*consistent-interp I*›
shows ‹*I* ⊨*sm N* ⟷ (∀ *C* ∈# *N*. ¬*I* ⊨*s CNot C*)›
apply *rule*
subgoal
using *assms* **by** (*auto dest*!: *multi-member-split*
simp: *consistent-CNot-not*)
subgoal
by (*smt assms*(*1*) *atms-of-atms-of-ms-mono atms-of-ms-CNot-atms-of*
atms-of-ms-insert atms-of-ms-mono atms-of-s-def empty-iff
subset-iff sup.orderE total-not-true-cls-true-clss-CNot
total-over-m-alt-def true-clss-def)
done

definition ϱ_e :: ‹′*v literal multiset* ⇒ ′*a* :: {*linorder*}› **where**
‹ϱ_e *M* = ϱ (ϱ_e-*filter M*)›

lemma Σ-*no-weight*-ϱ_e: ‹*atm-of C* ∈ Σ − ΔΣ ⟹ ϱ_e (*add-mset C M*) = ϱ_e *M*›
using Σ-*no-weight*[*of C* ‹ϱ_e-*filter M*›]
apply (*auto simp*: ϱ_e-*def finite*-Σ *image-mset-mset-set inj-on-Neg inj-on-Pos*)
by (*smt Collect-cong image-iff literal.sel*(*1*) *literal.sel*(*2*) *new-vars-neg new-vars-pos*)

lemma ϱ-*cancel-notin*-ΔΣ:
‹(⋀*x*. *x* ∈# *M* ⟹ *atm-of x* ∈ Σ − ΔΣ) ⟹ ϱ (*M* + *M*′) = ϱ *M*′›
by (*induction M*) (*auto simp*: Σ-*no-weight*)

lemma ϱ-*mono2*:
‹*consistent-interp* (*set-mset M*′) ⟹ *distinct-mset M*′ ⟹
(⋀*A*. *A* ∈# *M* ⟹ *atm-of A* ∈ Σ) ⟹ (⋀*A*. *A* ∈# *M*′ ⟹ *atm-of A* ∈ Σ) ⟹
{#*A* ∈# *M*. *atm-of A* ∈ ΔΣ#} ⊆# {#*A* ∈# *M*′. *atm-of A* ∈ ΔΣ#} ⟹ ϱ *M* ≤ ϱ *M*′›
apply (*subst* (*2*) *multiset-partition*[*of* - ‹λ*A*. *atm-of A* ∉ ΔΣ›])
apply (*subst multiset-partition*[*of* - ‹λ*A*. *atm-of A* ∉ ΔΣ›])
apply (*subst* ϱ-*cancel-notin*-ΔΣ)
subgoal by *auto*
apply (*subst* ϱ-*cancel-notin*-ΔΣ)
subgoal by *auto*
by (*auto intro*!: ϱ-*mono intro*: *consistent-interp-subset intro*!: *distinct-mset-mono*[*of* - *M*′])

lemma ϱ_e*-mono*: ‹*distinct-mset B* $\Longrightarrow$ *A* $\subseteq\#$ *B* $\Longrightarrow$ ϱ_e *A* $\leq$ ϱ_e *B*›
  unfolding ϱ_e*-def*
  apply (*rule* ϱ*-mono*)
  subgoal
    by (*subst distinct-mset-add*)
      (*auto simp*: *distinct-image-mset-inj distinct-mset-filter distinct-mset-mset-set inj-on-Pos mset-inter-empty-set-mset image-mset-mset-set inj-on-Neg*)
  subgoal
    by (*rule subset-mset.add-mono*; *rule filter-mset-mono-subset*) *auto*
  done

lemma ϱ_e*-upostp-*ϱ:
  assumes [*simp*]: ‹*finite* Σ› **and**
    ‹*finite I*› **and**
    cons: ‹*consistent-interp I*› **and**
    I-Σ: ‹*atm-of* ‘ *I* ⊆ Σ›
  shows ‹ϱ_e (*mset-set* (*upostp I*)) = ϱ (*mset-set I*)› (**is** ‹*?A* = *?B*›)
proof −
  have [*simp*]: ‹*finite I*›
    using *assms* **by** *auto*
  have [*simp*]: ‹*mset-set*
        {*x* ∈ *I*.
         atm-of x ∈ Σ ∧
         atm-of x ∉ *replacement-pos* ‘ ΔΣ ∧
         atm-of x ∉ *replacement-neg* ‘ ΔΣ} = *mset-set I*›
    using *I*-Σ **by** *auto*
  have [*simp*]: ‹*finite* {*A* ∈ ΔΣ. *P A*}› **for** *P*
    by (*rule finite-subset*[*of* - ΔΣ])
      (*use* ΔΣ-Σ *finite*-Σ **in** *auto*)
  have [*dest*]: ‹*xa* ∈ ΔΣ $\Longrightarrow$ *Pos* ($xa^{\mapsto 1}$) ∈ *upostp I* $\Longrightarrow$ *Pos* ($xa^{\mapsto 0}$) ∈ *upostp I* $\Longrightarrow$ *False*› **for** *xa*
    using *cons* **unfolding** *penc-def*
    by (*auto simp*: *additional-constraint-def additional-constraints-def true-cls-mset-def consistent-interp-def upostp-def*)
  have ‹*?A* ≤ *?B*›
    using *assms* ΔΣ-Σ **apply** −
    unfolding ϱ_e*-def filter-filter-mset*
    apply (*rule* ϱ*-mono2*)
    subgoal using *cons* **by** *auto*
    subgoal using *distinct-mset-mset-set* **by** *auto*
    subgoal by *auto*
    subgoal by *auto*
    apply (*rule filter-mset-mono-subset*)
    subgoal
      by (*subst distinct-subseteq-iff*[*symmetric*])
        (*auto simp*: *upostp-def simp*: *image-mset-mset-set inj-on-Neg inj-on-Pos distinct-mset-add mset-inter-empty-set-mset distinct-mset-mset-set*)
    subgoal for *x*
      by (*cases* ‹*x* ∈ *I*›; *cases x*) (*auto simp*: *upostp-def*)
    done
  moreover have ‹*?B* ≤ *?A*›
    using *assms* ΔΣ-Σ **apply** −
    unfolding ϱ_e*-def filter-filter-mset*
    apply (*rule* ϱ*-mono2*)
    subgoal using *cons* **by** (*auto intro*:

```
      intro: consistent-interp-subset[of - ‹Pos ‘ ΔΣ›]
      intro: consistent-interp-subset[of - ‹Neg ‘ ΔΣ›]
      intro!: consistent-interp-unionI
      simp: consistent-interp-upostp finite-upostp consistent-interp-poss
        consistent-interp-negs)
    subgoal by (auto
      simp: distinct-mset-mset-set distinct-mset-add image-mset-mset-set inj-on-Pos inj-on-Neg
        mset-inter-empty-set-mset)
    subgoal by auto
    subgoal by auto
    apply (auto simp: image-mset-mset-set inj-on-Neg inj-on-Pos)
      apply (subst distinct-subseteq-iff[symmetric])
     apply (auto simp: distinct-mset-mset-set distinct-mset-add image-mset-mset-set inj-on-Pos inj-on-Neg
        mset-inter-empty-set-mset finite-upostp)
        apply (metis image-eqI literal.exhaust-sel)
    apply (auto simp: upostp-def image-image)
    apply (metis (mono-tags, lifting) imageI literal.collapse(1) literal.collapse(2) mem-Collect-eq)
    apply (metis (mono-tags, lifting) imageI literal.collapse(1) literal.collapse(2) mem-Collect-eq)
    apply (metis (mono-tags, lifting) imageI literal.collapse(1) literal.collapse(2) mem-Collect-eq)
    done
  ultimately show ?thesis
    by simp
qed

end

locale optimal-encoding = optimal-encoding-opt
    state-eq
    state
    — functions for the state:
    — access functions:
    trail init-clss learned-clss conflicting
    — changing state:
    cons-trail tl-trail add-learned-cls remove-cls
    update-conflicting

    — get state:
    init-state
    update-additional-info
    Σ ΔΣ
    ϱ
    new-vars +
    optimal-encoding-ops
    Σ ΔΣ
    new-vars ϱ
  for
    state-eq :: ‹'st ⇒ 'st ⇒ bool› (infix ‹∼› 50) and
    state :: 'st ⇒ ('v, 'v clause) ann-lits × 'v clauses × 'v clauses × 'v clause option ×
        'v clause option × 'b and
    trail :: ‹'st ⇒ ('v, 'v clause) ann-lits› and
    init-clss :: ‹'st ⇒ 'v clauses› and
    learned-clss :: ‹'st ⇒ 'v clauses› and
    conflicting :: ‹'st ⇒ 'v clause option› and
    cons-trail :: ‹('v, 'v clause) ann-lit ⇒ 'st ⇒ 'st› and
    tl-trail :: ‹'st ⇒ 'st› and
    add-learned-cls :: ‹'v clause ⇒ 'st ⇒ 'st› and
```

remove-cls :: ‹′*v clause* ⇒ ′*st* ⇒ ′*st*› **and**
update-conflicting :: ‹′*v clause option* ⇒ ′*st* ⇒ ′*st*› **and**

init-state :: ‹′*v clauses* ⇒ ′*st*› **and**
ϱ :: ‹′*v clause* ⇒ ′*a* :: {*linorder*}› **and**
update-additional-info :: ‹′*v clause option* × ′*b* ⇒ ′*st* ⇒ ′*st*› **and**
Σ ΔΣ :: ‹′*v set*› **and**
new-vars :: ‹′*v* ⇒ ′*v* × ′*v*›
begin

interpretation *enc-weight-opt*: *conflict-driven-clause-learning*$_W$*-optimal-weight* **where**
state-eq = *state-eq* **and**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state* **and**
$\varrho = \varrho_e$ **and**
update-additional-info = *update-additional-info*
apply *unfold-locales*
subgoal by (*rule* ϱ_e*-mono*)
subgoal using *update-additional-info* **by** *fast*
subgoal using *weight-init-state* **by** *fast*
done

theorem *full-encoding-OCDCL-correctness*:
assumes
st: ‹*full enc-weight-opt.cdcl-bnb-stgy* (*init-state* (*penc N*)) *T*› **and**
dist: ‹*distinct-mset-mset N*› **and**
atms: ‹*atms-of-mm N* = Σ›
shows
‹*weight T* = *None* ⟹ *unsatisfiable* (*set-mset N*)› **and**
‹*weight T* ≠ *None* ⟹ *postp* (*set-mset* (*the* (*weight T*))) ⊨*sm N*›
‹*weight T* ≠ *None* ⟹ *distinct-mset I* ⟹ *consistent-interp* (*set-mset I*) ⟹
atms-of I ⊆ *atms-of-mm N* ⟹ *set-mset I* ⊨*sm N* ⟹
ϱ *I* ≥ ϱ (*mset-set* (*postp* (*set-mset* (*the* (*weight T*)))))›
‹*weight T* ≠ *None* ⟹ ϱ_e (*the* (*enc-weight-opt.weight T*)) =
ϱ (*mset-set* (*postp* (*set-mset* (*the* (*enc-weight-opt.weight T*)))))›
proof −
let *?N* = ‹*penc N*›
have ‹*distinct-mset-mset* (*penc N*)›
by (*subst distinct-mset-penc*)
(*use dist atms* **in** *auto*)
then have
unsat: ‹*weight T* = *None* ⟹ *unsatisfiable* (*set-mset ?N*)› **and**
model: ‹*weight T* ≠ *None* ⟹ *consistent-interp* (*set-mset* (*the* (*weight T*))) ∧
atms-of (*the* (*weight T*)) ⊆ *atms-of-mm ?N* ∧ *set-mset* (*the* (*weight T*)) ⊨*sm ?N* ∧
distinct-mset (*the* (*weight T*))› **and**
opt: ‹*distinct-mset I* ⟹ *consistent-interp* (*set-mset I*) ⟹ *atms-of I* = *atms-of-mm ?N* ⟹

set-mset I ⊨sm ?N ⟹ Found (ϱ_e *I*) ≥ *enc-weight-opt*.ϱ' (*weight T*)›
for *I*
using *enc-weight-opt.full-cdcl-bnb-stgy-no-conflicting-clause-from-init-state*[*of* ‹*penc N*› *T*, *OF st*]
by *fast+*

show ‹*unsatisfiable* (*set-mset N*)› **if** ‹*weight T* = *None*›
using *unsat*[*OF that*] *satisfiable-penc*[*OF atms*] **by** *blast*
let *?K* = ‹*postp* (*set-mset* (*the* (*weight T*)))›
show ‹*?K* ⊨*sm N*› **if** ‹*weight T* ≠ *None*›
using *penc-ent-postp*[*OF atms*, *of* ‹*set-mset* (*the* (*weight T*))›] *model*[*OF that*]
by *auto*

assume *Some*: ‹*weight T* ≠ *None*›
have *Some′*: ‹*enc-weight-opt.weight T* ≠ *None*›
using *Some* **by** *auto*
have *cons-K*: ‹*consistent-interp ?K*›
using *model Some* **by** (*auto simp*: *consistent-interp-postp*)
define *J* **where** ‹*J* = *the* (*weight T*)›
then have [*simp*]: ‹*weight T* = *Some J*› ‹*enc-weight-opt.weight T* = *Some J*›
using *Some* **by** *auto*
have ‹*set-mset J* ⊨*sm additional-constraints*›
using *model* **by** (*auto simp*: *penc-def*)
then have *H*: ‹*x* ∈ ΔΣ ⟹ *Neg* (*replacement-pos x*) ∈# *J* ∨ *Neg* (*replacement-neg x*) ∈# *J*› **and**
[*dest*]: ‹*Pos* ($xa^{\mapsto 1}$) ∈# *J* ⟹ *Pos* ($xa^{\mapsto 0}$) ∈# *J* ⟹ *xa* ∈ ΔΣ ⟹ *False*› **for** *x xa*
using *model*
apply (*auto simp*: *additional-constraints-def additional-constraint-def true-clss-def consistent-interp-def*)
by (*metis uminus-Pos*)
have *cons-f*: ‹*consistent-interp* (*set-mset* (ϱ_e*-filter* (*the* (*weight T*))))›
using *model*
by (*auto simp*: *postp-def* ϱ_e*-def* Σ_{add}*-def conj-disj-distribR*
consistent-interp-poss
consistent-interp-negs
mset-set-Union intro!: *consistent-interp-unionI*
intro: *consistent-interp-subset distinct-mset-mset-set*
consistent-interp-subset[*of* - ‹*Pos* ‘ ΔΣ›]
consistent-interp-subset[*of* - ‹*Neg* ‘ ΔΣ›])
have *dist-f*: ‹*distinct-mset* ((ϱ_e*-filter* (*the* (*weight T*))))›
using *model*
by (*auto simp*: *postp-def simp*: *image-mset-mset-set inj-on-Neg inj-on-Pos distinct-mset-add mset-inter-empty-set-mset distinct-mset-mset-set*)

have ‹*enc-weight-opt*.ϱ' (*weight T*) ≤ *Found* (ϱ (*mset-set ?K*))›
using *Some′*
apply *auto*
unfolding ϱ_e*-def*
apply (*rule* ϱ*-mono2*)
subgoal
using *model Some′* **by** (*auto simp*: *finite-postp consistent-interp-postp*)
subgoal by (*auto simp*: *distinct-mset-mset-set*)
subgoal using *atms dist model*[*OF Some*] *atms* ΔΣ-Σ **by** (*auto simp*: *postp-def*)
subgoal using *atms dist model*[*OF Some*] *atms* ΔΣ-Σ **by** (*auto simp*: *postp-def*)
subgoal
apply (*subst distinct-subseteq-iff*[*symmetric*])
using *dist model*[*OF Some*] *H*

```
    by (force simp: filter-filter-mset consistent-interp-def postp-def
          image-mset-mset-set inj-on-Neg inj-on-Pos finite-postp
          distinct-mset-add mset-inter-empty-set-mset distinct-mset-mset-set
        intro: distinct-mset-mono[of - ‹the (enc-weight-opt.weight T)›])+
  done
moreover {
  have ‹ϱ (mset-set ?K) ≤ ϱe (the (weight T))›
    unfolding ϱe-def
    apply (rule ϱ-mono2)
    subgoal by (rule cons-f)
    subgoal by (rule dist-f)
    subgoal using atms dist model[OF Some] atms ΔΣ-Σ by (auto simp: postp-def)
    subgoal using atms dist model[OF Some] atms ΔΣ-Σ by (auto simp: postp-def)
    subgoal
      by (subst distinct-subseteq-iff[symmetric])
      (auto simp: postp-def simp: image-mset-mset-set inj-on-Neg inj-on-Pos
         distinct-mset-add mset-inter-empty-set-mset distinct-mset-mset-set)
    done
  then have ‹Found (ϱ (mset-set ?K)) ≤ enc-weight-opt.ϱ′ (weight T)›
    using Some by auto
  } note le =this
ultimately show ‹ϱe (the (weight T)) = (ϱ (mset-set ?K))›
  using Some′ by auto

show ‹ϱ I ≥ ϱ (mset-set ?K)›
  if dist: ‹distinct-mset I› and
    cons: ‹consistent-interp (set-mset I)› and
    atm: ‹atms-of I ⊆ atms-of-mm N› and
    I-N: ‹set-mset I ⊨sm N›
proof –
  let ?I = ‹mset-set (upostp (set-mset I))›
  have [simp]: ‹finite (upostp (set-mset I))›
    by (rule finite-upostp)
      (use atms in auto)
  then have I: ‹set-mset ?I = upostp (set-mset I)›
    by auto
  have ‹set-mset ?I ⊨m ?N›
    unfolding I
    by (rule penc-ent-upostp[OF atms I-N cons])
      (use atm in ‹auto dest: multi-member-split›)
  moreover have ‹distinct-mset ?I›
    by (rule distinct-mset-mset-set)
  moreover {
    have A: ‹atms-of (mset-set (upostp (set-mset I))) = atm-of ‘ (upostp (set-mset I))›
      ‹atm-of ‘ set-mset I = atms-of I›
      by (auto simp: atms-of-def)
    have ‹atms-of ?I = atms-of-mm ?N›
      apply (subst atms-of-mm-penc-subset2[OF finite-Σ])
      subgoal using ΔΣ-Σ atms by auto
      subgoal
        using atm-of-upostp-subset[of ‹set-mset I›] atm-of-upostp-subset2[of ‹set-mset I›] atm
        unfolding atms A
        by (auto simp: upostp-def)
      done
  }
  moreover have cons′: ‹consistent-interp (set-mset ?I)›
```

```
      using cons unfolding I by (rule consistent-interp-upostp)
    ultimately have ‹Found (ϱ_e ?I) ≥ enc-weight-opt.ϱ′ (weight T)›
      using opt[of ?I] by auto
    moreover {
      have ‹ϱ_e ?I = ϱ (mset-set (set-mset I))›
        by (rule ϱ_e-upostp-ϱ)
          (use ΔΣ-Σ atms atm cons in ‹auto dest: multi-member-split›)
      then have ‹ϱ_e ?I = ϱ I›
        by (subst (asm) distinct-mset-set-mset-ident)
          (use atms dist in auto)
    }
    ultimately have ‹Found (ϱ I) ≥ enc-weight-opt.ϱ′ (weight T)›
      using Some′
      by auto
    moreover {
      have ‹ϱ_e (mset-set ?K) ≤ ϱ_e (mset-set (set-mset (the (weight T))))›
        unfolding ϱ_e-def
        apply (rule ϱ-mono2)
        subgoal using cons-f by auto
        subgoal using dist-f by auto
        subgoal using atms dist model[OF Some] atms ΔΣ-Σ by (auto simp: postp-def)
        subgoal using atms dist model[OF Some] atms ΔΣ-Σ by (auto simp: postp-def)
        subgoal
          by (subst distinct-subseteq-iff[symmetric])
          (auto simp: postp-def simp: image-mset-mset-set inj-on-Neg inj-on-Pos
              distinct-mset-add mset-inter-empty-set-mset distinct-mset-mset-set)
        done
      then have ‹Found (ϱ_e (mset-set ?K)) ≤ enc-weight-opt.ϱ′ (weight T)›
        apply (subst (asm) distinct-mset-set-mset-ident)
         apply (use atms dist model[OF Some] in auto; fail)[]
        using Some′ by auto
    }
    moreover have ‹ϱ_e (mset-set ?K) ≤ ϱ (mset-set ?K)›
      unfolding ϱ_e-def
      apply (rule ϱ-mono2)
      subgoal
        using model Some′ by (auto simp: finite-postp consistent-interp-postp)
      subgoal by (auto simp: distinct-mset-mset-set)
      subgoal using atms dist model[OF Some] atms ΔΣ-Σ by (auto simp: postp-def)
      subgoal using atms dist model[OF Some] atms ΔΣ-Σ by (auto simp: postp-def)
      subgoal
        by (subst distinct-subseteq-iff[symmetric])
          (auto simp: postp-def simp: image-mset-mset-set inj-on-Neg inj-on-Pos
              distinct-mset-add mset-inter-empty-set-mset distinct-mset-mset-set)
      done
    ultimately show ?thesis
      using Some′ le by auto
  qed
qed

theorem full-encoding-OCDCL-complexity:
  assumes
    st: ‹full enc-weight-opt.cdcl-bnb-stgy (init-state (penc N)) T› and
    dist: ‹distinct-mset-mset N› and
    atms: ‹atms-of-mm N = Σ›
  shows ‹size (learned-clss T) ≤ 2 ^(card (atms-of-mm N − ΔΣ)) ∗ 4^(card ΔΣ)›
```

proof −
  have [*simp*]: ‹*finite* Σ›
    unfolding *atms*[*symmetric*]
    by *auto*
  have [*simp*]: ‹*card* (*atms-of-mm* *N* − ΔΣ ∪ *replacement-pos* ‘ ΔΣ ∪ *replacement-neg* ‘ ΔΣ) =
    card (*atms-of-mm* *N* − ΔΣ) + *card* (*replacement-pos* ‘ ΔΣ) + *card* (*replacement-neg* ‘ ΔΣ)›
    by (*subst card-Un-disjoint*; *auto simp*: *atms*)+
  have [*simp*]: ‹*card* (*replacement-pos* ‘ ΔΣ) = *card* ΔΣ› ‹*card* (*replacement-neg* ‘ ΔΣ) = *card* ΔΣ›
    by (*auto intro*!: *card-image simp*: *inj-on-def*)

  show *?thesis*
    apply (*rule order-trans*[*OF enc-weight-opt.cdcl-bnb-pow2-n-learned-clauses*[*of* ‹*penc N*›]])
   using *assms* ΔΣ-Σ *monoid-mult-class.power-mult*[*of* ‹*2* :: *nat*› ‹*2* :: *nat*› ‹*card* ΔΣ›, *unfolded mult-2*]
    by (*auto simp*: *full-def distinct-mset-penc monoid-mult-class.power-add*
      enc-weight-opt.rtranclp-cdcl-bnb-stgy-cdcl-bnb atms-of-mm-penc-subset2)
qed

inductive $ocdcl_W$*-o-r* :: ‹′*st* ⇒ ′*st* ⇒ *bool*› **for** *S* :: ′*st* **where**
  decide: ‹*odecide S S*′ ⟹ $ocdcl_W$*-o-r S S*′› |
  bj: ‹*enc-weight-opt.cdcl-bnb-bj S S*′ ⟹ $ocdcl_W$*-o-r S S*′›

inductive *cdcl-bnb-r* :: ‹′*st* ⇒ ′*st* ⇒ *bool*› **for** *S* :: ′*st* **where**
  cdcl-conflict: ‹*conflict S S*′ ⟹ *cdcl-bnb-r S S*′› |
  cdcl-propagate: ‹*propagate S S*′ ⟹ *cdcl-bnb-r S S*′› |
  cdcl-improve: ‹*enc-weight-opt.improvep S S*′ ⟹ *cdcl-bnb-r S S*′› |
  cdcl-conflict-opt: ‹*enc-weight-opt.conflict-opt S S*′ ⟹ *cdcl-bnb-r S S*′› |
  cdcl-o′: ‹$ocdcl_W$*-o-r S S*′ ⟹ *cdcl-bnb-r S S*′›

inductive *cdcl-bnb-r-stgy* :: ‹′*st* ⇒ ′*st* ⇒ *bool*› **for** *S* :: ′*st* **where**
  cdcl-bnb-r-conflict: ‹*conflict S S*′ ⟹ *cdcl-bnb-r-stgy S S*′› |
  cdcl-bnb-r-propagate: ‹*propagate S S*′ ⟹ *cdcl-bnb-r-stgy S S*′› |
  cdcl-bnb-r-improve: ‹*enc-weight-opt.improvep S S*′ ⟹ *cdcl-bnb-r-stgy S S*′› |
  cdcl-bnb-r-conflict-opt: ‹*enc-weight-opt.conflict-opt S S*′ ⟹ *cdcl-bnb-r-stgy S S*′› |
  cdcl-bnb-r-other′: ‹$ocdcl_W$*-o-r S S*′ ⟹ *no-confl-prop-impr S* ⟹ *cdcl-bnb-r-stgy S S*′›

lemma $ocdcl_W$*-o-r-cases*[*consumes 1*, *case-names odecode obacktrack skip resolve*]:
  assumes
    ‹$ocdcl_W$*-o-r S T*›
    ‹*odecide S T* ⟹ *P T*›
    ‹*enc-weight-opt.obacktrack S T* ⟹ *P T*›
    ‹*skip S T* ⟹ *P T*›
    ‹*resolve S T* ⟹ *P T*›
  shows ‹*P T*›
  using *assms* **by** (*auto simp*: $ocdcl_W$*-o-r.simps enc-weight-opt.cdcl-bnb-bj.simps*)

context
  fixes *S* :: ′*st*
  assumes *S*-Σ: ‹*atms-of-mm* (*init-clss S*) = (Σ − ΔΣ) ∪ *replacement-pos* ‘ ΔΣ
    ∪ *replacement-neg* ‘ ΔΣ›
begin

lemma *odecide-decide*:
  ‹*odecide S T* ⟹ *decide S T*›
  apply (*elim odecideE*)
  subgoal for *L*
    by (*rule decide.intros*[*of S* ‹*L*›]) *auto*

subgoal for *L*
by (*rule decide.intros*[*of S* ‹*Pos* ($L^{\mapsto 1}$)›]) (*use S*-Σ ΔΣ-Σ **in** *auto*)
subgoal for *L*
by (*rule decide.intros*[*of S* ‹*Pos* ($L^{\mapsto 0}$)›]) (*use S*-Σ ΔΣ-Σ **in** *auto*)
done

lemma *ocdcl*$_W$*-o-r-ocdcl*$_W$*-o*:
‹*ocdcl*$_W$*-o-r S T* ⟹ *enc-weight-opt.ocdcl*$_W$*-o S T*›
using *S*-Σ **by** (*auto simp*: *ocdcl*$_W$*-o-r.simps enc-weight-opt.ocdcl*$_W$*-o.simps*
dest: *odecide-decide*)

lemma *cdcl-bnb-r-cdcl-bnb*:
‹*cdcl-bnb-r S T* ⟹ *enc-weight-opt.cdcl-bnb S T*›
using *S*-Σ **by** (*auto simp*: *cdcl-bnb-r.simps enc-weight-opt.cdcl-bnb.simps*
dest: *ocdcl*$_W$*-o-r-ocdcl*$_W$*-o*)

lemma *cdcl-bnb-r-stgy-cdcl-bnb-stgy*:
‹*cdcl-bnb-r-stgy S T* ⟹ *enc-weight-opt.cdcl-bnb-stgy S T*›
using *S*-Σ **by** (*auto simp*: *cdcl-bnb-r-stgy.simps enc-weight-opt.cdcl-bnb-stgy.simps*
dest: *ocdcl*$_W$*-o-r-ocdcl*$_W$*-o*)

end

context
fixes *S* :: *'st*
assumes *S*-Σ: ‹*atms-of-mm* (*init-clss S*) = (Σ − ΔΣ) ∪ *replacement-pos* ‘ ΔΣ
∪ *replacement-neg* ‘ ΔΣ›
begin

lemma *rtranclp-cdcl-bnb-r-cdcl-bnb*:
‹*cdcl-bnb-r*** *S T* ⟹ *enc-weight-opt.cdcl-bnb*** *S T*›
apply (*induction rule*: *rtranclp-induct*)
subgoal by *auto*
subgoal for *T U*
using *S*-Σ *enc-weight-opt.rtranclp-cdcl-bnb-no-more-init-clss*[*of S T*]
by(*auto dest*: *cdcl-bnb-r-cdcl-bnb*)
done

lemma *rtranclp-cdcl-bnb-r-stgy-cdcl-bnb-stgy*:
‹*cdcl-bnb-r-stgy*** *S T* ⟹ *enc-weight-opt.cdcl-bnb-stgy*** *S T*›
apply (*induction rule*: *rtranclp-induct*)
subgoal by *auto*
subgoal for *T U*
using *S*-Σ
enc-weight-opt.rtranclp-cdcl-bnb-no-more-init-clss[*of S T*,
OF enc-weight-opt.rtranclp-cdcl-bnb-stgy-cdcl-bnb]
by (*auto dest*: *cdcl-bnb-r-stgy-cdcl-bnb-stgy*)
done

lemma *rtranclp-cdcl-bnb-r-all-struct-inv*:
‹*cdcl-bnb-r*** *S T* ⟹
cdcl$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*) ⟹
cdcl$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*enc-weight-opt.abs-state T*)›

using *rtranclp-cdcl-bnb-r-cdcl-bnb*[*of T*]
enc-weight-opt.rtranclp-cdcl-bnb-stgy-all-struct-inv **by** *blast*

lemma *rtranclp-cdcl-bnb-r-stgy-all-struct-inv*:
‹*cdcl-bnb-r-stgy*** *S T* ⟹
$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*) ⟹
$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state T*)›
using *rtranclp-cdcl-bnb-r-stgy-cdcl-bnb-stgy*[*of T*]
enc-weight-opt.rtranclp-cdcl-bnb-stgy-all-struct-inv[*of S T*]
enc-weight-opt.rtranclp-cdcl-bnb-stgy-cdcl-bnb[*of S T*]
by *auto*

end

lemma *no-step-cdcl-bnb-r-stgy-no-step-cdcl-bnb-stgy*:
assumes
N: ‹*init-clss S* = *penc N*› **and**
Σ: ‹*atms-of-mm N* = Σ› **and**
n-d: ‹*no-dup* (*trail S*)› **and**
tr-alien: ‹*atm-of* ‘ *lits-of-l* (*trail S*) ⊆ Σ ∪ *replacement-pos* ‘ ΔΣ ∪ *replacement-neg* ‘ ΔΣ›
shows
‹*no-step cdcl-bnb-r-stgy S* ⟷ *no-step enc-weight-opt.cdcl-bnb-stgy S*› (**is** ‹*?A* ⟷ *?B*›)
proof
assume *?B*
then show ‹*?A*›
using *N cdcl-bnb-r-stgy-cdcl-bnb-stgy*[*of S*] *atms-of-mm-encode-clause-subset*[*of N*]
atms-of-mm-encode-clause-subset2[*of N*] *finite-*Σ ΔΣ-Σ
atms-of-mm-penc-subset2[*of N*]
by (*auto simp*: Σ)
next
assume *?A*
then have
nsd: ‹*no-step odecide S*› **and**
nsp: ‹*no-step propagate S*› **and**
nsc: ‹*no-step conflict S*› **and**
nsi: ‹*no-step enc-weight-opt.improvep S*› **and**
nsco: ‹*no-step enc-weight-opt.conflict-opt S*›
by (*auto simp*: *cdcl-bnb-r-stgy.simps* $ocdcl_W$*-o-r.simps*)
have
nsi': ‹⋀*M'*. *conflicting S* = *None* ⟹ ¬*enc-weight-opt.is-improving* (*trail S*) *M' S*› **and**
nsco': ‹*conflicting S* = *None* ⟹ *negate-ann-lits* (*trail S*) ∉# *enc-weight-opt.conflicting-clss S*›
using *nsi nsco* **unfolding** *enc-weight-opt.improvep.simps enc-weight-opt.conflict-opt.simps*
by *auto*
have *N-*Σ: ‹*atms-of-mm* (*penc N*) =
(Σ − ΔΣ) ∪ *replacement-pos* ‘ ΔΣ ∪ *replacement-neg* ‘ ΔΣ›
using *N* Σ *cdcl-bnb-r-stgy-cdcl-bnb-stgy*[*of S*] *atms-of-mm-encode-clause-subset*[*of N*]
atms-of-mm-encode-clause-subset2[*of N*] *finite-*Σ ΔΣ-Σ
atms-of-mm-penc-subset2[*of N*]
by *auto*
have *False* **if** *dec*: ‹*decide S T*› **for** *T*
proof −
obtain *L* **where**
[*simp*]: ‹*conflicting S* = *None*› **and**
undef: ‹*undefined-lit* (*trail S*) *L*› **and**
L: ‹*atm-of L* ∈ *atms-of-mm* (*init-clss S*)› **and**
T: ‹*T* ∼ *cons-trail* (*Decided L*) *S*›

```
      using dec unfolding decide.simps
      by auto
    have 1: ‹atm-of L ∉ Σ − ΔΣ›
      using nsd L undef by (fastforce simp: odecide.simps N Σ)
    have 2: False if L: ‹atm-of L ∈ replacement-pos ‘ ΔΣ ∪
        replacement-neg ‘ ΔΣ›
    proof −
      obtain A where
        ‹A ∈ ΔΣ› and
        ‹atm-of L = replacement-pos A ∨ atm-of L = replacement-neg A› and
        ‹A ∈ Σ›
        using L ΔΣ-Σ by auto
      then show False
        using nsd L undef T N-Σ
        using odecide.intros(2−)[of S ‹A›]
        unfolding N Σ
        by (cases L) (auto 6 5 simp: defined-lit-Neg-Pos-iff Σ)
    qed
    have defined-replacement-pos: ‹defined-lit (trail S) (Pos (replacement-pos L))›
      if ‹L ∈ ΔΣ› for L
      using nsd that ΔΣ-Σ odecide.intros(2−)[of S ‹L›] by (auto simp: N Σ N-Σ)
    have defined-all: ‹defined-lit (trail S) L›
      if ‹atm-of L ∈ Σ − ΔΣ› for L
      using nsd that ΔΣ-Σ odecide.intros(1)[of S ‹L›] by (force simp: N Σ N-Σ)
    have defined-replacement-neg: ‹defined-lit (trail S) (Pos (replacement-neg L))›
      if ‹L ∈ ΔΣ› for L
      using nsd that ΔΣ-Σ odecide.intros(2−)[of S ‹L›] by (force simp: N Σ N-Σ)
    have [simp]: ‹{A ∈ ΔΣ. A ∈ Σ} = ΔΣ›
      using ΔΣ-Σ by auto
    have atms-tr': ‹Σ − ΔΣ ∪ replacement-pos ‘ ΔΣ ∪ replacement-neg ‘ ΔΣ ⊆
        atm-of ‘ (lits-of-l (trail S))›
      using N Σ cdcl-bnb-r-stgy-cdcl-bnb-stgy[of S] atms-of-mm-encode-clause-subset[of N]
        atms-of-mm-encode-clause-subset2[of N] finite-Σ ΔΣ-Σ
        defined-replacement-pos defined-replacement-neg defined-all
      unfolding N Σ N-Σ
      apply (auto simp: Decided-Propagated-in-iff-in-lits-of-l)
        apply (metis image-eqI literal.sel(1) literal.sel(2) uminus-Pos)
       apply (metis image-eqI literal.sel(1) literal.sel(2))
      apply (metis image-eqI literal.sel(1) literal.sel(2))
      done
    then have atms-tr: ‹atms-of-mm (encode-clauses N) ⊆ atm-of ‘ (lits-of-l (trail S))›
      using N atms-of-mm-encode-clause-subset[of N]
        atms-of-mm-encode-clause-subset2[of N, OF finite-Σ] ΔΣ-Σ
      unfolding N Σ N-Σ ‹{A ∈ ΔΣ. A ∈ Σ} = ΔΣ›
      by (meson order-trans)
    show False
      by (metis L N N-Σ atm-lit-of-set-lits-of-l
        atms-tr' defined-lit-map subsetCE undef)
  qed
  then show ?B
    using ‹?A›
    by (auto simp: cdcl-bnb-r-stgy.simps enc-weight-opt.cdcl-bnb-stgy.simps
        ocdcl_W-o-r.simps enc-weight-opt.ocdcl_W-o.simps)
qed

lemma cdcl-bnb-r-stgy-init-clss:
```

‹cdcl-bnb-r-stgy S T ⟹ init-clss S = init-clss T›
by (*auto simp*: *cdcl-bnb-r-stgy.simps* $ocdcl_W$*-o-r.simps* *enc-weight-opt.cdcl-bnb-bj.simps*
elim: *conflictE propagateE enc-weight-opt.improveE enc-weight-opt.conflict-optE*
odecideE skipE resolveE enc-weight-opt.obacktrackE)

lemma *rtranclp-cdcl-bnb-r-stgy-init-clss*:
‹$cdcl\text{-}bnb\text{-}r\text{-}stgy^{**}$ S T ⟹ init-clss S = init-clss T›
by (*induction rule*: *rtranclp-induct*)(*auto simp*: *dest*: *cdcl-bnb-r-stgy-init-clss*)

lemma [*simp*]:
‹*enc-weight-opt.abs-state* (*init-state N*) = *abs-state* (*init-state N*)›
by (*auto simp*: *enc-weight-opt.abs-state-def abs-state-def*)

corollary
assumes
Σ: ‹*atms-of-mm N* = Σ› **and** *dist*: ‹*distinct-mset-mset N*› **and**
‹*full cdcl-bnb-r-stgy* (*init-state* (*penc N*)) *T*›
shows
‹*full enc-weight-opt.cdcl-bnb-stgy* (*init-state* (*penc N*)) *T*›
proof −
have [*simp*]: ‹*atms-of-mm* (*CDCL-W-Abstract-State.init-clss* (*enc-weight-opt.abs-state T*)) =
atms-of-mm (*init-clss T*)›
by (*auto simp*: *enc-weight-opt.abs-state-def init-clss.simps*)
let *?S* = ‹*init-state* (*penc N*)›
have
st: ‹$cdcl\text{-}bnb\text{-}r\text{-}stgy^{**}$ *?S T*› **and**
ns: ‹*no-step cdcl-bnb-r-stgy T*›
using *assms* **unfolding** *full-def* **by** *metis+*
have *st'*: ‹$enc\text{-}weight\text{-}opt.cdcl\text{-}bnb\text{-}stgy^{**}$ *?S T*›
by (*rule rtranclp-cdcl-bnb-r-stgy-cdcl-bnb-stgy*[*OF - st*])
(*use atms-of-mm-penc-subset2*[*of N*] *finite*-Σ ΔΣ-Σ Σ **in** *auto*)
have [*simp*]:
‹*CDCL-W-Abstract-State.init-clss* (*abs-state* (*init-state* (*penc N*))) =
(*penc N*)›
by (*auto simp*: *abs-state-def init-clss.simps*)
have [*iff*]: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state ?S*)›
using *dist distinct-mset-penc*[*of N*]
by (*auto simp*: $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*
$cdcl_W$*-restart-mset.distinct-*$cdcl_W$*-state-def* Σ
$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause-alt-def*)
have ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state T*)›
using *enc-weight-opt.rtranclp-cdcl-bnb-stgy-all-struct-inv*[*of ?S T*]
enc-weight-opt.rtranclp-cdcl-bnb-stgy-cdcl-bnb[*OF st'*]
by *auto*
then have *alien*: ‹$cdcl_W$*-restart-mset.no-strange-atm* (*enc-weight-opt.abs-state T*)› **and**
lev: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv* (*enc-weight-opt.abs-state T*)›
unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*
by *fast+*
have [*simp*]: ‹*init-clss T* = *penc N*›
using *rtranclp-cdcl-bnb-r-stgy-init-clss*[*OF st*] **by** *auto*

have ‹*no-step enc-weight-opt.cdcl-bnb-stgy T*›
by (*rule no-step-cdcl-bnb-r-stgy-no-step-cdcl-bnb-stgy*[*THEN iffD1*, *of - N*, *OF - - - - ns*])
(*use alien atms-of-mm-penc-subset2*[*of N*] *finite*-Σ ΔΣ-Σ *lev*
in ‹*auto simp*: $cdcl_W$*-restart-mset.no-strange-atm-def* Σ
$cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv-def*›)

then show ‹*full enc-weight-opt.cdcl-bnb-stgy* (*init-state* (*penc N*)) *T*›
using *st'* **unfolding** *full-def*
by *auto*
qed

lemma *propagation-one-lit-of-same-lvl*:
assumes
‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state S*)› **and**
‹*no-smaller-propa S*› **and**
‹*Propagated L E* ∈ *set* (*trail S*)› **and**
rea: ‹*reasons-in-clauses S*› **and**
nempty: ‹*E* − {#*L*#} ≠ {#}›
shows
‹∃ *L'* ∈# *E* − {#*L*#}. *get-level* (*trail S*) *L* = *get-level* (*trail S*) *L'*›
proof (*rule ccontr*)
assume *H*: ‹¬*?thesis*›
have *ns*: ‹⋀*M K M' D L*.
trail S = *M'* @ *Decided K* # *M* ⟹
D + {#*L*#} ∈# *clauses S* ⟹ *undefined-lit M L* ⟹ ¬ *M* ⊨*as CNot D*› **and**
n-d: ‹*no-dup* (*trail S*)›
using *assms* **unfolding** *no-smaller-propa-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv-def*
by *auto*
obtain *M1 M2* **where** *M2*: ‹*trail S* = *M2* @ *Propagated L E* # *M1*›
using *assms* **by** (*auto dest*!: *split-list*)

have ‹⋀*L mark a b*.
a @ *Propagated L mark* # *b* = *trail S* ⟹
b ⊨*as CNot* (*remove1-mset L mark*) ∧ *L* ∈# *mark*› **and**
‹*set* (*get-all-mark-of-propagated* (*trail S*)) ⊆ *set-mset* (*clauses S*)›
using *assms* **unfolding** $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-conflicting-def*
reasons-in-clauses-def
by *auto*
from *this*(*1*)[*OF M2*[*symmetric*]] *this*(*2*)
have ‹*M1* ⊨*as CNot* (*remove1-mset L E*)› **and** ‹*L* ∈# *E*› **and** ‹*E* ∈# *clauses S*›
by (*auto simp*: *M2*)
then have *lev-le*:
‹*L'* ∈# *E* − {#*L*#} ⟹ *get-level* (*trail S*) *L* > *get-level* (*trail S*) *L'*› **and**
‹*trail S* ⊨*as CNot* (*remove1-mset L E*)› **for** *L'*
using *H n-d defined-lit-no-dupD*(*1*)[*of M1 - M2*]
count-decided-ge-get-level[*of M1 L'*]
by (*auto simp*: *M2 get-level-append-if get-level-cons-if*
Decided-Propagated-in-iff-in-lits-of-l atm-of-eq-atm-of
true-annots-append-l
dest!: *multi-member-split*)
define *i* **where** ‹*i* = *get-level* (*trail S*) *L* − *1*›
have ‹*i* < *local.backtrack-lvl S*› **and** ‹*get-level* (*trail S*) *L* ≥ *1*›
‹*get-level* (*trail S*) *L* > *i*› **and**
i2: ‹*get-level* (*trail S*) *L* = *Suc i*›
using *lev-le nempty count-decided-ge-get-level*[*of* ‹*trail S*› *L*] *i-def*
by (*cases* ‹*E* − {#*L*#}›; *force*)+
from *backtrack-ex-decomp*[*OF n-d this*(*1*)] **obtain** *M3 M4 K* **where**
decomp: ‹(*Decided K* # *M3*, *M4*) ∈ *set* (*get-all-ann-decomposition* (*trail S*))› **and**
lev-K: ‹*get-level* (*trail S*) *K* = *Suc i*›

by *blast*
then obtain *M5* **where**
tr: ‹*trail S* = (*M5* @ *M4*) @ *Decided K* # *M3*›
by *auto*
define *M4'* **where** ‹*M4'* = *M5* @ *M4*›
have ‹*undefined-lit M3 L*›
using *n-d* ‹*get-level* (*trail S*) *L* > *i*› *lev-K*
count-decided-ge-get-level[*of M3 L*] **unfolding** *tr M4'-def*[*symmetric*]
by (*auto simp*: *get-level-append-if get-level-cons-if*
atm-of-eq-atm-of
split: *if-splits dest*: *defined-lit-no-dupD*)
moreover have ‹*M3* ⊨*as CNot* (*remove1-mset L E*)›
using ‹*trail S* ⊨*as CNot* (*remove1-mset L E*)› *lev-K n-d*
unfolding *true-annots-def true-annot-def*
apply *clarsimp*
subgoal for *L'*
using *lev-le*[*of* ‹−*L'*›] *lev-le*[*of* ‹*L'*›] *lev-K*
unfolding *i2*
unfolding *tr M4'-def*[*symmetric*]
by (*auto simp*: *get-level-append-if get-level-cons-if*
atm-of-eq-atm-of if-distrib if-distribR Decided-Propagated-in-iff-in-lits-of-l
split: *if-splits dest*: *defined-lit-no-dupD*
dest!: *multi-member-split*)
done
ultimately show *False*
using *ns*[*OF tr*, *of* ‹*remove1-mset L E*› *L*] ‹*E* ∈# *clauses S*› ‹*L* ∈# *E*›
by *auto*
qed

lemma *simple-backtrack-obacktrack*:
‹*simple-backtrack S T* ⟹ $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*) ⟹
enc-weight-opt.obacktrack S T›
unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-conflicting-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause-alt-def*
apply (*auto simp*: *simple-backtrack.simps*
enc-weight-opt.obacktrack.simps)
apply (*rule-tac x=L* **in** *exI*)
apply (*rule-tac x=D* **in** *exI*)
apply *auto*
apply (*rule-tac x=K* **in** *exI*)
apply (*rule-tac x=M1* **in** *exI*)
apply *auto*
apply (*rule-tac x=D* **in** *exI*)
apply (*auto simp*:)
done

end

interpretation *test-real*: *optimal-encoding-opt* **where**
state-eq = ‹(=)› **and**
state = *id* **and**
trail = ‹λ(*M*, *N*, *U*, *D*, *W*). *M*› **and**
init-clss = ‹λ(*M*, *N*, *U*, *D*, *W*). *N*› **and**
learned-clss = ‹λ(*M*, *N*, *U*, *D*, *W*). *U*› **and**

```
  conflicting = ‹λ(M, N, U, D, W). D› and
  cons-trail = ‹λK (M, N, U, D, W). (K # M, N, U, D, W)› and
  tl-trail = ‹λ(M, N, U, D, W). (tl M, N, U, D, W)› and
  add-learned-cls = ‹λC (M, N, U, D, W). (M, N, add-mset C U, D, W)› and
  remove-cls = ‹λC (M, N, U, D, W). (M, removeAll-mset C N, removeAll-mset C U, D, W)› and
  update-conflicting = ‹λC (M, N, U, -, W). (M, N, U, C, W)› and
  init-state = ‹λN. ([], N, {#}, None, None, ())› and
  ϱ = ‹λ-. (0::real)› and
  update-additional-info = ‹λW (M, N, U, D, -, -). (M, N, U, D, W)› and
  Σ = ‹{1..(100::nat)}› and
  ΔΣ = ‹{1..(50::nat)}› and
  new-vars = ‹λn. (200 + 2*n, 200 + 2*n+1)›
  by unfold-locales

lemma mult3-inj:
  ‹2 * A = Suc (2 * Aa) ⟷ False› for A Aa::nat
  by presburger+

interpretation test-real: optimal-encoding where
  state-eq = ‹(=)› and
  state = id and
  trail = ‹λ(M, N, U, D, W). M› and
  init-clss = ‹λ(M, N, U, D, W). N› and
  learned-clss = ‹λ(M, N, U, D, W). U› and
  conflicting = ‹λ(M, N, U, D, W). D› and
  cons-trail = ‹λK (M, N, U, D, W). (K # M, N, U, D, W)› and
  tl-trail = ‹λ(M, N, U, D, W). (tl M, N, U, D, W)› and
  add-learned-cls = ‹λC (M, N, U, D, W). (M, N, add-mset C U, D, W)› and
  remove-cls = ‹λC (M, N, U, D, W). (M, removeAll-mset C N, removeAll-mset C U, D, W)› and
  update-conflicting = ‹λC (M, N, U, -, W). (M, N, U, C, W)› and
  init-state = ‹λN. ([], N, {#}, None, None, ())› and
  ϱ = ‹λ-. (0::real)› and
  update-additional-info = ‹λW (M, N, U, D, -, -). (M, N, U, D, W)› and
  Σ = ‹{1..(100::nat)}› and
  ΔΣ = ‹{1..(50::nat)}› and
  new-vars = ‹λn. (200 + 2*n, 200 + 2*n+1)›
  by unfold-locales (auto simp: inj-on-def mult3-inj)

interpretation test-nat: optimal-encoding-opt where
  state-eq = ‹(=)› and
  state = id and
  trail = ‹λ(M, N, U, D, W). M› and
  init-clss = ‹λ(M, N, U, D, W). N› and
  learned-clss = ‹λ(M, N, U, D, W). U› and
  conflicting = ‹λ(M, N, U, D, W). D› and
  cons-trail = ‹λK (M, N, U, D, W). (K # M, N, U, D, W)› and
  tl-trail = ‹λ(M, N, U, D, W). (tl M, N, U, D, W)› and
  add-learned-cls = ‹λC (M, N, U, D, W). (M, N, add-mset C U, D, W)› and
  remove-cls = ‹λC (M, N, U, D, W). (M, removeAll-mset C N, removeAll-mset C U, D, W)› and
  update-conflicting = ‹λC (M, N, U, -, W). (M, N, U, C, W)› and
  init-state = ‹λN. ([], N, {#}, None, None, ())› and
  ϱ = ‹λ-. (0::nat)› and
  update-additional-info = ‹λW (M, N, U, D, -, -). (M, N, U, D, W)› and
  Σ = ‹{1..(100::nat)}› and
  ΔΣ = ‹{1..(50::nat)}› and
  new-vars = ‹λn. (200 + 2*n, 200 + 2*n+1)›
```

by *unfold-locales*

interpretation *test-nat*: *optimal-encoding* **where**
state-eq = ‹(=)› **and**
state = *id* **and**
trail = ‹λ(*M*, *N*, *U*, *D*, *W*). *M*› **and**
init-clss = ‹λ(*M*, *N*, *U*, *D*, *W*). *N*› **and**
learned-clss = ‹λ(*M*, *N*, *U*, *D*, *W*). *U*› **and**
conflicting = ‹λ(*M*, *N*, *U*, *D*, *W*). *D*› **and**
cons-trail = ‹λ*K* (*M*, *N*, *U*, *D*, *W*). (*K* # *M*, *N*, *U*, *D*, *W*)› **and**
tl-trail = ‹λ(*M*, *N*, *U*, *D*, *W*). (*tl M*, *N*, *U*, *D*, *W*)› **and**
add-learned-cls = ‹λ*C* (*M*, *N*, *U*, *D*, *W*). (*M*, *N*, *add-mset C U*, *D*, *W*)› **and**
remove-cls = ‹λ*C* (*M*, *N*, *U*, *D*, *W*). (*M*, *removeAll-mset C N*, *removeAll-mset C U*, *D*, *W*)› **and**
update-conflicting = ‹λ*C* (*M*, *N*, *U*, -, *W*). (*M*, *N*, *U*, *C*, *W*)› **and**
init-state = ‹λ*N*. ([], *N*, {#}, *None*, *None*, ())› **and**
ϱ = ‹λ-. (*0*::*nat*)› **and**
update-additional-info = ‹λ*W* (*M*, *N*, *U*, *D*, -, -). (*M*, *N*, *U*, *D*, *W*)› **and**
Σ = ‹{*1*..(*100*::*nat*)}› **and**
ΔΣ = ‹{*1*..(*50*::*nat*)}› **and**
new-vars = ‹λ*n*. (*200* + *2*∗*n*, *200* + *2*∗*n*+*1*)›
by *unfold-locales* (*auto simp*: *inj-on-def mult3-inj*)

end
theory *CDCL-W-MaxSAT*
imports *CDCL-W-Optimal-Model*
begin

0.1.3 Partial MAX-SAT

definition *weight-on-clauses* **where**
‹*weight-on-clauses* N_S ϱ *I* = (∑ *C* ∈# (*filter-mset* (λ*C*. *I* ⊨ *C*) N_S). ϱ *C*)›

definition *atms-exactly-m* :: ‹*'v partial-interp* ⇒ *'v clauses* ⇒ *bool*› **where**
‹*atms-exactly-m I N* ⟷
total-over-m I (*set-mset N*) ∧
atms-of-s I ⊆ *atms-of-mm N*›

Partial in the name refers to the fact that not all clauses are soft clauses, not to the fact that we consider partial models.

inductive *partial-max-sat* :: ‹*'v clauses* ⇒ *'v clauses* ⇒ (*'v clause* ⇒ *nat*) ⇒
'v partial-interp option ⇒ *bool*› **where**
partial-max-sat:
‹*partial-max-sat* N_H N_S ϱ (*Some I*)›
if
‹*I* ⊨*sm* N_H› **and**
‹*atms-exactly-m I* ((N_H + N_S))› **and**
‹*consistent-interp I*› **and**
‹⋀*I'*. *consistent-interp I'* ⟹ *atms-exactly-m I'* (N_H + N_S) ⟹ *I'* ⊨*sm* N_H ⟹
weight-on-clauses N_S ϱ *I'* ≤ *weight-on-clauses* N_S ϱ *I*› |
partial-max-unsat:
‹*partial-max-sat* N_H N_S ϱ *None*›
if
‹*unsatisfiable* (*set-mset* N_H)›

inductive *partial-min-sat* :: ‹*'v clauses* ⇒ *'v clauses* ⇒ (*'v clause* ⇒ *nat*) ⇒

′v partial-interp option ⇒ bool› **where**
partial-min-sat:
‹*partial-min-sat* N_H N_S ϱ (*Some I*)›
if
‹*I* ⊨*sm* N_H› **and**
‹*atms-exactly-m I* (N_H + N_S)› **and**
‹*consistent-interp I*› **and**
‹⋀*I′*. *consistent-interp I′* ⟹ *atms-exactly-m I′* (N_H + N_S) ⟹ *I′* ⊨*sm* N_H ⟹
weight-on-clauses N_S ϱ *I′* ≥ *weight-on-clauses* N_S ϱ *I*› |
partial-min-unsat:
‹*partial-min-sat* N_H N_S ϱ *None*›
if
‹*unsatisfiable* (*set-mset* N_H)›

lemma *atms-exactly-m-finite*:
assumes ‹*atms-exactly-m I N*›
shows ‹*finite I*›
proof −
have ‹*I* ⊆ *Pos* ‘ (*atms-of-mm N*) ∪ *Neg* ‘ *atms-of-mm N*›
using *assms* **by** (*force simp*: *total-over-m-def atms-exactly-m-def lit-in-set-iff-atm*
atms-of-s-def)
from *finite-subset*[*OF this*] **show** *?thesis* **by** *auto*
qed

lemma
fixes N_H :: ‹*′v clauses*›
assumes ‹*satisfiable* (*set-mset* N_H)›
shows *sat-partial-max-sat*: ‹∃ *I*. *partial-max-sat* N_H N_S ϱ (*Some I*)› **and**
sat-partial-min-sat: ‹∃ *I*. *partial-min-sat* N_H N_S ϱ (*Some I*)›
proof −
let *?Is* = ‹{*I*. *atms-exactly-m I* ((N_H + N_S)) ∧ *consistent-interp I* ∧
I ⊨*sm* N_H}›
let *?Is′*= ‹{*I*. *atms-exactly-m I* ((N_H + N_S)) ∧ *consistent-interp I* ∧
I ⊨*sm* N_H ∧ *finite I*}›
have *Is*: ‹*?Is* = *?Is′*›
by (*auto simp*: *atms-of-s-def atms-exactly-m-finite*)
have ‹*?Is′* ⊆ *set-mset* ‘ *simple-clss* (*atms-of-mm* (N_H + N_S))›
apply *rule*
unfolding *image-iff*
by (*rule-tac x*= ‹*mset-set x*› **in** *bexI*)
(*auto simp*: *simple-clss-def atms-exactly-m-def image-iff*
atms-of-s-def atms-of-def distinct-mset-mset-set consistent-interp-tuatology-mset-set)
from *finite-subset*[*OF this*] **have** *fin*: ‹*finite ?Is*› **unfolding** *Is*
by (*auto simp*: *simple-clss-finite*)
then have *fin′*: ‹*finite* (*weight-on-clauses* N_S ϱ ‘ *?Is*)›
by *auto*
define ϱ*I* **where**
‹ϱ*I* = *Min* (*weight-on-clauses* N_S ϱ ‘ *?Is*)›
have *nempty*: ‹*?Is* ≠ {}›
proof −
obtain *I* **where** *I*:
‹*total-over-m I* (*set-mset* N_H)›
‹*I* ⊨*sm* N_H›
‹*consistent-interp I*›
‹*atms-of-s I* ⊆ *atms-of-mm* N_H›

using *assms* **unfolding** *satisfiable-def-min atms-exactly-m-def*
by (*auto simp*: *atms-of-s-def atm-of-def total-over-m-def*)
let *?I* = ‹$I \cup$ *Pos* ‘ {$x \in$ *atms-of-mm* N_S. $x \notin$ *atm-of* ‘ I}›
have ‹*?I* $\in$ *?Is*›
using *I*
by (*auto simp*: *atms-exactly-m-def total-over-m-alt-def image-iff lit-in-set-iff-atm*)
(*auto simp*: *consistent-interp-def uminus-lit-swap*)
then show *?thesis*
by *blast*
qed
have ‹$\varrho I \in$ *weight-on-clauses* N_S ϱ ‘ *?Is*›
unfolding *ϱI-def*
by (*rule Min-in*[*OF fin'*]) (*use nempty* **in** *auto*)
then obtain I :: ‹*'v partial-interp*› **where**
‹*weight-on-clauses* N_S ϱ $I = \varrho I$› **and**
‹$I \in$ *?Is*›
by *blast*
then have *H*: ‹*consistent-interp* $I' \Longrightarrow$ *atms-exactly-m* I' $(N_H + N_S) \Longrightarrow I' \models$*sm* $N_H \Longrightarrow$
weight-on-clauses N_S ϱ $I' \geq$ *weight-on-clauses* N_S ϱ I› **for** I'
using *Min-le*[*OF fin'*, *of* ‹*weight-on-clauses* N_S ϱ I'›]
unfolding *ϱI-def*[*symmetric*]
by *auto*
then have ‹*partial-min-sat* N_H N_S ϱ (*Some* I)›
apply −
by (*rule partial-min-sat*)
(*use fin* ‹$I \in$ *?Is*› **in** ‹*auto simp*: *atms-exactly-m-finite*›)
then show ‹$\exists I$. *partial-min-sat* N_H N_S ϱ (*Some* I)›
by *fast*

define *ϱI* **where**
‹$\varrho I =$ *Max* (*weight-on-clauses* N_S ϱ ‘ *?Is*)›
have ‹$\varrho I \in$ *weight-on-clauses* N_S ϱ ‘ *?Is*›
unfolding *ϱI-def*
by (*rule Max-in*[*OF fin'*]) (*use nempty* **in** *auto*)
then obtain I :: ‹*'v partial-interp*› **where**
‹*weight-on-clauses* N_S ϱ $I = \varrho I$› **and**
‹$I \in$ *?Is*›
by *blast*
then have *H*: ‹*consistent-interp* $I' \Longrightarrow$ *atms-exactly-m* I' $(N_H + N_S) \Longrightarrow I' \models$*m* $N_H \Longrightarrow$
weight-on-clauses N_S ϱ $I' \leq$ *weight-on-clauses* N_S ϱ I› **for** I'
using *Max-ge*[*OF fin'*, *of* ‹*weight-on-clauses* N_S ϱ I'›]
unfolding *ϱI-def*[*symmetric*]
by *auto*
then have ‹*partial-max-sat* N_H N_S ϱ (*Some* I)›
apply −
by (*rule partial-max-sat*)
(*use fin* ‹$I \in$ *?Is*› **in** ‹*auto simp*: *atms-exactly-m-finite consistent-interp-tuatology-mset-set*›)
then show ‹$\exists I$. *partial-max-sat* N_H N_S ϱ (*Some* I)›
by *fast*
qed

inductive *weight-sat*
:: ‹*'v clauses* ⇒ (*'v literal multiset* ⇒ *'a* :: *linorder*) ⇒
'v literal multiset option ⇒ *bool*›

where
weight-sat:
‹*weight-sat N ϱ* (*Some I*)›
if
‹*set-mset I* ⊨*sm N*› **and**
‹*atms-exactly-m* (*set-mset I*) *N*› **and**
‹*consistent-interp* (*set-mset I*)› **and**
‹*distinct-mset I*›
‹⋀*I′*. *consistent-interp* (*set-mset I′*) ⟹ *atms-exactly-m* (*set-mset I′*) *N* ⟹ *distinct-mset I′* ⟹
set-mset I′ ⊨*sm N* ⟹ *ϱ I′* ≥ *ϱ I*› |
partial-max-unsat:
‹*weight-sat N ϱ None*›
if
‹*unsatisfiable* (*set-mset N*)›

lemma *partial-max-sat-is-weight-sat*:
fixes *additional-atm* :: ‹*′v clause* ⇒ *′v*› **and**
ϱ :: ‹*′v clause* ⇒ *nat*› **and**
N_S :: ‹*′v clauses*›
defines
‹*ϱ′* ≡ (λ*C*. *sum-mset*
((λ*L*. *if L* ∈ *Pos* ‘ *additional-atm* ‘ *set-mset* N_S
then count N_S (*SOME C*. *L* = *Pos* (*additional-atm C*) ∧ *C* ∈# N_S)
∗ *ϱ* (*SOME C*. *L* = *Pos* (*additional-atm C*) ∧ *C* ∈# N_S)
else 0) ‘# *C*))›
assumes
add: ‹⋀*C*. *C* ∈# N_S ⟹ *additional-atm C* ∉ *atms-of-mm* (N_H + N_S)›
‹⋀*C D*. *C* ∈# N_S ⟹ *D* ∈# N_S ⟹ *additional-atm C* = *additional-atm D* ⟷ *C* = *D*› **and**
w: ‹*weight-sat* (N_H + (λ*C*. *add-mset* (*Pos* (*additional-atm C*)) *C*) ‘# N_S) *ϱ′* (*Some I*)›
shows
‹*partial-max-sat* N_H N_S *ϱ* (*Some* {*L* ∈ *set-mset I*. *atm-of L* ∈ *atms-of-mm* (N_H + N_S)})›
proof −
define *N* **where** ‹*N* ≡ N_H + (λ*C*. *add-mset* (*Pos* (*additional-atm C*)) *C*) ‘# N_S›
define *cl-of* **where** ‹*cl-of L* = (*SOME C*. *L* = *Pos* (*additional-atm C*) ∧ *C* ∈# N_S)› **for** *L*
from *w*
have
ent: ‹*set-mset I* ⊨*sm N*› **and**
bi: ‹*atms-exactly-m* (*set-mset I*) *N*› **and**
cons: ‹*consistent-interp* (*set-mset I*)› **and**
dist: ‹*distinct-mset I*› **and**
weight: ‹⋀*I′*. *consistent-interp* (*set-mset I′*) ⟹ *atms-exactly-m* (*set-mset I′*) *N* ⟹
distinct-mset I′ ⟹ *set-mset I′* ⊨*sm N* ⟹ *ϱ′ I′* ≥ *ϱ′ I*›
unfolding *N-def*[*symmetric*]
by (*auto simp*: *weight-sat.simps*)
let *?I* = ‹{*L*. *L* ∈# *I* ∧ *atm-of L* ∈ *atms-of-mm* (N_H + N_S)}›
have *ent′*: ‹*set-mset I* ⊨*sm* N_H›
using *ent* **unfolding** *true-clss-restrict*
by (*auto simp*: *N-def*)
then have *ent′*: ‹*?I* ⊨*sm* N_H›
apply (*subst* (*asm*) *true-clss-restrict*[*symmetric*])
apply (*rule true-clss-mono-left*, *assumption*)
apply *auto*
done
have [*simp*]: ‹*atms-of-ms* ((λ*C*. *add-mset* (*Pos* (*additional-atm C*)) *C*) ‘ *set-mset* N_S) =
additional-atm ‘ *set-mset* N_S ∪ *atms-of-ms* (*set-mset* N_S)›
by (*auto simp*: *atms-of-ms-def*)

have *bi'*: ‹*atms-exactly-m ?I* $(N_H + N_S)$›
using *bi*
by (*auto simp*: *atms-exactly-m-def total-over-m-def total-over-set-def*
atms-of-s-def N-def)
have *cons'*: ‹*consistent-interp ?I*›
using *cons* **by** (*auto simp*: *consistent-interp-def*)
have [*simp*]: ‹*cl-of* (*Pos* (*additional-atm xb*)) = *xb*›
if ‹*xb* ∈# N_S› **for** *xb*
using *someI*[*of* ‹λ*C*. *additional-atm xb* = *additional-atm C*› *xb*] *add that*
unfolding *cl-of-def*
by *auto*

let *?I* = ‹{*L*. *L* ∈# *I* ∧ *atm-of L* ∈ *atms-of-mm* $(N_H + N_S)$} ∪ *Pos* ‘ *additional-atm* ‘ {*C* ∈ *set-mset* N_S. ¬*set-mset I* ⊨ *C*}
∪ *Neg* ‘ *additional-atm* ‘ {*C* ∈ *set-mset* N_S. *set-mset I* ⊨ *C*}›
have ‹*consistent-interp ?I*›
using *cons add* **by** (*auto simp*: *consistent-interp-def*
atms-exactly-m-def uminus-lit-swap
dest: *add*)
moreover have ‹*atms-exactly-m ?I N*›
using *bi*
by (*auto simp*: *N-def atms-exactly-m-def total-over-m-def*
total-over-set-def image-image)
moreover have ‹*?I* ⊨*sm N*›
using *ent* **by** (*auto simp*: *N-def true-clss-def image-image*
atm-of-lit-in-atms-of true-cls-def
dest!: *multi-member-split*)
moreover have ‹*set-mset* (*mset-set ?I*) = *?I*› **and** *fin*: ‹*finite ?I*›
by (*auto simp*: *atms-exactly-m-finite*)
moreover have ‹*distinct-mset* (*mset-set ?I*)›
by (*auto simp*: *distinct-mset-mset-set*)
ultimately have ‹ϱ' (*mset-set ?I*) ≥ ϱ' *I*›
using *weight*[*of* ‹*mset-set ?I*›]
by *argo*
moreover have ‹ϱ' (*mset-set ?I*) ≤ ϱ' *I*›
using *ent*
by (*auto simp*: *ϱ'-def sum-mset-inter-restrict*[*symmetric*] *mset-set-subset-iff N-def*
intro!: *sum-image-mset-mono*
dest!: *multi-member-split*)
ultimately have *I-I*: ‹ϱ' (*mset-set ?I*) = ϱ' *I*›
by *linarith*

have *min*: ‹*weight-on-clauses* N_S ϱ *I'*
≤ *weight-on-clauses* N_S ϱ {*L*. *L* ∈# *I* ∧ *atm-of L* ∈ *atms-of-mm* $(N_H + N_S)$}›
if
cons: ‹*consistent-interp I'*› **and**
bit: ‹*atms-exactly-m I'* $(N_H + N_S)$› **and**
I': ‹*I'* ⊨*sm* N_H›
for *I'*
proof −
let *?I'* = ‹*I'* ∪ *Pos* ‘ *additional-atm* ‘ {*C* ∈ *set-mset* N_S. ¬*I'* ⊨ *C*}
∪ *Neg* ‘ *additional-atm* ‘ {*C* ∈ *set-mset* N_S. *I'* ⊨ *C*}›
have ‹*consistent-interp ?I'*›
using *cons bit add* **by** (*auto simp*: *consistent-interp-def*
atms-exactly-m-def uminus-lit-swap
dest: *add*)

```
moreover have ‹atms-exactly-m ?I′ N›
  using bit
  by (auto simp: N-def atms-exactly-m-def total-over-m-def
      total-over-set-def image-image)
moreover have ‹?I′ ⊨sm N›
  using I′ by (auto simp: N-def true-clss-def image-image
      dest!: multi-member-split)
moreover have ‹set-mset (mset-set ?I′) = ?I′› and fin: ‹finite ?I′›
  using bit by (auto simp: atms-exactly-m-finite)
moreover have ‹distinct-mset (mset-set ?I′)›
  by (auto simp: distinct-mset-mset-set)
ultimately have I′-I: ‹ϱ′ (mset-set ?I′) ≥ ϱ′ I›
  using weight[of ‹mset-set ?I′›]
  by argo
have inj: ‹inj-on cl-of (I′ ∩ (λx. Pos (additional-atm x)) ‘ set-mset N_S)› for I′
  using add by (auto simp: inj-on-def)

have we: ‹weight-on-clauses N_S ϱ I′ = sum-mset (ϱ ‘# N_S) −
  sum-mset (ϱ ‘# filter-mset (Not ∘ (⊨) I′) N_S)› for I′
  unfolding weight-on-clauses-def
  apply (subst (3) multiset-partition[of - ‹(⊨) I′›])
  unfolding image-mset-union sum-mset.union
  by (auto simp: comp-def)
have H: ‹sum-mset
   (ϱ ‘#
    filter-mset (Not ∘ (⊨) {L. L ∈# I ∧ atm-of L ∈ atms-of-mm (N_H + N_S)})
     N_S) = ϱ′ I›
        unfolding I-I[symmetric] unfolding ϱ′-def cl-of-def[symmetric]
          sum-mset-sum-count if-distrib
        apply (auto simp: sum-mset-sum-count image-image simp flip: sum.inter-restrict
            cong: if-cong)
        apply (subst comm-monoid-add-class.sum.reindex-cong[symmetric, of cl-of, OF - refl])
        apply ((use inj in auto; fail)+)[2]
        apply (rule sum.cong)
        apply auto[]
        using inj[of ‹set-mset I›] ‹set-mset I ⊨sm N› assms(2)
        apply (auto dest!: multi-member-split simp: N-def image-Int
            atm-of-lit-in-atms-of true-cls-def)[]
        using add apply (auto simp: true-cls-def)
        done
have ‹(∑ x∈(I′ ∪ (λx. Pos (additional-atm x)) ‘ {C. C ∈# N_S ∧ ¬ I′ ⊨ C} ∪
     (λx. Neg (additional-atm x)) ‘ {C. C ∈# N_S ∧ I′ ⊨ C}) ∩
    (λx. Pos (additional-atm x)) ‘ set-mset N_S.
   count N_S (cl-of x) ∗ ϱ (cl-of x))
≤ (∑ A∈{a. a ∈# N_S ∧ ¬ I′ ⊨ a}. count N_S A ∗ ϱ A)›
  apply (subst comm-monoid-add-class.sum.reindex-cong[symmetric, of cl-of, OF - refl])
  apply ((use inj in auto; fail)+)[2]
  apply (rule ordered-comm-monoid-add-class.sum-mono2)
  using that add by (auto dest:  simp: N-def
      atms-exactly-m-def)
then have ‹sum-mset (ϱ ‘# filter-mset (Not ∘ (⊨) I′) N_S) ≥ ϱ′ (mset-set ?I′)›
  using fin unfolding cl-of-def[symmetric] ϱ′-def
  by (auto simp: ϱ′-def
      simp add: sum-mset-sum-count image-image simp flip: sum.inter-restrict)
then have ‹ϱ′ I ≤ sum-mset (ϱ ‘# filter-mset (Not ∘ (⊨) I′) N_S)›
  using I′-I by auto
```

```
    then show ?thesis
      unfolding we H I-I apply −
      by auto
  qed

  show ?thesis
    apply (rule partial-max-sat.intros)
    subgoal using ent′ by auto
    subgoal using bi′ by fast
    subgoal using cons′ by fast
    subgoal for I′
      by (rule min)
    done
qed

lemma sum-mset-cong:
  ‹(⋀a. a ∈# A ⟹ f a = g a) ⟹ (∑ a∈#A. f a) = (∑ a∈#A. g a)›
  by (induction A) auto

lemma partial-max-sat-is-weight-sat-distinct:
  fixes additional-atm :: ‹′v clause ⇒ ′v› and
    ϱ :: ‹′v clause ⇒ nat› and
    N_S :: ‹′v clauses›
  defines
    ‹ϱ′ ≡ (λC. sum-mset
       ((λL. if L ∈ Pos ‘ additional-atm ‘ set-mset N_S
         then ϱ (SOME C. L = Pos (additional-atm C) ∧ C ∈# N_S)
         else 0) ‘# C))›
  assumes
    ‹distinct-mset N_S› and — This is implicit on paper
    add: ‹⋀C. C ∈# N_S ⟹ additional-atm C ∉ atms-of-mm (N_H + N_S)›
    ‹⋀C D. C ∈# N_S ⟹ D ∈# N_S ⟹ additional-atm C = additional-atm D ⟷ C = D› and
    w: ‹weight-sat (N_H + (λC. add-mset (Pos (additional-atm C)) C) ‘# N_S) ϱ′ (Some I)›
  shows
    ‹partial-max-sat N_H N_S ϱ (Some {L ∈ set-mset I. atm-of L ∈ atms-of-mm (N_H + N_S)})›
proof −
  define cl-of where ‹cl-of L = (SOME C. L = Pos (additional-atm C) ∧ C ∈# N_S)› for L
  have [simp]: ‹cl-of (Pos (additional-atm xb)) = xb›
    if ‹xb ∈# N_S› for xb
    using someI[of ‹λC. additional-atm xb = additional-atm C› xb] add that
    unfolding cl-of-def
    by auto
  have ϱ′: ‹ϱ′ = (λC. ∑ L∈#C. if L ∈ Pos ‘ additional-atm ‘ set-mset N_S
               then count N_S
                     (SOME C. L = Pos (additional-atm C) ∧ C ∈# N_S) ∗
                    ϱ (SOME C. L = Pos (additional-atm C) ∧ C ∈# N_S)
               else 0)›
    unfolding cl-of-def[symmetric] ϱ′-def
   using assms(2,4) by (auto intro!: ext sum-mset-cong simp: ϱ′-def not-in-iff dest!: multi-member-split)
  show ?thesis
    apply (rule partial-max-sat-is-weight-sat[where additional-atm=additional-atm])
    subgoal by (rule assms(3))
    subgoal by (rule assms(4))
    subgoal unfolding ϱ′[symmetric] by (rule assms(5))
    done
qed
```

lemma *atms-exactly-m-alt-def*:
‹*atms-exactly-m* (*set-mset y*) *N* ⟷ *atms-of y* ⊆ *atms-of-mm N* ∧
total-over-m (*set-mset y*) (*set-mset N*)›
by (*auto simp*: *atms-exactly-m-def atms-of-s-def atms-of-def*
atms-of-ms-def dest!: *multi-member-split*)

lemma *atms-exactly-m-alt-def2*:
‹*atms-exactly-m* (*set-mset y*) *N* ⟷ *atms-of y* = *atms-of-mm N*›
by (*metis atms-of-def atms-of-s-def atms-exactly-m-alt-def equalityI order-refl total-over-m-def*
total-over-set-alt-def)

lemma (**in** *conflict-driven-clause-learning*$_W$*-optimal-weight*) *full-cdcl-bnb-stgy-weight-sat*:
‹*full cdcl-bnb-stgy* (*init-state N*) *T* ⟹ *distinct-mset-mset N* ⟹ *weight-sat N* ϱ (*weight T*)›
using *full-cdcl-bnb-stgy-no-conflicting-clause-from-init-state*[*of N T*]
apply (*cases* ‹*weight T* = *None*›)
subgoal
by (*auto intro*!: *weight-sat.intros*(*2*))
subgoal premises *p*
using *p*(*1*−*4*,*6*)
apply (*clarsimp simp only*:)
apply (*rule weight-sat.intros*(*1*))
subgoal by *auto*
subgoal by (*auto simp*: *atms-exactly-m-alt-def*)
subgoal by *auto*
subgoal by *auto*
subgoal for *J I′*
using *p*(*5*)[*of I′*] **by** (*auto simp*: *atms-exactly-m-alt-def2*)
done
done

end
theory *CDCL-W-Partial-Optimal-Model*
imports *CDCL-W-Partial-Encoding*
begin
lemma *isabelle-should-do-that-automatically*: ‹*Suc* (*a* − *Suc 0*) = *a* ⟷ *a* ≥ *1*›
by *auto*

lemma (**in** *conflict-driven-clause-learning*$_W$*-optimal-weight*)
conflict-opt-state-eq-compatible:
‹*conflict-opt S T* ⟹ *S* ∼ *S′* ⟹ *T* ∼ *T′* ⟹ *conflict-opt S′ T′*›
using *state-eq-trans*[*of T′ T*
‹*update-conflicting* (*Some* (*negate-ann-lits* (*trail S′*))) *S*›]
using *state-eq-trans*[*of T*
‹*update-conflicting* (*Some* (*negate-ann-lits* (*trail S′*))) *S*›
‹*update-conflicting* (*Some* (*negate-ann-lits* (*trail S′*))) *S′*›]
update-conflicting-state-eq[*of S S′* ‹*Some* {#}›]
apply (*auto simp*: *conflict-opt.simps state-eq-sym*)
using *reduce-trail-to-state-eq state-eq-trans update-conflicting-state-eq* **by** *blast*

context *optimal-encoding*
begin

definition *base-atm* :: ‹*′v* ⇒ *′v*› **where**
‹*base-atm L* = (*if L* ∈ Σ − ΔΣ *then L else*

if $L \in$ *replacement-neg* ‘ $\Delta\Sigma$ *then* (*SOME* K. ($K \in \Delta\Sigma \wedge L =$ *replacement-neg* K))
else (*SOME* K. ($K \in \Delta\Sigma \wedge L =$ *replacement-pos* K)))›

lemma *normalize-lit-Some-simp*[*simp*]: ‹(*SOME* K. $K \in \Delta\Sigma \wedge (L^{\mapsto 0} = K^{\mapsto 0})) = L$› **if** ‹$L \in \Delta\Sigma$› **for** K
by (*rule some1-equality*) (*use that* **in** *auto*)

lemma *base-atm-simps1*[*simp*]:
‹$L \in \Sigma \Longrightarrow L \notin \Delta\Sigma \Longrightarrow$ *base-atm* $L = L$›
by (*auto simp*: *base-atm-def*)

lemma *base-atm-simps2*[*simp*]:
‹$L \in (\Sigma - \Delta\Sigma) \cup$ *replacement-neg* ‘ $\Delta\Sigma \cup$ *replacement-pos* ‘ $\Delta\Sigma \Longrightarrow$
$K \in \Sigma \Longrightarrow K \notin \Delta\Sigma \Longrightarrow L \in \Sigma \Longrightarrow K =$ *base-atm* $L \longleftrightarrow L = K$›
by (*auto simp*: *base-atm-def*)

lemma *base-atm-simps3*[*simp*]:
‹$L \in \Sigma - \Delta\Sigma \Longrightarrow$ *base-atm* $L \in \Sigma$›
‹$L \in$ *replacement-neg* ‘ $\Delta\Sigma \cup$ *replacement-pos* ‘ $\Delta\Sigma \Longrightarrow$ *base-atm* $L \in \Delta\Sigma$›
apply (*auto simp*: *base-atm-def*)
by (*metis* (*mono-tags*, *lifting*) *tfl-some*)

lemma *base-atm-simps4*[*simp*]:
‹$L \in \Delta\Sigma \Longrightarrow$ *base-atm* (*replacement-pos* L) $= L$›
‹$L \in \Delta\Sigma \Longrightarrow$ *base-atm* (*replacement-neg* L) $= L$›
by (*auto simp*: *base-atm-def*)

fun *normalize-lit* :: ‹'*v literal* ⇒ '*v literal*› **where**
‹*normalize-lit* (*Pos* L) =
(*if* $L \in$ *replacement-neg* ‘ $\Delta\Sigma$
then Neg (*replacement-pos* (*SOME* K. ($K \in \Delta\Sigma \wedge L =$ *replacement-neg* K)))
else Pos L)› |
‹*normalize-lit* (*Neg* L) =
(*if* $L \in$ *replacement-neg* ‘ $\Delta\Sigma$
then Pos (*replacement-pos* (*SOME* K. $K \in \Delta\Sigma \wedge L =$ *replacement-neg* K))
else Neg L)›

abbreviation *normalize-clause* :: ‹'*v clause* ⇒ '*v clause*› **where**
‹*normalize-clause* $C \equiv$ *normalize-lit* ‘# C›

lemma *normalize-lit*[*simp*]:
‹$L \in \Sigma - \Delta\Sigma \Longrightarrow$ *normalize-lit* (*Pos* L) = (*Pos* L)›
‹$L \in \Sigma - \Delta\Sigma \Longrightarrow$ *normalize-lit* (*Neg* L) = (*Neg* L)›
‹$L \in \Delta\Sigma \Longrightarrow$ *normalize-lit* (*Pos* (*replacement-neg* L)) = *Neg* (*replacement-pos* L)›
‹$L \in \Delta\Sigma \Longrightarrow$ *normalize-lit* (*Neg* (*replacement-neg* L)) = *Pos* (*replacement-pos* L)›
by *auto*

definition *all-clauses-literals* :: ‹'*v list*› **where**
‹*all-clauses-literals* =
(*SOME xs*. *mset xs* = *mset-set* (($\Sigma - \Delta\Sigma$) ∪ *replacement-neg* ‘ $\Delta\Sigma$ ∪ *replacement-pos* ‘ $\Delta\Sigma$))›

datatype (**in** −) *'c search-depth* =
sd-is-zero: *SD-ZERO* (*the-search-depth*: *'c*) |
sd-is-one: *SD-ONE* (*the-search-depth*: *'c*) |
sd-is-two: *SD-TWO* (*the-search-depth*: *'c*)

abbreviation (**in** −) *un-hide-sd* :: ‹*'a search-depth list* ⇒ *'a list*› **where**
‹*un-hide-sd* ≡ *map the-search-depth*›

fun *nat-of-search-deph* :: ‹*'c search-depth* ⇒ *nat*› **where**
‹*nat-of-search-deph* (*SD-ZERO* -) = *0*› |
‹*nat-of-search-deph* (*SD-ONE* -) = *1*› |
‹*nat-of-search-deph* (*SD-TWO* -) = *2*›

definition *opposite-var* **where**
‹*opposite-var L* = (*if L* ∈ *replacement-pos* ‘ ΔΣ *then replacement-neg* (*base-atm L*)
else replacement-pos (*base-atm L*))›

lemma *opposite-var-replacement-if*[*simp*]:
‹*L* ∈ (*replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ) ⟹ *A* ∈ ΔΣ ⟹
opposite-var L = *replacement-pos A* ⟷ *L* = *replacement-neg A*›
‹*L* ∈ (*replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ) ⟹ *A* ∈ ΔΣ ⟹
opposite-var L = *replacement-neg A* ⟷ *L* = *replacement-pos A*›
‹*A* ∈ ΔΣ ⟹ *opposite-var* (*replacement-pos A*) = *replacement-neg A*›
‹*A* ∈ ΔΣ ⟹ *opposite-var* (*replacement-neg A*) = *replacement-pos A*›
by (*auto simp*: *opposite-var-def*)

context
assumes [*simp*]: ‹*finite* Σ›
begin

lemma *all-clauses-literals*:
‹*mset all-clauses-literals* = *mset-set* ((Σ − ΔΣ) ∪ *replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ)›
‹*distinct all-clauses-literals*›
‹*set all-clauses-literals* = ((Σ − ΔΣ) ∪ *replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ)›
proof −
let *?A* = ‹*mset-set* ((Σ − ΔΣ) ∪ *replacement-neg* ‘ ΔΣ ∪
replacement-pos ‘ ΔΣ)›
show *1*: ‹*mset all-clauses-literals* = *?A*›
using *someI*[*of* ‹λ*xs*. *mset xs* = *?A*›]
finite-Σ *ex-mset*[*of ?A*]
unfolding *all-clauses-literals-def*[*symmetric*]
by *metis*
show *2*: ‹*distinct all-clauses-literals*›
using *someI*[*of* ‹λ*xs*. *mset xs* = *?A*›]
finite-Σ *ex-mset*[*of ?A*]
unfolding *all-clauses-literals-def*[*symmetric*]
by (*metis distinct-mset-mset-set distinct-mset-mset-distinct*)
show *3*: ‹*set all-clauses-literals* = ((Σ − ΔΣ) ∪ *replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ)›
using *arg-cong*[*OF 1*, *of set-mset*] *finite*-Σ
by *simp*
qed

definition *unset-literals-in*-Σ **where**
‹*unset-literals-in*-Σ *M L* ⟷ *undefined-lit M* (*Pos L*) ∧ *L* ∈ Σ − ΔΣ›

definition *full-unset-literals-in-*ΔΣ **where**
‹*full-unset-literals-in-*ΔΣ *M L* ⟷
undefined-lit M (*Pos L*) ∧ *L* ∉ Σ − ΔΣ ∧ *undefined-lit M* (*Pos* (*opposite-var L*)) ∧
L ∈ *replacement-pos* ‘ ΔΣ›

definition *full-unset-literals-in-*ΔΣ′ **where**
‹*full-unset-literals-in-*ΔΣ′ *M L* ⟷
undefined-lit M (*Pos L*) ∧ *L* ∉ Σ − ΔΣ ∧ *undefined-lit M* (*Pos* (*opposite-var L*)) ∧
L ∈ *replacement-neg* ‘ ΔΣ›

definition *half-unset-literals-in-*ΔΣ **where**
‹*half-unset-literals-in-*ΔΣ *M L* ⟷
undefined-lit M (*Pos L*) ∧ *L* ∉ Σ − ΔΣ ∧ *defined-lit M* (*Pos* (*opposite-var L*))›

definition *sorted-unadded-literals* :: ‹(′*v*, ′*v clause*) *ann-lits* ⇒ ′*v list*› **where**
‹*sorted-unadded-literals M* =
(*let*
M0 = *filter* (*full-unset-literals-in-*ΔΣ′ *M*) *all-clauses-literals*;
— weight is 0
M1 = *filter* (*unset-literals-in-*Σ *M*) *all-clauses-literals*;
— weight is 2
M2 = *filter* (*full-unset-literals-in-*ΔΣ *M*) *all-clauses-literals*;
— weight is 2
M3 = *filter* (*half-unset-literals-in-*ΔΣ *M*) *all-clauses-literals*
— weight is 1
in
M0 @ *M3* @ *M1* @ *M2*)›

definition *complete-trail* :: ‹(′*v*, ′*v clause*) *ann-lits* ⇒ (′*v*, ′*v clause*) *ann-lits*› **where**
‹*complete-trail M* =
(*map* (*Decided o Pos*) (*sorted-unadded-literals M*) @ *M*)›

lemma *in-sorted-unadded-literals-undefD*:
‹*atm-of* (*lit-of l*) ∈ *set* (*sorted-unadded-literals M*) ⟹ *l* ∉ *set M*›
‹*atm-of* (*l*′) ∈ *set* (*sorted-unadded-literals M*) ⟹ *undefined-lit M l*′›
‹*xa* ∈ *set* (*sorted-unadded-literals M*) ⟹ *lit-of x* = *Neg xa* ⟹ *x* ∉ *set M*› **and**
set-sorted-unadded-literals[*simp*]:
‹*set* (*sorted-unadded-literals M*) =
Set.filter (λ*L*. *undefined-lit M* (*Pos L*)) (*set all-clauses-literals*)›
by (*auto simp*: *sorted-unadded-literals-def undefined-notin all-clauses-literals*(*1*,*2*)
*defined-lit-Neg-Pos-iff half-unset-literals-in-*ΔΣ*-def full-unset-literals-in-*ΔΣ*-def*
*unset-literals-in-*Σ*-def Let-def full-unset-literals-in-*ΔΣ′*-def*
all-clauses-literals(*3*))

lemma [*simp*]:
‹*full-unset-literals-in-*ΔΣ [] = (λ*L*. *L* ∈ *replacement-pos* ‘ ΔΣ)›
‹*full-unset-literals-in-*ΔΣ′ [] = (λ*L*. *L* ∈ *replacement-neg* ‘ ΔΣ)›
‹*half-unset-literals-in-*ΔΣ [] = (λ*L*. *False*)›
‹*unset-literals-in-*Σ [] = (λ*L*. *L* ∈ Σ − ΔΣ)›
by (*auto simp*: *full-unset-literals-in-*ΔΣ*-def*
*unset-literals-in-*Σ*-def full-unset-literals-in-*ΔΣ′*-def*
*half-unset-literals-in-*ΔΣ*-def intro*!: *ext*)

lemma *filter-disjount-union*:
‹(⋀*x*. *x* ∈ *set xs* ⟹ *P x* ⟹ ¬*Q x*) ⟹
length (*filter P xs*) + *length* (*filter Q xs*) =

length (*filter* ($\lambda x.$ *P x* $\vee$ *Q x*) *xs*)›
by (*induction xs*) *auto*

lemma *length-sorted-unadded-literals-empty*[*simp*]:
‹*length* (*sorted-unadded-literals* []) = *length all-clauses-literals*›
apply (*auto simp*: *sorted-unadded-literals-def sum-length-filter-compl*
Let-def ac-simps filter-disjount-union)
apply (*subst filter-disjount-union*)
apply *auto*
apply (*subst filter-disjount-union*)
apply *auto*
by (*metis* (*no-types*, *lifting*) *Diff-iff UnE all-clauses-literals*(*3*) *filter-True*)

lemma *sorted-unadded-literals-Cons-notin-all-clauses-literals*[*simp*]:
assumes
‹*atm-of* (*lit-of K*) $\notin$ *set all-clauses-literals*›
shows
‹*sorted-unadded-literals* (*K* # *M*) = *sorted-unadded-literals M*›
proof −
have [*simp*]: ‹*filter* (*full-unset-literals-in-*$\Delta\Sigma'$ (*K* # *M*))
all-clauses-literals =
filter (*full-unset-literals-in-*$\Delta\Sigma'$ *M*)
all-clauses-literals›
‹*filter* (*full-unset-literals-in-*$\Delta\Sigma$ (*K* # *M*))
all-clauses-literals =
filter (*full-unset-literals-in-*$\Delta\Sigma$ *M*)
all-clauses-literals›
‹*filter* (*half-unset-literals-in-*$\Delta\Sigma$ (*K* # *M*))
all-clauses-literals =
filter (*half-unset-literals-in-*$\Delta\Sigma$ *M*)
all-clauses-literals›
‹*filter* (*unset-literals-in-*Σ (*K* # *M*)) *all-clauses-literals* =
filter (*unset-literals-in-*Σ *M*) *all-clauses-literals*›
using *assms* **unfolding** *full-unset-literals-in-*$\Delta\Sigma'$*-def full-unset-literals-in-*$\Delta\Sigma$*-def*
half-unset-literals-in-$\Delta\Sigma$*-def unset-literals-in-*Σ*-def*
by (*auto simp*: *sorted-unadded-literals-def undefined-notin all-clauses-literals*(*1*,*2*)
defined-lit-Neg-Pos-iff all-clauses-literals(*3*) *defined-lit-cons*
intro!: *ext filter-cong*)

show *?thesis*
by (*auto simp*: *undefined-notin all-clauses-literals*(*1*,*2*)
defined-lit-Neg-Pos-iff all-clauses-literals(*3*) *sorted-unadded-literals-def*)
qed

lemma *sorted-unadded-literals-cong*:
assumes ‹$\bigwedge L.$ *L* $\in$ *set all-clauses-literals* $\Longrightarrow$ *defined-lit M* (*Pos L*) = *defined-lit M'* (*Pos L*)›
shows ‹*sorted-unadded-literals M* = *sorted-unadded-literals M'*›
proof −
have [*simp*]: ‹*filter* (*full-unset-literals-in-*$\Delta\Sigma'$ (*M*))
all-clauses-literals =
filter (*full-unset-literals-in-*$\Delta\Sigma'$ *M'*)
all-clauses-literals›
‹*filter* (*full-unset-literals-in-*$\Delta\Sigma$ (*M*))
all-clauses-literals =
filter (*full-unset-literals-in-*$\Delta\Sigma$ *M'*)
all-clauses-literals›
‹*filter* (*half-unset-literals-in-*$\Delta\Sigma$ (*M*))

all-clauses-literals =
filter (*half-unset-literals-in-*ΔΣ *M'*)
all-clauses-literals›
‹*filter* (*unset-literals-in-*Σ (*M*)) *all-clauses-literals* =
filter (*unset-literals-in-*Σ *M'*) *all-clauses-literals*›
using *assms* **unfolding** *full-unset-literals-in-*ΔΣ'*-def* *full-unset-literals-in-*ΔΣ*-def*
*half-unset-literals-in-*ΔΣ*-def* *unset-literals-in-*Σ*-def*
by (*auto simp*: *sorted-unadded-literals-def undefined-notin all-clauses-literals*(*1*,*2*)
defined-lit-Neg-Pos-iff all-clauses-literals(*3*) *defined-lit-cons*
intro!: *ext filter-cong*)

show *?thesis*
by (*auto simp*: *undefined-notin all-clauses-literals*(*1*,*2*)
defined-lit-Neg-Pos-iff all-clauses-literals(*3*) *sorted-unadded-literals-def*)

qed

lemma *sorted-unadded-literals-Cons-already-set*[*simp*]:
assumes
‹*defined-lit M* (*lit-of K*)›
shows
‹*sorted-unadded-literals* (*K* # *M*) = *sorted-unadded-literals M*›
by (*rule sorted-unadded-literals-cong*)
(*use assms* **in** ‹*auto simp*: *defined-lit-cons*›)

lemma *distinct-sorted-unadded-literals*[*simp*]:
‹*distinct* (*sorted-unadded-literals M*)›
unfolding *half-unset-literals-in-*ΔΣ*-def*
*full-unset-literals-in-*ΔΣ*-def* *unset-literals-in-*Σ*-def*
sorted-unadded-literals-def
*full-unset-literals-in-*ΔΣ'*-def*
by (*auto simp*: *sorted-unadded-literals-def all-clauses-literals*(*1*,*2*))

lemma *Collect-req-remove1*:
‹{*a* ∈ *A*. *a* ≠ *b* ∧ *P a*} = (*if P b then Set.remove b* {*a* ∈ *A*. *P a*} *else* {*a* ∈ *A*. *P a*})› **and**
Collect-req-remove2:
‹{*a* ∈ *A*. *b* ≠ *a* ∧ *P a*} = (*if P b then Set.remove b* {*a* ∈ *A*. *P a*} *else* {*a* ∈ *A*. *P a*})›
by *auto*

lemma *card-remove*:
‹*card* (*Set.remove a A*) = (*if a* ∈ *A then card A* − *1 else card A*)›
by (*auto simp*: *Set.remove-def*)

lemma *sorted-unadded-literals-cons-in-undef*[*simp*]:
‹*undefined-lit M* (*lit-of K*) ⟹
atm-of (*lit-of K*) ∈ *set all-clauses-literals* ⟹
Suc (*length* (*sorted-unadded-literals* (*K* # *M*))) =
length (*sorted-unadded-literals M*)›
by (*auto simp flip*: *distinct-card simp*: *Set.filter-def Collect-req-remove2*
card-remove isabelle-should-do-that-automatically
card-gt-0-iff simp flip: *less-eq-Suc-le*)

lemma *no-dup-complete-trail*[*simp*]:

‹no-dup (complete-trail M) ⟷ no-dup M›
by (auto simp: complete-trail-def no-dup-def comp-def all-clauses-literals(1,2) undefined-notin)

lemma tautology-complete-trail[simp]:
‹tautology (lit-of ‘# mset (complete-trail M)) ⟷ tautology (lit-of ‘# mset M)›
by (auto simp: complete-trail-def tautology-decomp′ comp-def all-clauses-literals
undefined-notin uminus-lit-swap defined-lit-Neg-Pos-iff
simp flip: defined-lit-Neg-Pos-iff)

lemma atms-of-complete-trail:
‹atms-of (lit-of ‘# mset (complete-trail M)) =
atms-of (lit-of ‘# mset M) ∪ (Σ − ΔΣ) ∪ replacement-neg ‘ ΔΣ ∪ replacement-pos ‘ ΔΣ›
by (auto simp add: complete-trail-def all-clauses-literals
image-image image-Un atms-of-def defined-lit-map)

fun depth-lit-of :: ‹(′v,-) ann-lit ⇒ (′v, -) ann-lit search-depth› **where**
‹depth-lit-of (Decided L) = SD-TWO (Decided L)› |
‹depth-lit-of (Propagated L C) = SD-ZERO (Propagated L C)›

fun depth-lit-of-additional-fst :: ‹(′v,-) ann-lit ⇒ (′v, -) ann-lit search-depth› **where**
‹depth-lit-of-additional-fst (Decided L) = SD-ONE (Decided L)› |
‹depth-lit-of-additional-fst (Propagated L C) = SD-ZERO (Propagated L C)›

fun depth-lit-of-additional-snd :: ‹(′v,-) ann-lit ⇒ (′v, -) ann-lit search-depth list› **where**
‹depth-lit-of-additional-snd (Decided L) = [SD-ONE (Decided L)]› |
‹depth-lit-of-additional-snd (Propagated L C) = []›

This function is suprisingly complicated to get right. Remember that the last set element is at the beginning of the list

fun remove-dup-information-raw :: ‹(′v, -) ann-lits ⇒ (′v, -) ann-lit search-depth list› **where**
‹remove-dup-information-raw [] = []› |
‹remove-dup-information-raw (L # M) =
(if atm-of (lit-of L) ∈ Σ − ΔΣ then depth-lit-of L # remove-dup-information-raw M
else if defined-lit (M) (Pos (opposite-var (atm-of (lit-of L))))
then if Decided (Pos (opposite-var (atm-of (lit-of L)))) ∈ set (M)
then remove-dup-information-raw M
else depth-lit-of-additional-fst L # remove-dup-information-raw M
else depth-lit-of-additional-snd L @ remove-dup-information-raw M)›

definition remove-dup-information **where**
‹remove-dup-information xs = un-hide-sd (remove-dup-information-raw xs)›

lemma [simp]: ‹the-search-depth (depth-lit-of L) = L›
by (cases L) auto

lemma length-complete-trail[simp]: ‹length (complete-trail []) = length all-clauses-literals›
unfolding complete-trail-def
by (auto simp: sum-length-filter-compl)

lemma distinct-count-list-if: ‹distinct xs ⟹ count-list xs x = (if x ∈ set xs then 1 else 0)›
by (induction xs) auto

lemma length-complete-trail-Cons:
‹no-dup (K # M) ⟹

length (*complete-trail* (*K* # *M*)) =
(*if atm-of* (*lit-of K*) ∈ *set all-clauses-literals then 0 else 1*) + *length* (*complete-trail M*)›
unfolding *complete-trail-def* **by** *auto*

lemma *length-complete-trail-eq*:
‹*no-dup M* ⟹ *atm-of* ‘ (*lits-of-l M*) ⊆ *set all-clauses-literals* ⟹
length (*complete-trail M*) = *length all-clauses-literals*›
by (*induction M rule*: *ann-lit-list-induct*) (*auto simp*: *length-complete-trail-Cons*)

lemma *in-set-all-clauses-literals-simp*[*simp*]:
‹*atm-of L* ∈ Σ − ΔΣ ⟹ *atm-of L* ∈ *set all-clauses-literals*›
‹*K* ∈ ΔΣ ⟹ *replacement-pos K* ∈ *set all-clauses-literals*›
‹*K* ∈ ΔΣ ⟹ *replacement-neg K* ∈ *set all-clauses-literals*›
by (*auto simp*: *all-clauses-literals*)

lemma [*simp*]:
‹*remove-dup-information* [] = []›
by (*auto simp*: *remove-dup-information-def*)

lemma *atm-of-remove-dup-information*:
‹*atm-of* ‘ (*lits-of-l M*) ⊆ *set all-clauses-literals* ⟹
atm-of ‘ (*lits-of-l* (*remove-dup-information M*)) ⊆ *set all-clauses-literals*›
unfolding *remove-dup-information-def*
apply (*induction M rule*: *ann-lit-list-induct*)
apply (*auto simp*: *Decided-Propagated-in-iff-in-lits-of-l lits-of-def image-image*)
done

primrec *remove-dup-information-raw2* :: ‹(′*v*, -) *ann-lits* ⇒ (′*v*, -) *ann-lits* ⇒
(′*v*, -) *ann-lit search-depth list*› **where**
‹*remove-dup-information-raw2 M′* [] = []› |
‹*remove-dup-information-raw2 M′* (*L* # *M*) =
(*if atm-of* (*lit-of L*) ∈ Σ − ΔΣ *then depth-lit-of L* # *remove-dup-information-raw2 M′ M*
else if defined-lit (*M* @ *M′*) (*Pos* (*opposite-var* (*atm-of* (*lit-of L*))))
then if Decided (*Pos* (*opposite-var* (*atm-of* (*lit-of L*)))) ∈ *set* (*M* @ *M′*)
then remove-dup-information-raw2 M′ M
else depth-lit-of-additional-fst L # *remove-dup-information-raw2 M′ M*
else depth-lit-of-additional-snd L @ *remove-dup-information-raw2 M′ M*)›

lemma *remove-dup-information-raw2-Nil*[*simp*]:
‹*remove-dup-information-raw2* [] *M* = *remove-dup-information-raw M*›
by (*induction M*) *auto*

This can be useful as simp, but I am not certain (yet), because the RHS does not look simpler than the LHS.

lemma *remove-dup-information-raw-cons*:
‹*remove-dup-information-raw* (*L* # *M2*) =
remove-dup-information-raw2 M2 [*L*] @
remove-dup-information-raw M2›
by (*auto simp*: *defined-lit-append*)

lemma *remove-dup-information-raw-append*:
‹*remove-dup-information-raw* (*M1* @ *M2*) =
remove-dup-information-raw2 M2 M1 @
remove-dup-information-raw M2›

by (*induction M1*)
(*auto simp*: *defined-lit-append*)

lemma *remove-dup-information-raw-append2*:
‹*remove-dup-information-raw2 M* (*M1* @ *M2*) =
remove-dup-information-raw2 (*M* @ *M2*) *M1* @
remove-dup-information-raw2 M M2›
by (*induction M1*)
(*auto simp*: *defined-lit-append*)

lemma *remove-dup-information-subset*: ‹*mset* (*remove-dup-information M*) ⊆# *mset M*›
unfolding *remove-dup-information-def*
apply (*induction M rule*: *ann-lit-list-induct*) **apply** *auto*
apply (*metis add-mset-remove-trivial diff-subset-eq-self subset-mset.dual-order.trans*)+
done

lemma *no-dup-subsetD*: ‹*no-dup M* ⟹ *mset M'* ⊆# *mset M* ⟹ *no-dup M'*›
unfolding *no-dup-def distinct-mset-mset-distinct*[*symmetric*] *mset-map*
apply (*drule image-mset-subseteq-mono*[*of* - - ‹*atm-of o lit-of*›])
apply (*drule distinct-mset-mono*)
apply *auto*
done

lemma *no-dup-remove-dup-information*:
‹*no-dup M* ⟹ *no-dup* (*remove-dup-information M*)›
using *no-dup-subsetD*[*OF* - *remove-dup-information-subset*] **by** *blast*

lemma *atm-of-complete-trail*:
‹*atm-of* ‘ (*lits-of-l M*) ⊆ *set all-clauses-literals* ⟹
atm-of ‘ (*lits-of-l* (*complete-trail M*)) = *set all-clauses-literals*›
unfolding *complete-trail-def* **by** (*auto simp*: *lits-of-def image-image image-Un defined-lit-map*)

lemmas [*simp del*] =
remove-dup-information-raw.simps
remove-dup-information-raw2.simps

lemmas [*simp*] =
remove-dup-information-raw-append
remove-dup-information-raw-cons
remove-dup-information-raw-append2

definition *truncate-trail* :: ‹('*v*, -) *ann-lits* ⇒ -› **where**
‹*truncate-trail M* ≡
(*snd* (*backtrack-split M*))›

definition *ocdcl-score* :: ‹('*v*, -) *ann-lits* ⇒ -› **where**
‹*ocdcl-score M* =
rev (*map nat-of-search-deph* (*remove-dup-information-raw* (*complete-trail* (*truncate-trail M*))))›

interpretation *enc-weight-opt*: *conflict-driven-clause-learning*$_W$*-optimal-weight* **where**
state-eq = *state-eq* **and**
state = *state* **and**
trail = *trail* **and**

init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state* **and**
$\varrho = \varrho_e$ **and**
update-additional-info = *update-additional-info*
apply *unfold-locales*
subgoal by (*rule* ϱ_e*-mono*)
subgoal using *update-additional-info* **by** *fast*
subgoal using *weight-init-state* **by** *fast*
done

lemma
‹(*a*, *b*) ∈ *lexn less-than n* ⟹ (*b*, *c*) ∈ *lexn less-than n* ∨ *b* = *c* ⟹ (*a*, *c*) ∈ *lexn less-than n*›
‹(*a*, *b*) ∈ *lexn less-than n* ⟹ (*b*, *c*) ∈ *lexn less-than n* ∨ *b* = *c* ⟹ (*a*, *c*) ∈ *lexn less-than n*›
apply (*auto intro*:)
apply (*meson lexn-transI trans-def trans-less-than*)+
done

lemma *truncate-trail-Prop*[*simp*]:
‹*truncate-trail* (*Propagated L E* # *S*) = *truncate-trail* (*S*)›
by (*auto simp*: *truncate-trail-def*)

lemma *ocdcl-score-Prop*[*simp*]:
‹*ocdcl-score* (*Propagated L E* # *S*) = *ocdcl-score* (*S*)›
by (*auto simp*: *ocdcl-score-def truncate-trail-def*)

lemma *remove-dup-information-raw2-undefined-Σ*:
‹*distinct xs* ⟹
(⋀*L*. *L* ∈ *set xs* ⟹ *undefined-lit M* (*Pos L*) ⟹ *L* ∈ Σ ⟹ *undefined-lit MM* (*Pos L*)) ⟹
remove-dup-information-raw2 MM
(*map* (*Decided* ∘ *Pos*)
(*filter* (*unset-literals-in-Σ M*)
xs)) =
map (*SD-TWO o Decided* ∘ *Pos*)
(*filter* (*unset-literals-in-Σ M*)
xs)›
by (*induction xs*)
(*auto simp*: *remove-dup-information-raw2.simps*
unset-literals-in-Σ-def)

lemma *defined-lit-map-Decided-pos*:
‹*defined-lit* (*map* (*Decided* ∘ *Pos*) *M*) *L* ⟷ *atm-of L* ∈ *set M*›
by (*induction M*) (*auto simp*: *defined-lit-cons*)

lemma *remove-dup-information-raw2-full-undefined-Σ*:
‹*distinct xs* ⟹ *set xs* ⊆ *set all-clauses-literals* ⟹
(⋀*L*. *L* ∈ *set xs* ⟹ *undefined-lit M* (*Pos L*) ⟹ *L* ∉ Σ − ΔΣ ⟹
undefined-lit M (*Pos* (*opposite-var L*)) ⟹ *L* ∈ *replacement-pos* ‘ ΔΣ ⟹
undefined-lit MM (*Pos* (*opposite-var L*))) ⟹
remove-dup-information-raw2 MM

(*map* (*Decided* ∘ *Pos*)
(*filter* (*full-unset-literals-in-*ΔΣ *M*)
xs)) =
map (*SD-ONE o Decided* ∘ *Pos*)
(*filter* (*full-unset-literals-in-*ΔΣ *M*)
xs)›
unfolding *all-clauses-literals*
apply (*induction xs*)
subgoal
by (*simp-all add*: *remove-dup-information-raw2.simps*)
subgoal premises *p* **for** *L xs*
using *p*(*1*−*3*) *p*(*4*)[*of L*] *p*(*4*)
by (*clarsimp simp add*: *remove-dup-information-raw2.simps*
defined-lit-map-Decided-pos
*full-unset-literals-in-*ΔΣ*-def defined-lit-append*)
done

lemma *full-unset-literals-in-*ΔΣ*-notin*[*simp*]:
‹*La* ∈ Σ ⟹ *full-unset-literals-in-*ΔΣ *M La* ⟷ *False*›
‹*La* ∈ Σ ⟹ *full-unset-literals-in-*ΔΣ′ *M La* ⟷ *False*›
apply (*metis* (*mono-tags*) *full-unset-literals-in-*ΔΣ*-def*
image-iff new-vars-pos)
by (*simp add*: *full-unset-literals-in-*ΔΣ′*-def image-iff*)

lemma *Decided-in-definedD*: ‹*Decided K* ∈ *set M* ⟹ *defined-lit M K*›
by (*simp add*: *defined-lit-def*)

lemma *full-unset-literals-in-*ΔΣ′*-full-unset-literals-in-*ΔΣ:
‹*L* ∈ *replacement-pos* ‘ ΔΣ ∪ *replacement-neg* ‘ ΔΣ ⟹
*full-unset-literals-in-*ΔΣ′ *M* (*opposite-var L*) ⟷ *full-unset-literals-in-*ΔΣ *M L*›
by (*auto simp*: *full-unset-literals-in-*ΔΣ′*-def full-unset-literals-in-*ΔΣ*-def*
opposite-var-def)

lemma *remove-dup-information-raw2-full-unset-literals-in-*ΔΣ′:
‹(⋀*L*. *L* ∈ *set* (*filter* (*full-unset-literals-in-*ΔΣ′ *M*) *xs*) ⟹ *Decided* (*Pos* (*opposite-var L*)) ∈ *set M*′)
⟹
set xs ⊆ *set all-clauses-literals* ⟹
(*remove-dup-information-raw2*
M′
(*map* (*Decided* ∘ *Pos*)
(*filter* (*full-unset-literals-in-*ΔΣ′ (*M*))
xs))) = []›
supply [[*goals-limit*=*1*]]
apply (*induction xs*)
subgoal by (*auto simp*: *remove-dup-information-raw2.simps*)
subgoal premises *p* **for** *L xs*
using *p*
by (*force simp add*: *remove-dup-information-raw2.simps*
*full-unset-literals-in-*ΔΣ′*-full-unset-literals-in-*ΔΣ
all-clauses-literals
defined-lit-map-Decided-pos defined-lit-append image-iff
dest: *Decided-in-definedD*)
done

lemma
fixes *M* :: ‹(′*v*, -) *ann-lits*› **and** *L* :: ‹(′*v*, -) *ann-lit*›

defines ‹*n1* ≡ *map nat-of-search-deph* (*remove-dup-information-raw* (*complete-trail* (*L* # *M*)))› **and**
‹*n2* ≡ *map nat-of-search-deph* (*remove-dup-information-raw* (*complete-trail M*))›
assumes
lits: ‹*atm-of* ‘ (*lits-of-l* (*L* # *M*)) ⊆ *set all-clauses-literals*› **and**
undef: ‹*undefined-lit M* (*lit-of L*)›
shows
‹(*rev n1*, *rev n2*) ∈ *lexn less-than n* ∨ *n1* = *n2*›
proof −
show *?thesis*
using *lits*
apply (*auto simp*: *n1-def n2-def complete-trail-def prepend-same-lexn*)
apply (*auto simp*: *sorted-unadded-literals-def*
remove-dup-information-raw2.simps all-clauses-literals(*2*) *defined-lit-map-Decided-pos*
*remove-dup-information-raw2-undefined-*Σ)
subgoal
apply (*subst remove-dup-information-raw2-undefined-*Σ)
apply (*simp-all add*: *all-clauses-literals*(*2*) *defined-lit-map-Decided-pos*
*remove-dup-information-raw2-undefined-*Σ)
apply (*subst remove-dup-information-raw2-full-undefined-*Σ)
apply (*auto simp*: *all-clauses-literals*(*2*))
apply (*subst remove-dup-information-raw2-full-unset-literals-in-*ΔΣ′)
apply (*auto simp*: *full-unset-literals-in-*ΔΣ′*-full-unset-literals-in-*ΔΣ)[*2*]
oops
lemma
defines ‹*n* ≡ *card* Σ›
assumes
‹*init-clss S* = *penc N*› **and**
‹*enc-weight-opt.cdcl-bnb-stgy S T*› **and**
struct: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*)› **and**
smaller-propa: ‹*no-smaller-propa S*› **and**
smaller-confl: ‹*cdcl-bnb-stgy-inv S*›
shows ‹(*ocdcl-score* (*trail T*), *ocdcl-score* (*trail S*)) ∈ *lexn less-than n* ∨
ocdcl-score (*trail T*) = *ocdcl-score* (*trail S*)›
using *assms*(*3*)
proof (*cases*)
case *cdcl-bnb-conflict*
then show *?thesis* **by** (*auto elim*!: *rulesE*)
next
case *cdcl-bnb-propagate*
then show *?thesis*
by (*auto elim*!: *rulesE*)
next
case *cdcl-bnb-improve*
then show *?thesis*
by (*auto elim*!: *enc-weight-opt.improveE*)
next
case *cdcl-bnb-conflict-opt*
then show *?thesis*
by (*auto elim*!: *enc-weight-opt.conflict-optE*)
next
case *cdcl-bnb-other′*
then show *?thesis*
proof *cases*
case *bj*
then show *?thesis*
proof *cases*

case *skip*
then show *?thesis* **by** (*auto elim*!: *rulesE*)
next
case *resolve*
then show *?thesis* **by** (*cases* ‹*trail S*›) (*auto elim*!: *rulesE*)
next
case *backtrack*
then obtain *M1 M2* :: ‹('*v*, '*v clause*) *ann-lits*› **and** *K L* :: ‹'*v literal*› **and**
D D' :: ‹'*v clause*› **where**
confl: ‹*conflicting S* = *Some* (*add-mset L D*)› **and**
decomp: ‹(*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail S*))› **and**
‹*get-maximum-level* (*trail S*) (*add-mset L D'*) = *local.backtrack-lvl S*› **and**
‹*get-level* (*trail S*) *L* = *local.backtrack-lvl S*› **and**
lev-K: ‹*get-level* (*trail S*) *K* = *Suc* (*get-maximum-level* (*trail S*) *D'*)› **and**
D'-D: ‹*D'* ⊆# *D*› **and**
‹*set-mset* (*clauses S*) ∪ *set-mset* (*enc-weight-opt.conflicting-clss S*) ⊨*p*
add-mset L D'› **and**
T: ‹*T* ∼
cons-trail (*Propagated L* (*add-mset L D'*))
(*reduce-trail-to M1*
(*add-learned-cls* (*add-mset L D'*) (*update-conflicting None S*)))›
by (*auto simp*: *enc-weight-opt.obacktrack.simps*)
have
tr-D: ‹*trail S* ⊨*as CNot* (*add-mset L D*)› **and**
‹*distinct-mset* (*add-mset L D*)› **and**
‹$cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv* (*abs-state S*)› **and**
n-d: ‹*no-dup* (*trail S*)›
using *struct confl*
unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-conflicting-def*
$cdcl_W$*-restart-mset.distinct-*$cdcl_W$*-state-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv-def*
by *auto*
have *tr-D'*: ‹*trail S* ⊨*as CNot* (*add-mset L D'*)›
using *D'-D tr-D*
by (*auto simp*: *true-annots-true-cls-def-iff-negation-in-model*)
have ‹*trail S* ⊨*as CNot D'* ⟹ *trail S* ⊨*as CNot* (*normalize2 D'*)›
if ‹*get-maximum-level* (*trail S*) *D'* < *backtrack-lvl S*›
for *D'*
oops

end

interpretation *enc-weight-opt*: *conflict-driven-clause-learning*$_W$*-optimal-weight* **where**
state-eq = *state-eq* **and**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**

init-state = *init-state* **and**
$\varrho = \varrho_e$ **and**
update-additional-info = *update-additional-info*
apply *unfold-locales*
subgoal by (*rule* ϱ_e*-mono*)
subgoal using *update-additional-info* **by** *fast*
subgoal using *weight-init-state* **by** *fast*
done

inductive *simple-backtrack-conflict-opt* :: ‹'*st* ⇒ '*st* ⇒ *bool*› **where**
‹*simple-backtrack-conflict-opt S T*›
if
‹*backtrack-split* (*trail S*) = (*M2*, *Decided K* # *M1*)› **and**
‹*negate-ann-lits* (*trail S*) ∈# *enc-weight-opt.conflicting-clss S*› **and**
‹*conflicting S* = *None*› **and**
‹*T* ∼ *cons-trail* (*Propagated* (−*K*) (*DECO-clause* (*trail S*)))
(*add-learned-cls* (*DECO-clause* (*trail S*)) (*reduce-trail-to M1 S*))›

inductive-cases *simple-backtrack-conflict-optE*: ‹*simple-backtrack-conflict-opt S T*›

lemma *simple-backtrack-conflict-opt-conflict-analysis*:
assumes ‹*simple-backtrack-conflict-opt S U*› **and**
inv: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*)›
shows ‹∃ *T T'*. *enc-weight-opt.conflict-opt S T* ∧ *resolve*** *T T'*
∧ *enc-weight-opt.obacktrack T' U*›
using *assms*
proof (*cases rule*: *simple-backtrack-conflict-opt.cases*)
case (*1 M2 K M1*)
have *tr*: ‹*trail S* = *M2* @ *Decided K* # *M1*›
using *1 backtrack-split-list-eq*[*of* ‹*trail S*›]
by *auto*
let *?S* = ‹*update-conflicting* (*Some* (*negate-ann-lits* (*trail S*))) *S*›
have ‹*enc-weight-opt.conflict-opt S ?S*›
by (*rule enc-weight-opt.conflict-opt.intros*[*OF 1*(*2*,*3*)]) *auto*

let *?T* = ‹λ*n*. *update-conflicting*
(*Some* (*negate-ann-lits* (*drop n* (*trail S*))))
(*reduce-trail-to* (*drop n* (*trail S*)) *S*)›
have *proped-M2*: ‹*is-proped* (*M2* ! *n*)› **if** ‹*n* < *length M2*› **for** *n*
using *that 1*(*1*) *nth-length-takeWhile*[*of* ‹*Not* ∘ *is-decided*› ‹*trail S*›]
length-takeWhile-le[*of* ‹*Not* ∘ *is-decided*› ‹*trail S*›]
unfolding *backtrack-split-takeWhile-dropWhile*
apply *auto*
by (*metis annotated-lit.exhaust-disc comp-apply nth-mem set-takeWhileD*)
have *is-dec-M2*[*simp*]: ‹*filter-mset is-decided* (*mset M2*) = {#}›
using *1*(*1*) *nth-length-takeWhile*[*of* ‹*Not* ∘ *is-decided*› ‹*trail S*›]
length-takeWhile-le[*of* ‹*Not* ∘ *is-decided*› ‹*trail S*›]
unfolding *backtrack-split-takeWhile-dropWhile*
apply (*auto simp*: *filter-mset-empty-conv*)
by (*metis annotated-lit.exhaust-disc comp-apply nth-mem set-takeWhileD*)
have *n-d*: ‹*no-dup* (*trail S*)› **and**
le: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-conflicting* (*enc-weight-opt.abs-state S*)› **and**
dist: ‹$cdcl_W$*-restart-mset.distinct-*$cdcl_W$*-state* (*enc-weight-opt.abs-state S*)› **and**
decomp-imp: ‹*all-decomposition-implies-m* (*clauses S* + (*enc-weight-opt.conflicting-clss S*))
(*get-all-ann-decomposition* (*trail S*))› **and**
learned: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause* (*enc-weight-opt.abs-state S*)›

```
    using inv
    unfolding cdcl_W-restart-mset.cdcl_W-all-struct-inv-def
      cdcl_W-restart-mset.cdcl_W-M-level-inv-def
    by auto
  then have [simp]: ‹K ≠ lit-of (M2 ! n)› if ‹n < length M2› for n
    using that unfolding tr
    by (auto simp: defined-lit-nth)
  have n-d-n: ‹no-dup (drop n M2 @ Decided K # M1)› for n
    using n-d unfolding tr
    by (subst (asm) append-take-drop-id[symmetric, of - n])
      (auto simp del: append-take-drop-id dest: no-dup-appendD)
  have mark-dist: ‹distinct-mset (mark-of (M2!n))› if ‹n < length M2› for n
    using dist that proped-M2[OF that] nth-mem[OF that]
    unfolding cdcl_W-restart-mset.distinct-cdcl_W-state-def tr
    by (cases ‹M2!n›) (auto simp: tr)

  have [simp]: ‹undefined-lit (drop n M2) K› for n
    using n-d defined-lit-mono[of ‹drop n M2› K M2]
    unfolding tr
    by (auto simp: set-drop-subset)
  from this[of 0] have [simp]: ‹undefined-lit M2 K›
    by auto
  have [simp]: ‹count-decided (drop n M2) = 0› for n
    apply (subst count-decided-0-iff)
    using 1(1) nth-length-takeWhile[of ‹Not ∘ is-decided› ‹trail S›]
    length-takeWhile-le[of ‹Not ∘ is-decided› ‹trail S›]
    unfolding backtrack-split-takeWhile-dropWhile
    by (auto simp: dest!: in-set-dropD set-takeWhileD)
  from this[of 0] have [simp]: ‹count-decided M2 = 0› by simp
  have proped: ‹⋀L mark a b.
      a @ Propagated L mark # b = trail S ⟶
      b ⊨as CNot (remove1-mset L mark) ∧ L ∈# mark›
    using le
    unfolding cdcl_W-restart-mset.cdcl_W-conflicting-def
    by auto
  have mark: ‹drop (Suc n) M2 @ Decided K # M1 ⊨as
      CNot (mark-of (M2 ! n) − unmark (M2 ! n)) ∧
      lit-of (M2 ! n) ∈# mark-of (M2 ! n)›
    if ‹n < length M2› for n
    using proped-M2[OF that] that
      append-take-drop-id[of n M2, unfolded Cons-nth-drop-Suc[OF that, symmetric]]
      proped[of ‹take n M2› ‹lit-of (M2 ! n)› ‹mark-of (M2 ! n)›
    ‹drop (Suc n) M2 @ Decided K # M1›]
    unfolding tr by (cases ‹M2!n›) auto
  have confl: ‹enc-weight-opt.conflict-opt S ?S›
    by (rule enc-weight-opt.conflict-opt.intros) (use 1 in auto)
  have res: ‹resolve** ?S (?T n)› if ‹n ≤ length M2› for n
    using that unfolding tr
  proof (induction n)
    case 0
    then show ?case
      using get-all-ann-decomposition-backtrack-split[THEN iffD1, OF 1(1)]
        1
      by (cases ‹get-all-ann-decomposition (trail S)›) (auto simp: tr)
  next
    case (Suc n)
```

```
have [simp]: ‹¬ Suc (length M2 − Suc n) < length M2 ⟷ n = 0›
  using Suc(2) by auto
have [simp]: ‹reduce-trail-to (drop (Suc 0) M2 @ Decided K # M1) S = tl-trail S›
  apply (subst reduce-trail-to.simps)
  using Suc by (auto simp: tr )
have [simp]: ‹reduce-trail-to (M2 ! 0 # drop (Suc 0) M2 @ Decided K # M1) S = S›
  apply (subst reduce-trail-to.simps)
  using Suc by (auto simp: tr )
have [simp]: ‹(Suc (length M1) −
      (length M2 − n + (Suc (length M1) − (n − length M2)))) = 0›
  ‹(Suc (length M2 + length M1) −
      (length M2 − n + (Suc (length M1) − (n − length M2)))) =n›
  ‹length M2 − n + (Suc (length M1) − (n − length M2)) = Suc (length M2 + length M1) − n›
  using Suc by auto
have [symmetric,simp]: ‹M2 ! n = Propagated (lit-of (M2 ! n)) (mark-of (M2 ! n))›
  using Suc proped-M2[of n]
  by (cases ‹M2 ! n›) (auto simp: tr trail-reduce-trail-to-drop hd-drop-conv-nth
    intro!: resolve.intros)
have ‹− lit-of (M2 ! n) ∈# negate-ann-lits (drop n M2 @ Decided K # M1)›
  using Suc in-set-dropI[of ‹n› ‹map (uminus o lit-of) M2› n]
  by (simp add: negate-ann-lits-def comp-def drop-map
     del: nth-mem)
moreover have ‹get-maximum-level (drop n M2 @ Decided K # M1)
   (remove1-mset (− lit-of (M2 ! n)) (negate-ann-lits (drop n M2 @ Decided K # M1))) =
  Suc (count-decided M1)›
  using Suc(2) count-decided-ge-get-maximum-level[of ‹drop n M2 @ Decided K # M1›
    ‹(remove1-mset (− lit-of (M2 ! n)) (negate-ann-lits (drop n M2 @ Decided K # M1)))›]
  by (auto simp: negate-ann-lits-def tr max-def ac-simps
    remove1-mset-add-mset-If get-maximum-level-add-mset
   split: if-splits)
moreover have ‹lit-of (M2 ! n) ∈# mark-of (M2 ! n)›
  using mark[of n] Suc by auto
moreover have ‹(remove1-mset (− lit-of (M2 ! n))
     (negate-ann-lits (drop n M2 @ Decided K # M1)) ∪#
    (mark-of (M2 ! n) − unmark (M2 ! n))) = negate-ann-lits (drop (Suc n) (trail S))›
  apply (rule distinct-set-mset-eq)
  using n-d-n[of n] n-d-n[of ‹Suc n›] no-dup-distinct-mset[OF n-d-n[of n]] Suc
    mark[of n] mark-dist[of n]
  by (auto simp: tr Cons-nth-drop-Suc[symmetric, of n]
      entails-CNot-negate-ann-lits
    dest: in-diffD intro: distinct-mset-minus)
moreover { have 1: ‹(tl-trail
   (reduce-trail-to (drop n M2 @ Decided K # M1) S)) ∼
    (reduce-trail-to (drop (Suc n) M2 @ Decided K # M1) S)›
  apply (subst Cons-nth-drop-Suc[symmetric, of n M2])
  subgoal using Suc by (auto simp: tl-trail-update-conflicting)
  subgoal
    apply (rule state-eq-trans)
   apply simp
   apply (cases ‹length (M2 ! n # drop (Suc n) M2 @ Decided K # M1) < length (trail S)›)
   apply (auto simp: tl-trail-reduce-trail-to-cons tr)
   done
  done
have ‹update-conflicting
 (Some (negate-ann-lits (drop (Suc n) M2 @ Decided K # M1)))
 (reduce-trail-to (drop (Suc n) M2 @ Decided K # M1) S) ∼
```

```
    update-conflicting
     (Some (negate-ann-lits (drop (Suc n) M2 @ Decided K # M1)))
     (tl-trail
       (update-conflicting (Some (negate-ann-lits (drop n M2 @ Decided K # M1)))
         (reduce-trail-to (drop n M2 @ Decided K # M1) S)))›
       apply (rule state-eq-trans)
       prefer 2
       apply (rule update-conflicting-state-eq)
       apply (rule tl-trail-update-conflicting[THEN state-eq-sym[THEN iffD1]])
       apply (subst state-eq-sym)
       apply (subst update-conflicting-update-conflicting)
       apply (rule 1)
       by fast }
    ultimately have ‹resolve (?T n) (?T (n+1))› apply −
      apply (rule resolve.intros[of - ‹lit-of (M2 ! n)› ‹mark-of (M2 ! n)›])
      using Suc
        get-all-ann-decomposition-backtrack-split[THEN iffD1, OF 1(1)]
         in-get-all-ann-decomposition-trail-update-trail[of ‹Decided K› M1 ‹M2› ‹S›]
      by (auto simp: tr trail-reduce-trail-to-drop hd-drop-conv-nth
        intro!: resolve.intros intro: update-conflicting-state-eq)
    then show ?case
      using Suc by (auto simp add: tr)
  qed

 have ‹get-maximum-level (Decided K # M1) (DECO-clause M1) = get-maximum-level M1 (DECO-clause
M1)›
    by (rule get-maximum-level-cong)
      (use n-d in ‹auto simp: tr get-level-cons-if atm-of-eq-atm-of
      DECO-clause-def Decided-Propagated-in-iff-in-lits-of-l lits-of-def›)
  also have ‹... = count-decided M1›
    using n-d unfolding tr apply −
    apply (induction M1 rule: ann-lit-list-induct)
    subgoal by auto
    subgoal for L M1'
       apply (subgoal-tac ‹∀ La∈#DECO-clause M1'. get-level (Decided L # M1') La = get-level M1'
La›)
      subgoal
        using count-decided-ge-get-maximum-level[of ‹M1'› ‹DECO-clause M1'›]
        get-maximum-level-cong[of ‹DECO-clause M1'› ‹Decided L # M1'› ‹M1'›]
       by (auto simp: get-maximum-level-add-mset tr atm-of-eq-atm-of
        max-def)
      subgoal
        by (auto simp: DECO-clause-def
          get-level-cons-if atm-of-eq-atm-of Decided-Propagated-in-iff-in-lits-of-l
          lits-of-def)
       done
   subgoal for L C M1'
       apply (subgoal-tac ‹∀ La∈#DECO-clause M1'. get-level (Propagated L C # M1') La = get-level
M1' La›)
      subgoal
        using count-decided-ge-get-maximum-level[of ‹M1'› ‹DECO-clause M1'›]
        get-maximum-level-cong[of ‹DECO-clause M1'› ‹Propagated L C # M1'› ‹M1'›]
       by (auto simp: get-maximum-level-add-mset tr atm-of-eq-atm-of
        max-def)
      subgoal
        by (auto simp: DECO-clause-def
```

```
        get-level-cons-if atm-of-eq-atm-of Decided-Propagated-in-iff-in-lits-of-l
        lits-of-def)
      done
    done
  finally have max: ‹get-maximum-level (Decided K # M1) (DECO-clause M1) = count-decided M1›
.
  have ‹trail S ⊨as CNot (negate-ann-lits (trail S))›
    by (auto simp: true-annots-true-cls-def-iff-negation-in-model
      negate-ann-lits-def lits-of-def)
  then have ‹clauses S + (enc-weight-opt.conflicting-clss S) ⊨pm DECO-clause (trail S)›
     unfolding DECO-clause-def apply −
    apply (rule all-decomposition-implies-conflict-DECO-clause[OF decomp-imp,
      of ‹negate-ann-lits (trail S)›])
    using 1
    by auto

  have neg: ‹trail S ⊨as CNot (mset (map (uminus o lit-of) (trail S)))›
    by (auto simp: true-annots-true-cls-def-iff-negation-in-model
      lits-of-def)
  have ent: ‹clauses S + enc-weight-opt.conflicting-clss S ⊨pm DECO-clause (trail S)›
    unfolding DECO-clause-def
    by (rule all-decomposition-implies-conflict-DECO-clause[OF decomp-imp,
          of ‹mset (map (uminus o lit-of) (trail S))›])
      (use neg 1 in ‹auto simp: negate-ann-lits-def›)
  have deco: ‹DECO-clause (M2 @ Decided K # M1) = add-mset (− K) (DECO-clause M1)›
    by (auto simp: DECO-clause-def)
  have eg: ‹reduce-trail-to M1 (reduce-trail-to (Decided K # M1) S) ∼
    reduce-trail-to M1 S›
    apply (subst reduce-trail-to-compow-tl-trail-le)
    apply (solves ‹auto simp: tr›)
    apply (subst (3)reduce-trail-to-compow-tl-trail-le)
    apply (solves ‹auto simp: tr›)
    apply (auto simp: tr)
    apply (cases ‹M2 = []›)
    apply (auto simp: reduce-trail-to-compow-tl-trail-le reduce-trail-to-compow-tl-trail-eq tr)
    done

  have U: ‹cons-trail (Propagated (− K) (DECO-clause (M2 @ Decided K # M1)))
     (add-learned-cls (DECO-clause (M2 @ Decided K # M1))
       (reduce-trail-to M1 S)) ∼
    cons-trail (Propagated (− K) (add-mset (− K) (DECO-clause M1)))
     (reduce-trail-to M1
       (add-learned-cls (add-mset (− K) (DECO-clause M1))
         (update-conflicting None
           (update-conflicting (Some (add-mset (− K) (negate-ann-lits M1)))
             (reduce-trail-to (Decided K # M1) S)))))›
    unfolding deco
    apply (rule cons-trail-state-eq)
    apply (rule state-eq-trans)
    prefer 2
    apply (rule state-eq-sym[THEN iffD1])
    apply (rule reduce-trail-to-add-learned-cls-state-eq)
    apply (solves ‹auto simp: tr›)
    apply (rule add-learned-cls-state-eq)
    apply (rule state-eq-trans)
    prefer 2
```

```
    apply (rule state-eq-sym[THEN iffD1])
    apply (rule reduce-trail-to-update-conflicting-state-eq)
    apply (solves ‹auto simp: tr›)
    apply (rule state-eq-trans)
    prefer 2
    apply (rule state-eq-sym[THEN iffD1])
    apply (rule update-conflicting-state-eq)
    apply (rule reduce-trail-to-update-conflicting-state-eq)
    apply (solves ‹auto simp: tr›)
    apply (rule state-eq-trans)
    prefer 2
    apply (rule state-eq-sym[THEN iffD1])
    apply (rule update-conflicting-update-conflicting)
    apply (rule eg)
    apply (rule state-eq-trans)
    prefer 2
    apply (rule state-eq-sym[THEN iffD1])
    apply (rule update-conflicting-itself)
    by (use 1 in auto)

  have bt: ‹enc-weight-opt.obacktrack (?T (length M2)) U›
    apply (rule enc-weight-opt.obacktrack.intros[of - ‹−K› ‹negate-ann-lits M1› K M1 ‹[]›
      ‹DECO-clause M1› ‹count-decided M1›])
    subgoal by (auto simp: tr)
    subgoal by (auto simp: tr)
    subgoal by (auto simp: tr)
    subgoal
      using count-decided-ge-get-maximum-level[of ‹Decided K # M1› ‹DECO-clause M1›]
      by (auto simp: tr get-maximum-level-add-mset max-def)
    subgoal using max by (auto simp: tr)
    subgoal by (auto simp: tr)
    subgoal by (auto simp: DECO-clause-def negate-ann-lits-def
      image-mset-subseteq-mono)
    subgoal using ent by (auto simp: tr DECO-clause-def)
    subgoal
      apply (rule state-eq-trans [OF 1(4)])
      using 1(4) U by (auto simp: tr)
    done

  show ?thesis
    using confl res[of ‹length M2›, simplified] bt
    by blast
qed

inductive conflict-opt0 :: ‹'st ⇒ 'st ⇒ bool› where
  ‹conflict-opt0 S T›
  if
    ‹count-decided (trail S) = 0› and
    ‹negate-ann-lits (trail S) ∈# enc-weight-opt.conflicting-clss S› and
    ‹conflicting S = None› and
    ‹T ∼ update-conflicting (Some {#}) (reduce-trail-to ([] :: ('v, 'v clause) ann-lits) S)›

inductive-cases conflict-opt0E: ‹conflict-opt0 S T›

inductive cdcl-dpll-bnb-r :: ‹'st ⇒ 'st ⇒ bool› for S :: 'st where
  cdcl-conflict: ‹conflict S S' ⟹ cdcl-dpll-bnb-r S S'› |
```

cdcl-propagate: ‹*propagate S S′* ⟹ *cdcl-dpll-bnb-r S S′*› |
cdcl-improve: ‹*enc-weight-opt.improvep S S′* ⟹ *cdcl-dpll-bnb-r S S′*› |
cdcl-conflict-opt0: ‹*conflict-opt0 S S′* ⟹ *cdcl-dpll-bnb-r S S′*› |
cdcl-simple-backtrack-conflict-opt:
‹*simple-backtrack-conflict-opt S S′* ⟹ *cdcl-dpll-bnb-r S S′*› |
cdcl-o′: ‹$ocdcl_W$*-o-r S S′* ⟹ *cdcl-dpll-bnb-r S S′*›

inductive *cdcl-dpll-bnb-r-stgy* :: ‹*′st* ⇒ *′st* ⇒ *bool*› **for** *S* :: *′st* **where**
cdcl-dpll-bnb-r-conflict: ‹*conflict S S′* ⟹ *cdcl-dpll-bnb-r-stgy S S′*› |
cdcl-dpll-bnb-r-propagate: ‹*propagate S S′* ⟹ *cdcl-dpll-bnb-r-stgy S S′*› |
cdcl-dpll-bnb-r-improve: ‹*enc-weight-opt.improvep S S′* ⟹ *cdcl-dpll-bnb-r-stgy S S′*› |
cdcl-dpll-bnb-r-conflict-opt0: ‹*conflict-opt0 S S′* ⟹ *cdcl-dpll-bnb-r-stgy S S′*› |
cdcl-dpll-bnb-r-simple-backtrack-conflict-opt:
‹*simple-backtrack-conflict-opt S S′* ⟹ *cdcl-dpll-bnb-r-stgy S S′*› |
cdcl-dpll-bnb-r-other′: ‹$ocdcl_W$*-o-r S S′* ⟹ *no-confl-prop-impr S* ⟹ *cdcl-dpll-bnb-r-stgy S S′*›

lemma *no-dup-dropI*:
‹*no-dup M* ⟹ *no-dup* (*drop n M*)›
by (*cases* ‹*n* < *length M*›) (*auto simp*: *no-dup-def drop-map*[*symmetric*])

lemma *tranclp-resolve-state-eq-compatible*:
‹$resolve^{++}$ *S T* ⟹*T* ∼ *T′* ⟹ $resolve^{++}$ *S T′*›
apply (*induction arbitrary*: *T′ rule*: *tranclp-induct*)
apply (*auto dest*: *resolve-state-eq-compatible*)
by (*metis resolve-state-eq-compatible state-eq-ref tranclp-into-rtranclp tranclp-unfold-end*)

lemma *conflict-opt0-state-eq-compatible*:
‹*conflict-opt0 S T* ⟹ *S* ∼ *S′* ⟹ *T* ∼ *T′* ⟹ *conflict-opt0 S′ T′*›
using *state-eq-trans*[*of T′ T*
‹*update-conflicting* (*Some* {#}) (*reduce-trail-to* ([]::(*′v*,*′v clause*) *ann-lits*) *S*)›]
using *state-eq-trans*[*of T*
‹*update-conflicting* (*Some* {#}) (*reduce-trail-to* ([]::(*′v*,*′v clause*) *ann-lits*) *S*)›
‹*update-conflicting* (*Some* {#}) (*reduce-trail-to* ([]::(*′v*,*′v clause*) *ann-lits*) *S′*)›]
update-conflicting-state-eq[*of S S′* ‹*Some* {#}›]
apply (*auto simp*: *conflict-opt0.simps state-eq-sym*)
using *reduce-trail-to-state-eq state-eq-trans update-conflicting-state-eq* **by** *blast*

lemma *conflict-opt0-conflict-opt*:
assumes ‹*conflict-opt0 S U*› **and**
inv: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*)›
shows ‹∃ *T*. *enc-weight-opt.conflict-opt S T* ∧ $resolve^{**}$ *T U*›
proof −
have
1: ‹*count-decided* (*trail S*) = *0*› **and**
neg: ‹*negate-ann-lits* (*trail S*) ∈# *enc-weight-opt.conflicting-clss S*› **and**
confl: ‹*conflicting S* = *None*› **and**
U: ‹*U* ∼ *update-conflicting* (*Some* {#}) (*reduce-trail-to* ([]::(*′v*,*′v clause*)*ann-lits*) *S*)›
using *assms* **by** (*auto elim*: *conflict-opt0E*)
let *?T* = ‹*update-conflicting* (*Some* (*negate-ann-lits* (*trail S*))) *S*›
have *confl*: ‹*enc-weight-opt.conflict-opt S ?T*›
using *neg confl*
by (*auto simp*: *enc-weight-opt.conflict-opt.simps*)
let *?T* = ‹λ*n*. *update-conflicting*
(*Some* (*negate-ann-lits* (*drop n* (*trail S*))))
(*reduce-trail-to* (*drop n* (*trail S*)) *S*)›

have *proped-M2*: ‹*is-proped* (*trail S* ! *n*)› **if** ‹*n* < *length* (*trail S*)› **for** *n*
using *1 that* **by** (*auto simp*: *count-decided-0-iff is-decided-no-proped-iff*)
have *n-d*: ‹*no-dup* (*trail S*)› **and**
le: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-conflicting* (*enc-weight-opt.abs-state S*)› **and**
dist: ‹$cdcl_W$*-restart-mset.distinct-*$cdcl_W$*-state* (*enc-weight-opt.abs-state S*)› **and**
decomp-imp: ‹*all-decomposition-implies-m* (*clauses S* + (*enc-weight-opt.conflicting-clss S*))
(*get-all-ann-decomposition* (*trail S*))› **and**
learned: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause* (*enc-weight-opt.abs-state S*)›
using *inv*
unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv-def*
by *auto*
have *proped*: ‹$\bigwedge$*L mark a b*.
a @ *Propagated L mark* # *b* = *trail S* ⟶
b ⊨*as CNot* (*remove1-mset L mark*) ∧ *L* ∈# *mark*›
using *le*
unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-conflicting-def*
by *auto*
have [*simp*]: ‹*count-decided* (*drop n* (*trail S*)) = *0*› **for** *n*
using *1* **unfolding** *count-decided-0-iff*
by (*cases* ‹*n* < *length* (*trail S*)›) (*auto dest*: *in-set-dropD*)
have [*simp*]: ‹*get-maximum-level* (*drop n* (*trail S*)) *C* = *0*› **for** *n C*
using *count-decided-ge-get-maximum-level*[*of* ‹*drop n* (*trail S*)› *C*]
by *auto*
have *mark-dist*: ‹*distinct-mset* (*mark-of* (*trail S*!*n*))› **if** ‹*n* < *length* (*trail S*)› **for** *n*
using *dist that proped-M2*[*OF that*] *nth-mem*[*OF that*]
unfolding $cdcl_W$*-restart-mset.distinct-*$cdcl_W$*-state-def*
by (*cases* ‹*trail S*!*n*›) *auto*

have *res*: ‹*resolve* (*?T n*) (*?T* (*Suc n*))› **if** ‹*n* < *length* (*trail S*)› **for** *n*
proof −
define *L* **and** *E* **where**
‹*L* = *lit-of* (*trail S* ! *n*)› **and**
‹*E* = *mark-of* (*trail S* ! *n*)›
have ‹*hd* (*drop n* (*trail S*)) = *Propagated L E*› **and**
tr-Sn: ‹*trail S* ! *n* = *Propagated L E*›
using *proped-M2*[*OF that*]
by (*cases* ‹*trail S* ! *n*›; *auto simp*: *that hd-drop-conv-nth L-def E-def*; *fail*)+
have ‹*L* ∈# *E*› **and**
ent-E: ‹*drop* (*Suc n*) (*trail S*) ⊨*as CNot* (*remove1-mset L E*)›
using *proped*[*of* ‹*take n* (*trail S*)› *L E* ‹*drop* (*Suc n*) (*trail S*)›]
that **unfolding** *tr-Sn*[*symmetric*]
by (*auto simp*: *Cons-nth-drop-Suc*)
have *1*: ‹*negate-ann-lits* (*drop* (*Suc n*) (*trail S*)) =
(*remove1-mset* (− *L*) (*negate-ann-lits* (*drop n* (*trail S*))) ∪#
remove1-mset L E)›
apply (*subst distinct-set-mset-eq-iff*[*symmetric*])
subgoal
using *n-d* **by** (*auto simp*: *no-dup-dropI*)
subgoal
using *n-d mark-dist*[*OF that*] **unfolding** *tr-Sn*
by (*auto intro*: *distinct-mset-mono no-dup-dropI*
intro!: *distinct-mset-minus*)
subgoal
using *ent-E* **unfolding** *tr-Sn*[*symmetric*]

by (*auto simp*: *negate-ann-lits-def that*
Cons-nth-drop-Suc[*symmetric*] *L-def lits-of-def*
true-annots-true-cls-def-iff-negation-in-model
uminus-lit-swap
dest!: *multi-member-split*)
done
have ‹*update-conflicting* (*Some* (*negate-ann-lits* (*drop* (*Suc n*) (*trail S*))))
(*reduce-trail-to* (*drop* (*Suc n*) (*trail S*)) *S*) ∼
update-conflicting
(*Some*
(*remove1-mset* (− *L*) (*negate-ann-lits* (*drop n* (*trail S*))) ∪#
remove1-mset L E))
(*tl-trail*
(*update-conflicting* (*Some* (*negate-ann-lits* (*drop n* (*trail S*))))
(*reduce-trail-to* (*drop n* (*trail S*)) *S*)))›
unfolding *1*[*symmetric*]
apply (*rule state-eq-trans*)
prefer *2*
apply (*rule state-eq-sym*[*THEN iffD1*])
apply (*rule update-conflicting-state-eq*)
apply (*rule tl-trail-update-conflicting*)
apply (*rule state-eq-trans*)
prefer *2*
apply (*rule state-eq-sym*[*THEN iffD1*])
apply (*rule update-conflicting-update-conflicting*)
apply (*rule state-eq-ref*)
apply (*rule update-conflicting-state-eq*)
using *that*
by (*auto simp*: *reduce-trail-to-compow-tl-trail funpow-swap1*)
moreover have ‹*L* ∈# *E*›
using *proped*[*of* ‹*take n* (*trail S*)› *L E* ‹*drop* (*Suc n*) (*trail S*)›]
that **unfolding** *tr-Sn*[*symmetric*]
by (*auto simp*: *Cons-nth-drop-Suc*)
moreover have ‹− *L* ∈# *negate-ann-lits* (*drop n* (*trail S*))›
by (*auto simp*: *negate-ann-lits-def L-def*
in-set-dropI that)
term ‹*get-maximum-level* (*drop n* (*trail S*))›
ultimately show *?thesis* **apply** −
by (*rule resolve.intros*[*of - L E*])
(*use that* **in** ‹*auto simp*: *trail-reduce-trail-to-drop*
‹*hd* (*drop n* (*trail S*)) = *Propagated L E*››)
qed
have ‹$resolve^{**}$ (*?T 0*) (*?T n*)› **if** ‹*n* ≤ *length* (*trail S*)› **for** *n*
using *that*
apply (*induction n*)
subgoal by *auto*
subgoal for *n*
using *res*[*of n*] **by** *auto*
done
from *this*[*of* ‹*length* (*trail S*)›] **have** ‹$resolve^{**}$ (*?T 0*) (*?T* (*length* (*trail S*)))›
by *auto*
moreover have ‹*?T* (*length* (*trail S*)) ∼ *U*›
apply (*rule state-eq-trans*)
prefer *2* **apply** (*rule state-eq-sym*[*THEN iffD1*], *rule U*)
by *auto*
moreover have *False* **if** ‹(*?T 0*) = (*?T* (*length* (*trail S*)))› **and** ‹*length* (*trail S*) > *0*›

using *arg-cong*[*OF that*(*1*), *of conflicting*] *that*(*2*)
by (*auto simp*: *negate-ann-lits-def*)
ultimately have ‹*length* (*trail S*) > *0* ⟶ *resolve*** (*?T 0*) *U*›
using *tranclp-resolve-state-eq-compatible*[*of* ‹*?T 0*›
‹*?T* (*length* (*trail S*))› *U*] **by** (*subst* (*asm*) *rtranclp-unfold*) *auto*
then have *?thesis* **if** ‹*length* (*trail S*) > *0*›
using *confl that* **by** *auto*
moreover have *?thesis* **if** ‹*length* (*trail S*) = *0*›
using *that confl U*
enc-weight-opt.conflict-opt-state-eq-compatible[*of S* ‹(*update-conflicting* (*Some* {#}) *S*)› *S U*]
by (*auto simp*: *state-eq-sym*)
ultimately show *?thesis*
by *blast*
qed

lemma *backtrack-split-some-is-decided-then-snd-has-hd2*:
‹∃ *l*∈*set M*. *is-decided l* ⟹ ∃ *M′ L′ M″*. *backtrack-split M* = (*M″*, *Decided L′* # *M′*)›
by (*metis backtrack-split-snd-hd-decided backtrack-split-some-is-decided-then-snd-has-hd*
is-decided-def list.distinct(*1*) *list.sel*(*1*) *snd-conv*)

lemma *no-step-conflict-opt0-simple-backtrack-conflict-opt*:
‹*no-step conflict-opt0 S* ⟹ *no-step simple-backtrack-conflict-opt S* ⟹
no-step enc-weight-opt.conflict-opt S›
using *backtrack-split-some-is-decided-then-snd-has-hd2*[*of* ‹*trail S*›]
count-decided-0-iff[*of* ‹*trail S*›]
by (*fastforce simp*: *conflict-opt0.simps simple-backtrack-conflict-opt.simps*
enc-weight-opt.conflict-opt.simps
annotated-lit.is-decided-def)

lemma *no-step-cdcl-dpll-bnb-r-cdcl-bnb-r*:
assumes ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*)›
shows
‹*no-step cdcl-dpll-bnb-r S* ⟷ *no-step cdcl-bnb-r S*› (**is** ‹*?A* ⟷ *?B*›)
proof
assume *?A*
show *?B*
using ‹*?A*› *no-step-conflict-opt0-simple-backtrack-conflict-opt*[*of S*]
by (*auto simp*: *cdcl-bnb-r.simps*
cdcl-dpll-bnb-r.simps all-conj-distrib)
next
assume *?B*
show *?A*
using ‹*?B*› *simple-backtrack-conflict-opt-conflict-analysis*[*OF* - *assms*]
by (*auto simp*: *cdcl-bnb-r.simps cdcl-dpll-bnb-r.simps all-conj-distrib assms*
dest!: *conflict-opt0-conflict-opt*)
qed

lemma *cdcl-dpll-bnb-r-cdcl-bnb-r*:
assumes ‹*cdcl-dpll-bnb-r S T*› **and**
‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*)›
shows ‹*cdcl-bnb-r*** *S T*›
using *assms*
proof (*cases rule*: *cdcl-dpll-bnb-r.cases*)
case *cdcl-simple-backtrack-conflict-opt*
then obtain *S1 S2* **where**

‹*enc-weight-opt.conflict-opt S S1*›
‹*resolve*** *S1 S2*› **and**
‹*enc-weight-opt.obacktrack S2 T*›
using *simple-backtrack-conflict-opt-conflict-analysis*[*OF - assms*(*2*), *of T*]
by *auto*
then have ‹*cdcl-bnb-r S S1*›
‹*cdcl-bnb-r*** *S1 S2*›
‹*cdcl-bnb-r S2 T*›
using *mono-rtranclp*[*of resolve enc-weight-opt.cdcl-bnb-bj*]
mono-rtranclp[*of enc-weight-opt.cdcl-bnb-bj* $ocdcl_W$*-o-r*]
mono-rtranclp[*of* $ocdcl_W$*-o-r cdcl-bnb-r*]
$ocdcl_W$*-o-r.intros enc-weight-opt.cdcl-bnb-bj.resolve*
cdcl-bnb-r.intros
enc-weight-opt.cdcl-bnb-bj.intros
by (*auto 5 4 dest*: *cdcl-bnb-r.intros conflict-opt0-conflict-opt*)
then show *?thesis*
by *auto*
next
case *cdcl-conflict-opt0*
then obtain *S1* **where**
‹*enc-weight-opt.conflict-opt S S1*›
‹*resolve*** *S1 T*›
using *conflict-opt0-conflict-opt*[*OF - assms*(*2*), *of T*]
by *auto*
then have ‹*cdcl-bnb-r S S1*›
‹*cdcl-bnb-r*** *S1 T*›
using *mono-rtranclp*[*of resolve enc-weight-opt.cdcl-bnb-bj*]
mono-rtranclp[*of enc-weight-opt.cdcl-bnb-bj* $ocdcl_W$*-o-r*]
mono-rtranclp[*of* $ocdcl_W$*-o-r cdcl-bnb-r*]
$ocdcl_W$*-o-r.intros enc-weight-opt.cdcl-bnb-bj.resolve*
cdcl-bnb-r.intros
enc-weight-opt.cdcl-bnb-bj.intros
by (*auto 5 4 dest*: *cdcl-bnb-r.intros conflict-opt0-conflict-opt*)
then show *?thesis*
by *auto*
qed (*auto 5 4 dest*: *cdcl-bnb-r.intros conflict-opt0-conflict-opt simp*: *assms*)

lemma *resolve-no-prop-confl*: ‹*resolve S T* ⟹ *no-step propagate S* ∧ *no-step conflict S*›
by (*auto elim*!: *rulesE*)

lemma *cdcl-bnb-r-stgy-res*:
‹*resolve S T* ⟹ *cdcl-bnb-r-stgy S T*›
using *enc-weight-opt.cdcl-bnb-bj.resolve*[*of S T*]
$ocdcl_W$*-o-r.intros*[*of S T*]
cdcl-bnb-r-stgy.intros[*of S T*]
resolve-no-prop-confl[*of S T*]
by (*auto 5 4 dest*: *cdcl-bnb-r-stgy.intros conflict-opt0-conflict-opt*)

lemma *rtranclp-cdcl-bnb-r-stgy-res*:
‹*resolve*** *S T* ⟹ *cdcl-bnb-r-stgy*** *S T*›
using *mono-rtranclp*[*of resolve cdcl-bnb-r-stgy*]
cdcl-bnb-r-stgy-res
by (*auto*)

lemma *obacktrack-no-prop-confl*: ‹*enc-weight-opt.obacktrack S T* ⟹ *no-step propagate S* ∧ *no-step conflict S*›

by (*auto elim*!: *rulesE enc-weight-opt.obacktrackE*)

lemma *cdcl-bnb-r-stgy-bt*:
‹*enc-weight-opt.obacktrack S T* $\Longrightarrow$ *cdcl-bnb-r-stgy S T*›
using *enc-weight-opt.cdcl-bnb-bj.backtrack*[*of S T*]
$ocdcl_W$*-o-r.intros*[*of S T*]
cdcl-bnb-r-stgy.intros[*of S T*]
obacktrack-no-prop-confl[*of S T*]
by (*auto 5 4 dest*: *cdcl-bnb-r-stgy.intros conflict-opt0-conflict-opt*)

lemma *cdcl-dpll-bnb-r-stgy-cdcl-bnb-r-stgy*:
assumes ‹*cdcl-dpll-bnb-r-stgy S T*› **and**
‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*)›
shows ‹*cdcl-bnb-r-stgy*** *S T*›
using *assms*
proof (*cases rule*: *cdcl-dpll-bnb-r-stgy.cases*)
case *cdcl-dpll-bnb-r-simple-backtrack-conflict-opt*
then obtain *S1 S2* **where**
‹*enc-weight-opt.conflict-opt S S1*›
‹*resolve*** *S1 S2*› **and**
‹*enc-weight-opt.obacktrack S2 T*›
using *simple-backtrack-conflict-opt-conflict-analysis*[*OF - assms*(*2*), *of T*]
by *auto*
then have ‹*cdcl-bnb-r-stgy S S1*›
‹*cdcl-bnb-r-stgy*** *S1 S2*›
‹*cdcl-bnb-r-stgy S2 T*›
using *enc-weight-opt.cdcl-bnb-bj.resolve*
by (*auto dest*: *cdcl-bnb-r-stgy.intros conflict-opt0-conflict-opt*
rtranclp-cdcl-bnb-r-stgy-res cdcl-bnb-r-stgy-bt)
then show *?thesis*
by *auto*
next
case *cdcl-dpll-bnb-r-conflict-opt0*
then obtain *S1* **where**
‹*enc-weight-opt.conflict-opt S S1*›
‹*resolve*** *S1 T*›
using *conflict-opt0-conflict-opt*[*OF - assms*(*2*), *of T*]
by *auto*
then have ‹*cdcl-bnb-r-stgy S S1*›
‹*cdcl-bnb-r-stgy*** *S1 T*›
using *enc-weight-opt.cdcl-bnb-bj.resolve*
by (*auto dest*: *cdcl-bnb-r-stgy.intros conflict-opt0-conflict-opt*
rtranclp-cdcl-bnb-r-stgy-res cdcl-bnb-r-stgy-bt)
then show *?thesis*
by *auto*
qed (*auto 5 4 dest*: *cdcl-bnb-r-stgy.intros conflict-opt0-conflict-opt*)

lemma *cdcl-bnb-r-stgy-cdcl-bnb-r*:
‹*cdcl-bnb-r-stgy S T* $\Longrightarrow$ *cdcl-bnb-r S T*›
by (*auto simp*: *cdcl-bnb-r-stgy.simps cdcl-bnb-r.simps*)

lemma *rtranclp-cdcl-bnb-r-stgy-cdcl-bnb-r*:
‹*cdcl-bnb-r-stgy*** *S T* $\Longrightarrow$ *cdcl-bnb-r*** *S T*›
by (*induction rule*: *rtranclp-induct*)
(*auto dest*: *cdcl-bnb-r-stgy-cdcl-bnb-r*)

context
fixes *S* :: *'st*
assumes *S*-Σ: ‹*atms-of-mm* (*init-clss S*) = Σ − ΔΣ ∪ *replacement-pos* ‘ ΔΣ ∪ *replacement-neg* ‘ ΔΣ›
begin
lemma *cdcl-dpll-bnb-r-stgy-all-struct-inv*:
‹*cdcl-dpll-bnb-r-stgy S T* ⟹
$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*) ⟹
$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state T*)›
using *cdcl-dpll-bnb-r-stgy-cdcl-bnb-r-stgy*[*of S T*]
rtranclp-cdcl-bnb-r-all-struct-inv[*OF S*-Σ]
rtranclp-cdcl-bnb-r-stgy-cdcl-bnb-r[*of S T*]
by *auto*

end

lemma *cdcl-bnb-r-stgy-cdcl-dpll-bnb-r-stgy*:
‹*cdcl-bnb-r-stgy S T* ⟹ ∃ *T*. *cdcl-dpll-bnb-r-stgy S T*›
by (*meson cdcl-bnb-r-stgy.simps cdcl-dpll-bnb-r-conflict cdcl-dpll-bnb-r-conflict-opt0*
cdcl-dpll-bnb-r-other′ cdcl-dpll-bnb-r-propagate cdcl-dpll-bnb-r-simple-backtrack-conflict-opt
cdcl-dpll-bnb-r-stgy.intros(*3*) *no-step-conflict-opt0-simple-backtrack-conflict-opt*)

context
fixes *S* :: *'st*
assumes *S*-Σ: ‹*atms-of-mm* (*init-clss S*) = Σ − ΔΣ ∪ *replacement-pos* ‘ ΔΣ ∪ *replacement-neg* ‘ ΔΣ›
begin

lemma *rtranclp-cdcl-dpll-bnb-r-stgy-cdcl-bnb-r*:
assumes ‹*cdcl-dpll-bnb-r-stgy*** *S T*› **and**
‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*)›
shows ‹*cdcl-bnb-r-stgy*** *S T*›
using *assms*
apply (*induction rule*: *rtranclp-induct*)
subgoal by *auto*
subgoal for *T U*
using *cdcl-dpll-bnb-r-stgy-cdcl-bnb-r-stgy*[*of T U*]
rtranclp-cdcl-bnb-r-all-struct-inv[*OF S*-Σ, *of T*]
rtranclp-cdcl-bnb-r-stgy-cdcl-bnb-r[*of S T*]
by *auto*
done

lemma *rtranclp-cdcl-dpll-bnb-r-stgy-all-struct-inv*:
‹*cdcl-dpll-bnb-r-stgy*** *S T* ⟹
$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*) ⟹
$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state T*)›
using *rtranclp-cdcl-dpll-bnb-r-stgy-cdcl-bnb-r*[*of T*]
rtranclp-cdcl-bnb-r-all-struct-inv[*OF S*-Σ, *of T*]
rtranclp-cdcl-bnb-r-stgy-cdcl-bnb-r[*of S T*]
by *auto*

lemma *full-cdcl-dpll-bnb-r-stgy-full-cdcl-bnb-r-stgy*:
assumes ‹*full cdcl-dpll-bnb-r-stgy S T*› **and**
‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*)›
shows ‹*full cdcl-bnb-r-stgy S T*›
using *no-step-cdcl-dpll-bnb-r-cdcl-bnb-r*[*of T*]
rtranclp-cdcl-dpll-bnb-r-stgy-cdcl-bnb-r[*of T*]
rtranclp-cdcl-dpll-bnb-r-stgy-all-struct-inv[*of T*] *assms*

rtranclp-cdcl-bnb-r-stgy-cdcl-bnb-r[*of S T*]
by (*auto simp*: *full-def*
dest: *cdcl-bnb-r-stgy-cdcl-bnb-r cdcl-bnb-r-stgy-cdcl-dpll-bnb-r-stgy*)

end

lemma *replace-pos-neg-not-both-decided-highest-lvl*:
assumes
struct: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*)› **and**
smaller-propa: ‹*no-smaller-propa S*› **and**
smaller-confl: ‹*no-smaller-confl S*› **and**
dec0: ‹*Pos* ($A^{\mapsto 0}$) ∈ *lits-of-l* (*trail S*)› **and**
dec1: ‹*Pos* ($A^{\mapsto 1}$) ∈ *lits-of-l* (*trail S*)› **and**
add: ‹*additional-constraints* ⊆# *init-clss S*› **and**
[*simp*]: ‹*A* ∈ ΔΣ›
shows ‹*get-level* (*trail S*) (*Pos* ($A^{\mapsto 0}$)) = *backtrack-lvl S* ∧
get-level (*trail S*) (*Pos* ($A^{\mapsto 1}$)) = *backtrack-lvl S*›
proof (*rule ccontr*)
assume *neg*: ‹¬ *?thesis*›
let *?L0* = ‹*get-level* (*trail S*) (*Pos* ($A^{\mapsto 0}$))›
let *?L1* = ‹*get-level* (*trail S*) (*Pos* ($A^{\mapsto 1}$))›
define *KL* **where** ‹*KL* = (*if ?L0* > *?L1 then* (*Pos* ($A^{\mapsto 1}$)) *else* (*Pos* ($A^{\mapsto 0}$)))›
define *KL'* **where** ‹*KL'* = (*if ?L0* > *?L1 then* (*Pos* ($A^{\mapsto 0}$)) *else* (*Pos* ($A^{\mapsto 1}$)))›
then have ‹*get-level* (*trail S*) *KL* < *backtrack-lvl S*› **and**
le: ‹*?L0* < *backtrack-lvl S* ∨ *?L1* < *backtrack-lvl S*›
‹*?L0* ≤ *backtrack-lvl S* ∧ *?L1* ≤ *backtrack-lvl S*›
using *neg count-decided-ge-get-level*[*of* ‹*trail S*› ‹*Pos* ($A^{\mapsto 0}$)›]
count-decided-ge-get-level[*of* ‹*trail S*› ‹*Pos* ($A^{\mapsto 1}$)›]
unfolding *KL-def*
by *force+*

have ‹*KL* ∈ *lits-of-l* (*trail S*)›
using *dec1 dec0* **by** (*auto simp*: *KL-def*)
have *add*: ‹*additional-constraint A* ⊆# *init-clss S*›
using *add multi-member-split*[*of A* ‹*mset-set* ΔΣ›] **by** (*auto simp*: *additional-constraints-def*
subset-mset.dual-order.trans)
have *n-d*: ‹*no-dup* (*trail S*)›
using *struct* **unfolding** $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv-def*
by *auto*
have *H*: ‹⋀*M K M' D L*.
trail S = *M'* @ *Decided K* # *M* ⟹
D + {#*L*#} ∈# *additional-constraint A* ⟹ *undefined-lit M L* ⟹ ¬ *M* ⊨*as CNot D*› **and**
H': ‹⋀*M K M' D L*.
trail S = *M'* @ *Decided K* # *M* ⟹
D ∈# *additional-constraint A* ⟹ ¬ *M* ⊨*as CNot D*›
using *smaller-propa add smaller-confl* **unfolding** *no-smaller-propa-def no-smaller-confl-def clauses-def*
by *auto*

have *L1-L0*: ‹*?L1* = *?L0*›
proof (*rule ccontr*)
assume *neq*: ‹*?L1* ≠ *?L0*›
define *i* **where** ‹*i* ≡ *min ?L1 ?L0*›
obtain *K M1 M2* **where**
decomp: ‹(*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail S*))› **and**

‹get-level (trail S) K = Suc i›
using backtrack-ex-decomp[OF n-d, of i] neq le
by (cases ‹?L1 < ?L0›) (auto simp: min-def i-def)
have ‹get-level (trail S) KL ≤ i› **and** ‹get-level (trail S) KL' > i›
using neq neq le **by** (auto simp: KL-def KL'-def i-def)
then have ‹undefined-lit M1 KL'›
using n-d decomp ‹get-level (trail S) K = Suc i›
count-decided-ge-get-level[of ‹M1› KL']
by (force dest!: get-all-ann-decomposition-exists-prepend
simp: get-level-append-if get-level-cons-if atm-of-eq-atm-of
dest: defined-lit-no-dupD
split: if-splits)
moreover have ‹{#−KL', −KL#} ∈# additional-constraint A›
using neq **by** (auto simp: additional-constraint-def KL-def KL'-def)
moreover have ‹KL ∈ lits-of-l M1›
using ‹get-level (trail S) KL ≤ i› ‹get-level (trail S) K = Suc i›
n-d decomp ‹KL ∈ lits-of-l (trail S)›
count-decided-ge-get-level[of ‹M1› KL]
by (auto dest!: get-all-ann-decomposition-exists-prepend
simp: get-level-append-if get-level-cons-if atm-of-eq-atm-of
dest: defined-lit-no-dupD in-lits-of-l-defined-litD
split: if-splits)
ultimately show False
using H[of - K M1 ‹{#−KL#}› ‹−KL'›] decomp
by force
qed

obtain K M1 M2 **where**
decomp: ‹(Decided K # M1, M2) ∈ set (get-all-ann-decomposition (trail S))› **and**
lev-K: ‹get-level (trail S) K = Suc ?L1›
using backtrack-ex-decomp[OF n-d, of ?L1] le
by (cases ‹?L1 < ?L0›) (auto simp: min-def L1-L0)
then obtain M3 **where**
M3: ‹trail S = M3 @ Decided K # M1›
by auto
then have [simp]: ‹undefined-lit M3 (Pos ($A^{\mapsto 1}$))› ‹undefined-lit M3 (Pos ($A^{\mapsto 0}$))›
by (solves ‹use n-d L1-L0 lev-K M3 in auto›)
(solves ‹use n-d L1-L0[symmetric] lev-K M3 in auto›)
then have [simp]: ‹Pos ($A^{\mapsto 0}$) ∉ lits-of-l M3› ‹Pos ($A^{\mapsto 1}$) ∉ lits-of-l M3›
using Decided-Propagated-in-iff-in-lits-of-l **by** blast+
have ‹Pos ($A^{\mapsto 1}$) ∈ lits-of-l M1› ‹Pos ($A^{\mapsto 0}$) ∈ lits-of-l M1›
using n-d L1-L0 lev-K dec0 dec1 Decided-Propagated-in-iff-in-lits-of-l
by (auto dest!: get-all-ann-decomposition-exists-prepend
simp: M3 get-level-cons-if
split: if-splits)
then show False
using H'[of M3 K M1 ‹{#Neg ($A^{\mapsto 0}$), Neg ($A^{\mapsto 1}$)#}›]
by (auto simp: additional-constraint-def M3)
qed

lemma cdcl-dpll-bnb-r-stgy-clauses-mono:
‹cdcl-dpll-bnb-r-stgy S T ⟹ clauses S ⊆# clauses T›
by (cases rule: cdcl-dpll-bnb-r-stgy.cases, assumption)
(auto elim!: rulesE obacktrackE enc-weight-opt.improveE
conflict-opt0E simple-backtrack-conflict-optE odecideE

enc-weight-opt.obacktrackE
simp: $ocdcl_W$*-o-r.simps enc-weight-opt.cdcl-bnb-bj.simps*)

lemma *rtranclp-cdcl-dpll-bnb-r-stgy-clauses-mono*:
‹*cdcl-dpll-bnb-r-stgy*** *S T* ⟹ *clauses S* ⊆# *clauses T*›
by (*induction rule*: *rtranclp-induct*) (*auto dest*!: *cdcl-dpll-bnb-r-stgy-clauses-mono*)

lemma *cdcl-dpll-bnb-r-stgy-init-clss-eq*:
‹*cdcl-dpll-bnb-r-stgy S T* ⟹ *init-clss S* = *init-clss T*›
by (*cases rule*: *cdcl-dpll-bnb-r-stgy.cases*, *assumption*)
(*auto elim*!: *rulesE obacktrackE enc-weight-opt.improveE*
conflict-opt0E simple-backtrack-conflict-optE odecideE
enc-weight-opt.obacktrackE
simp: $ocdcl_W$*-o-r.simps enc-weight-opt.cdcl-bnb-bj.simps*)

lemma *rtranclp-cdcl-dpll-bnb-r-stgy-init-clss-eq*:
‹*cdcl-dpll-bnb-r-stgy*** *S T* ⟹ *init-clss S* = *init-clss T*›
by (*induction rule*: *rtranclp-induct*) (*auto dest*!: *cdcl-dpll-bnb-r-stgy-init-clss-eq*)

context
fixes *S* :: *'st* **and** *N* :: ‹*'v clauses*›
assumes *S-Σ*: ‹*init-clss S* = *penc N*›
begin

lemma *replacement-pos-neg-defined-same-lvl*:
assumes
struct: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*)› **and**
A: ‹*A* ∈ ΔΣ› **and**
lev: ‹*get-level* (*trail S*) (*Pos* (*replacement-pos A*)) < *backtrack-lvl S*› **and**
smaller-propa: ‹*no-smaller-propa S*› **and**
smaller-confl: ‹*cdcl-bnb-stgy-inv S*›
shows
‹*Pos* (*replacement-pos A*) ∈ *lits-of-l* (*trail S*) ⟹
Neg (*replacement-neg A*) ∈ *lits-of-l* (*trail S*)›
proof −
have *n-d*: ‹*no-dup* (*trail S*)›
using *struct*
unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv-def*
by *auto*
have *H*: ‹⋀*M K M' D L*.
trail S = *M'* @ *Decided K* # *M* ⟹
D + {#*L*#} ∈# *additional-constraint A* ⟹ *undefined-lit M L* ⟹ ¬ *M* ⊨*as CNot D*› **and**
H': ‹⋀*M K M' D L*.
trail S = *M'* @ *Decided K* # *M* ⟹
D ∈# *additional-constraint A* ⟹ ¬ *M* ⊨*as CNot D*›
using *smaller-propa S-Σ A smaller-confl* **unfolding** *no-smaller-propa-def clauses-def penc-def*
additional-constraints-def cdcl-bnb-stgy-inv-def no-smaller-confl-def **by** *fastforce+*

show ‹*Neg* (*replacement-neg A*) ∈ *lits-of-l* (*trail S*)›
if *Pos*: ‹*Pos* (*replacement-pos A*) ∈ *lits-of-l* (*trail S*)›
proof −
obtain *M1 M2 K* **where**
‹*trail S* = *M2* @ *Decided K* # *M1*› **and**
‹*Pos* (*replacement-pos A*) ∈ *lits-of-l M1*›

using *lev n-d Pos* **by** (*force dest*!: *split-list elim*!: *is-decided-ex-Decided*
simp: *lits-of-def count-decided-def filter-empty-conv*)
then show ‹*Neg* (*replacement-neg A*) ∈ *lits-of-l* (*trail S*)›
using *H*[*of M2 K M1* ‹{#*Neg* (*replacement-pos A*)#}› ‹*Neg* (*replacement-neg A*)›]
H'[*of M2 K M1* ‹{#*Neg* (*replacement-pos A*), *Neg* (*replacement-neg A*)#}›]
by (*auto simp*: *additional-constraint-def Decided-Propagated-in-iff-in-lits-of-l*)
qed
qed

lemma *replacement-pos-neg-defined-same-lvl'*:
assumes
struct: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*)› **and**
A: ‹*A* ∈ ΔΣ› **and**
lev: ‹*get-level* (*trail S*) (*Pos* (*replacement-neg A*)) < *backtrack-lvl S*› **and**
smaller-propa: ‹*no-smaller-propa S*› **and**
smaller-confl: ‹*cdcl-bnb-stgy-inv S*›
shows
‹*Pos* (*replacement-neg A*) ∈ *lits-of-l* (*trail S*) ⟹
Neg (*replacement-pos A*) ∈ *lits-of-l* (*trail S*)›
proof −
have *n-d*: ‹*no-dup* (*trail S*)›
using *struct*
unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv-def*
by *auto*
have *H*: ‹⋀*M K M' D L*.
trail S = *M'* @ *Decided K* # *M* ⟹
D + {#*L*#} ∈# *additional-constraint A* ⟹ *undefined-lit M L* ⟹ ¬ *M* ⊨*as CNot D*› **and**
H': ‹⋀*M K M' D L*.
trail S = *M'* @ *Decided K* # *M* ⟹
D ∈# *additional-constraint A* ⟹ ¬ *M* ⊨*as CNot D*›
using *smaller-propa S-Σ A smaller-confl* **unfolding** *no-smaller-propa-def clauses-def penc-def*
additional-constraints-def cdcl-bnb-stgy-inv-def no-smaller-confl-def **by** *fastforce*+

show ‹*Neg* (*replacement-pos A*) ∈ *lits-of-l* (*trail S*)›
if *Pos*: ‹*Pos* (*replacement-neg A*) ∈ *lits-of-l* (*trail S*)›
proof −
obtain *M1 M2 K* **where**
‹*trail S* = *M2* @ *Decided K* # *M1*› **and**
‹*Pos* (*replacement-neg A*) ∈ *lits-of-l M1*›
using *lev n-d Pos* **by** (*force dest*!: *split-list elim*!: *is-decided-ex-Decided*
simp: *lits-of-def count-decided-def filter-empty-conv*)
then show ‹*Neg* (*replacement-pos A*) ∈ *lits-of-l* (*trail S*)›
using *H*[*of M2 K M1* ‹{#*Neg* (*replacement-neg A*)#}› ‹*Neg* (*replacement-pos A*)›]
H'[*of M2 K M1* ‹{#*Neg* (*replacement-neg A*), *Neg* (*replacement-pos A*)#}›]
by (*auto simp*: *additional-constraint-def Decided-Propagated-in-iff-in-lits-of-l*)
qed
qed

end

definition *all-new-literals* :: ‹*'v list*› **where**
‹*all-new-literals* = (*SOME xs. mset xs* = *mset-set* (*replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ))›

lemma *set-all-new-literals*[*simp*]:
‹*set all-new-literals* = (*replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ)›
using *finite*-Σ **apply** (*simp add*: *all-new-literals-def*)
apply (*metis* (*mono-tags*) *ex-mset finite-Un finite*-Σ *finite-imageI finite-set-mset-mset-set set-mset-mset someI*)
done

This function is basically resolving the clause with all the additional clauses {#*Neg* ($L^{\mapsto 1}$), *Neg* ($L^{\mapsto 0}$)#}.

fun *resolve-with-all-new-literals* :: ‹'*v clause* ⇒ '*v list* ⇒ '*v clause*› **where**
‹*resolve-with-all-new-literals C* [] = *C*› |
‹*resolve-with-all-new-literals C* (*L* # *Ls*) =
remdups-mset (*resolve-with-all-new-literals* (*if Pos L* ∈# *C then add-mset* (*Neg* (*opposite-var L*)) (*removeAll-mset* (*Pos L*) *C*) *else C*) *Ls*)›

abbreviation *normalize2* **where**
‹*normalize2 C* ≡ *resolve-with-all-new-literals C all-new-literals*›

lemma *Neg-in-normalize2*[*simp*]: ‹*Neg L* ∈# *C* ⟹ *Neg L* ∈# *resolve-with-all-new-literals C xs*›
by (*induction arbitrary*: *C rule*: *resolve-with-all-new-literals.induct*) *auto*

lemma *Pos-in-normalize2D*[*dest*]: ‹*Pos L* ∈# *resolve-with-all-new-literals C xs* ⟹ *Pos L* ∈# *C*›
by (*induction arbitrary*: *C rule*: *resolve-with-all-new-literals.induct*) (*force split*: *if-splits*)+

lemma *opposite-var-involutive*[*simp*]:
‹*L* ∈ (*replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ) ⟹ *opposite-var* (*opposite-var L*) = *L*›
by (*auto simp*: *opposite-var-def*)

lemma *Neg-in-resolve-with-all-new-literals-Pos-notin*:
‹*L* ∈ (*replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ) ⟹ *set xs* ⊆ (*replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ) ⟹
Pos (*opposite-var L*) ∉# *C* ⟹ *Neg L* ∈# *resolve-with-all-new-literals C xs* ⟷ *Neg L* ∈# *C*›
apply (*induction arbitrary*: *C rule*: *resolve-with-all-new-literals.induct*)
apply *clarsimp*+
subgoal premises *p*
using *p*(*2*−)
by (*auto simp del*: *Neg-in-normalize2 simp*: *eq-commute*[*of* - ‹*opposite-var* -›])
done

lemma *Pos-in-normalize2-Neg-notin*[*simp*]:
‹*L* ∈ (*replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ) ⟹
Pos (*opposite-var L*) ∉# *C* ⟹ *Neg L* ∈# *normalize2 C* ⟷ *Neg L* ∈# *C*›
by (*rule Neg-in-resolve-with-all-new-literals-Pos-notin*) (*auto*)

lemma *all-negation-deleted*:
‹*L* ∈ *set all-new-literals* ⟹ *Pos L* ∉# *normalize2 C*›
apply (*induction arbitrary*: *C rule*: *resolve-with-all-new-literals.induct*)
subgoal by *auto*
subgoal by (*auto split*: *if-splits*)
done

lemma *Pos-in-resolve-with-all-new-literals-iff-already-in-or-negation-in*:
‹*L* ∈ *set all-new-literals* ⟹*set xs* ⊆ (*replacement-neg* ‘ ΔΣ ∪ *replacement-pos* ‘ ΔΣ) ⟹ *Neg L* ∈# *resolve-with-all-new-literals C xs*⟹

Neg L ∈# *C* ∨ *Pos* (*opposite-var L*) ∈# *C*›
apply (*induction arbitrary*: *C rule*: *resolve-with-all-new-literals.induct*)
subgoal by *auto*
subgoal premises *p* **for** *C La Ls Ca*
using *p*
by (*auto split*: *if-splits dest*: *simp*: *Neg-in-resolve-with-all-new-literals-Pos-notin*)
done

lemma *Pos-in-normalize2-iff-already-in-or-negation-in*:
‹*L* ∈ *set all-new-literals* ⟹ *Neg L* ∈# *normalize2 C* ⟹
Neg L ∈# *C* ∨ *Pos* (*opposite-var L*) ∈# *C*›
using *Pos-in-resolve-with-all-new-literals-iff-already-in-or-negation-in*[*of L* ‹*all-new-literals*› *C*]
by *auto*

This proof makes it hard to measure progress because I currently do not see a way to distinguish between *add-mset* $(A^{\mapsto 1})$ *C* and *add-mset* $(A^{\mapsto 1})$ (*add-mset* $(A^{\mapsto 0})$ *C*).

lemma
assumes
‹*enc-weight-opt.cdcl-bnb-stgy S T*› **and**
struct: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*enc-weight-opt.abs-state S*)› **and**
dist: ‹*distinct-mset* (*normalize-clause* ‘# *learned-clss S*)› **and**
smaller-propa: ‹*no-smaller-propa S*› **and**
smaller-confl: ‹*cdcl-bnb-stgy-inv S*›
shows ‹*distinct-mset* (*remdups-mset* (*normalize2* ‘# *learned-clss T*))›
using *assms*(*1*)
proof (*cases*)
case *cdcl-bnb-conflict*
then show *?thesis* **using** *dist* **by** (*auto elim*!: *rulesE*)
next
case *cdcl-bnb-propagate*
then show *?thesis* **using** *dist* **by** (*auto elim*!: *rulesE*)
next
case *cdcl-bnb-improve*
then show *?thesis* **using** *dist* **by** (*auto elim*!: *enc-weight-opt.improveE*)
next
case *cdcl-bnb-conflict-opt*
then show *?thesis* **using** *dist* **by** (*auto elim*!: *enc-weight-opt.conflict-optE*)
next
case *cdcl-bnb-other'*
then show *?thesis*
proof *cases*
case *decide*
then show *?thesis* **using** *dist* **by** (*auto elim*!: *rulesE*)
next
case *bj*
then show *?thesis*
proof *cases*
case *skip*
then show *?thesis* **using** *dist* **by** (*auto elim*!: *rulesE*)
next
case *resolve*
then show *?thesis* **using** *dist* **by** (*auto elim*!: *rulesE*)
next
case *backtrack*
then obtain *M1 M2* :: ‹(*'v*, *'v clause*) *ann-lits*› **and** *K L* :: ‹*'v literal*› **and**
D D' :: ‹*'v clause*› **where**

```
confl: ‹conflicting S = Some (add-mset L D)› and
decomp: ‹(Decided K # M1, M2) ∈ set (get-all-ann-decomposition (trail S))› and
‹get-maximum-level (trail S) (add-mset L D') = local.backtrack-lvl S› and
‹get-level (trail S) L = local.backtrack-lvl S› and
lev-K: ‹get-level (trail S) K = Suc (get-maximum-level (trail S) D')› and
D'-D: ‹D' ⊆# D› and
‹set-mset (clauses S) ∪ set-mset (enc-weight-opt.conflicting-clss S) ⊨p
 add-mset L D'› and
T: ‹T ∼
   cons-trail (Propagated L (add-mset L D'))
    (reduce-trail-to M1
      (add-learned-cls (add-mset L D') (update-conflicting None S)))›
        by (auto simp: enc-weight-opt.obacktrack.simps)
      have
        tr-D: ‹trail S ⊨as CNot (add-mset L D)› and
        ‹distinct-mset (add-mset L D)› and
‹cdcl_W-restart-mset.cdcl_W-M-level-inv (abs-state S)› and
n-d: ‹no-dup (trail S)›
        using struct confl
 unfolding cdcl_W-restart-mset.cdcl_W-all-struct-inv-def
   cdcl_W-restart-mset.cdcl_W-conflicting-def
   cdcl_W-restart-mset.distinct-cdcl_W-state-def
   cdcl_W-restart-mset.cdcl_W-M-level-inv-def
 by auto
      have tr-D': ‹trail S ⊨as CNot (add-mset L D')›
        using D'-D tr-D
 by (auto simp: true-annots-true-cls-def-iff-negation-in-model)
      have ‹trail S ⊨as CNot D' ⟹ trail S ⊨as CNot (normalize2 D')›
        if ‹get-maximum-level (trail S) D' < backtrack-lvl S›
        for D'
 oops
 find-theorems get-level Pos Neg


end

end
theory CDCL-W-Covering-Models
  imports CDCL-W-Optimal-Model
begin
```

0.2 Covering Models

I am only interested in the extension of CDCL to find covering mdoels, not in the required subsequent extraction of the minimal covering models.

```
type-synonym 'v cov = ‹'v literal multiset multiset›

lemma true-clss-cls-in-susbsuming:
  ‹C' ⊆# C ⟹ C' ∈ N ⟹ N ⊨p C›
  by (metis subset-mset.le-iff-add true-clss-cls-in true-clss-cls-mono-r)

locale covering-models =
  fixes
    ϱ :: ‹'v ⇒ bool›
```

begin

definition *model-is-dominated* :: ‹′v literal multiset ⇒ ′v literal multiset ⇒ bool› **where**
‹*model-is-dominated M M′* ⟷
filter-mset (λ*L*. *is-pos L* ∧ ϱ (*atm-of L*)) *M* ⊆# *filter-mset* (λ*L*. *is-pos L* ∧ ϱ (*atm-of L*)) *M′*›

lemma *model-is-dominated-refl*: ‹*model-is-dominated I I*›
by (*auto simp*: *model-is-dominated-def*)

lemma *model-is-dominated-trans*:
‹*model-is-dominated I J* ⟹ *model-is-dominated J K* ⟹ *model-is-dominated I K*›
by (*auto simp*: *model-is-dominated-def*)

definition *is-dominating* :: ‹′v literal multiset multiset ⇒ ′v literal multiset ⇒ bool› **where**
‹*is-dominating* 𝓜 *I* ⟷ (∃ *M*∈#𝓜. ∃ *J*. *I* ⊆# *J* ∧ *model-is-dominated J M*)›

lemma
is-dominating-in:
‹*I* ∈# 𝓜 ⟹ *is-dominating* 𝓜 *I*› **and**
is-dominating-mono:
‹*is-dominating* 𝓜 *I* ⟹ *set-mset* 𝓜 ⊆ *set-mset* 𝓜′ ⟹ *is-dominating* 𝓜′ *I*› **and**
is-dominating-mono-model:
‹*is-dominating* 𝓜 *I* ⟹ *I′* ⊆# *I* ⟹ *is-dominating* 𝓜 *I′*›
using *multiset-filter-mono*[*of I′ I* ‹λ*L*. *is-pos L* ∧ ϱ (*atm-of L*)›]
by (*auto 5 5 simp*: *is-dominating-def model-is-dominated-def*
dest!: *multi-member-split*)

lemma *is-dominating-add-mset*:
‹*is-dominating* (*add-mset x* 𝓜) *I* ⟷
is-dominating 𝓜 *I* ∨ (∃ *J*. *I* ⊆# *J* ∧ *model-is-dominated J x*)›
by (*auto simp*: *is-dominating-def*)

definition *is-improving-int*
:: ‹(′v, ′v clause) ann-lits ⇒ (′v, ′v clause) ann-lits ⇒ ′v clauses ⇒ ′v cov ⇒ bool›
where
‹*is-improving-int M M′ N* 𝓜 ⟷
M = *M′* ∧ (∀ *I* ∈# 𝓜. ¬*model-is-dominated* (*lit-of* ‘# *mset M*) *I*) ∧
total-over-m (*lits-of-l M*) (*set-mset N*) ∧
lit-of ‘# *mset M* ∈ *simple-clss* (*atms-of-mm N*) ∧
lit-of ‘# *mset M* ∉# 𝓜 ∧
M ⊨*asm N* ∧
no-dup M›

This criteria is a bit more general than Weidenbach's version.

abbreviation *conflicting-clauses-ent* **where**
‹*conflicting-clauses-ent N* 𝓜 ≡
{#*pNeg* {#*L* ∈# *x*. ϱ (*atm-of L*)#}.
x ∈# *filter-mset* (λ*x*. *is-dominating* 𝓜 *x* ∧ *atms-of x* = *atms-of-mm N*)
(*mset-set* (*simple-clss* (*atms-of-mm N*)))#}+ *N*›

definition *conflicting-clauses*
:: ‹′v clauses ⇒ ′v cov ⇒ ′v clauses›
where
‹*conflicting-clauses N* 𝓜 =
{#*C* ∈# *mset-set* (*simple-clss* (*atms-of-mm N*)).
conflicting-clauses-ent N 𝓜 ⊨*pm C*#}›

lemma *conflicting-clauses-insert*:
  assumes ‹*M* ∈ *simple-clss* (*atms-of-mm N*)› **and** ‹*atms-of M* = *atms-of-mm N*›
  shows ‹*pNeg M* ∈# *conflicting-clauses N* (*add-mset M w*)›
  using *assms true-clss-cls-in-susbsuming*[*of* ‹*pNeg* {#*L* ∈# *M*. ϱ (*atm-of L*)#}›
    ‹*pNeg M*› ‹*set-mset* (*conflicting-clauses-ent N* (*add-mset M w*))›]
    is-dominating-in
  by (*auto simp*: *conflicting-clauses-def simple-clss-finite*
    pNeg-def image-mset-subseteq-mono)

lemma *is-dominating-in-conflicting-clauses*:
  assumes ‹*is-dominating* 𝓜 *I*› **and**
    atm: ‹*atms-of-s* (*set-mset I*) = *atms-of-mm N*› **and**
    ‹*set-mset I* ⊨*m N*› **and**
    ‹*consistent-interp* (*set-mset I*)› **and**
    ‹¬*tautology I*› **and**
    ‹*distinct-mset I*›
  shows
    ‹*pNeg I* ∈# *conflicting-clauses N* 𝓜›
proof −
  have *simpI*: ‹*I* ∈ *simple-clss* (*atms-of-mm N*)›
    using *assms* **by** (*auto simp*: *simple-clss-def atms-of-s-def atms-of-def*)
  obtain *I′ J* **where** ‹*J* ∈# 𝓜› **and** ‹*model-is-dominated I′ J*› **and** ‹*I* ⊆# *I′*›
    using *assms*(*1*) **unfolding** *is-dominating-def*
    by *auto*
  then have ‹*I* ∈ {*x* ∈ *simple-clss* (*atms-of-mm N*).
        (*is-dominating A x* ∨ (∃ *Ja*. *x* ⊆# *Ja* ∧ *model-is-dominated Ja J*)) ∧
        atms-of x = *atms-of-mm N*}›
    using *assms*(*1*) *atm*
    by (*auto simp*: *conflicting-clauses-def simple-clss-finite simpI atms-of-def*
        pNeg-mono true-clss-cls-in-susbsuming is-dominating-add-mset atms-of-s-def
      dest!: *multi-member-split*)
  then show *?thesis*
    using *assms*(*1*)
    by (*auto simp*: *conflicting-clauses-def simple-clss-finite simpI*
        pNeg-mono is-dominating-add-mset
      dest!: *multi-member-split*
      intro!: *true-clss-cls-in-susbsuming*[*of* ‹(λ*x*. *pNeg* {#*L* ∈# *x*. ϱ (*atm-of L*)#}) *I*›])
qed

end

locale *conflict-driven-clause-learning*$_W$*-covering-models* =
  conflict-driven-clause-learning$_W$
    state-eq
    state
    — functions for the state:
      — access functions:
    trail init-clss learned-clss conflicting
      — changing state:
    cons-trail tl-trail add-learned-cls remove-cls
    update-conflicting
      — get state:
    init-state +
  covering-models ϱ
  for

state-eq :: ‹′*st* ⇒ ′*st* ⇒ *bool*› (**infix** ‹∼› *50*) **and**
state :: ′*st* ⇒ (′*v*, ′*v clause*) *ann-lits* × ′*v clauses* × ′*v clauses* × ′*v clause option* ×
′*v cov* × ′*b* **and**
trail :: ‹′*st* ⇒ (′*v*, ′*v clause*) *ann-lits*› **and**
init-clss :: ‹′*st* ⇒ ′*v clauses*› **and**
learned-clss :: ‹′*st* ⇒ ′*v clauses*› **and**
conflicting :: ‹′*st* ⇒ ′*v clause option*› **and**

cons-trail :: ‹(′*v*, ′*v clause*) *ann-lit* ⇒ ′*st* ⇒ ′*st*› **and**
tl-trail :: ‹′*st* ⇒ ′*st*› **and**
add-learned-cls :: ‹′*v clause* ⇒ ′*st* ⇒ ′*st*› **and**
remove-cls :: ‹′*v clause* ⇒ ′*st* ⇒ ′*st*› **and**
update-conflicting :: ‹′*v clause option* ⇒ ′*st* ⇒ ′*st*› **and**
init-state :: ‹′*v clauses* ⇒ ′*st*› **and**
ϱ :: ‹′*v* ⇒ *bool*› +
fixes
update-additional-info :: ‹′*v cov* × ′*b* ⇒ ′*st* ⇒ ′*st*›
assumes
update-additional-info:
‹*state S* = (*M*, *N*, *U*, *C*, $\mathcal{M}$) ⟹ *state* (*update-additional-info K′ S*) = (*M*, *N*, *U*, *C*, *K′*)› **and**
weight-init-state:
‹⋀*N* :: ′*v clauses*. *fst* (*additional-info* (*init-state N*)) = {#}›
begin

definition *update-weight-information* :: ‹(′*v*, ′*v clause*) *ann-lits* ⇒ ′*st* ⇒ ′*st*› **where**
‹*update-weight-information M S* =
update-additional-info (*add-mset* (*lit-of* ‘# *mset M*) (*fst* (*additional-info S*)), *snd* (*additional-info S*)) *S*›

lemma
trail-update-additional-info[*simp*]: ‹*trail* (*update-additional-info w S*) = *trail S*› **and**
init-clss-update-additional-info[*simp*]:
‹*init-clss* (*update-additional-info w S*) = *init-clss S*› **and**
learned-clss-update-additional-info[*simp*]:
‹*learned-clss* (*update-additional-info w S*) = *learned-clss S*› **and**
backtrack-lvl-update-additional-info[*simp*]:
‹*backtrack-lvl* (*update-additional-info w S*) = *backtrack-lvl S*› **and**
conflicting-update-additional-info[*simp*]:
‹*conflicting* (*update-additional-info w S*) = *conflicting S*› **and**
clauses-update-additional-info[*simp*]:
‹*clauses* (*update-additional-info w S*) = *clauses S*›
using *update-additional-info*[*of S*] **unfolding** *clauses-def*
by (*subst* (*asm*) *state-prop*; *subst* (*asm*) *state-prop*; *auto*; *fail*)+

lemma
trail-update-weight-information[*simp*]:
‹*trail* (*update-weight-information w S*) = *trail S*› **and**
init-clss-update-weight-information[*simp*]:
‹*init-clss* (*update-weight-information w S*) = *init-clss S*› **and**
learned-clss-update-weight-information[*simp*]:
‹*learned-clss* (*update-weight-information w S*) = *learned-clss S*› **and**
backtrack-lvl-update-weight-information[*simp*]:
‹*backtrack-lvl* (*update-weight-information w S*) = *backtrack-lvl S*› **and**
conflicting-update-weight-information[*simp*]:
‹*conflicting* (*update-weight-information w S*) = *conflicting S*› **and**
clauses-update-weight-information[*simp*]:

‹*clauses* (*update-weight-information w S*) = *clauses S*›
using *update-additional-info*[*of S*] **unfolding** *update-weight-information-def* **by** *auto*

definition *covering* :: ‹*'st* ⇒ *'v cov*› **where**
‹*covering S* = *fst* (*additional-info S*)›

lemma
additional-info-update-additional-info[*simp*]:
‹*additional-info* (*update-additional-info w S*) = *w*›
unfolding *additional-info-def* **using** *update-additional-info*[*of S*]
by (*cases* ‹*state S*›; *auto*; *fail*)+

lemma
covering-cons-trail2[*simp*]: ‹*covering* (*cons-trail L S*) = *covering S*› **and**
clss-tl-trail2[*simp*]: ‹*covering* (*tl-trail S*) = *covering S*› **and**
covering-add-learned-cls-unfolded:
‹*covering* (*add-learned-cls U S*) = *covering S*›
and
covering-update-conflicting2[*simp*]: ‹*covering* (*update-conflicting D S*) = *covering S*› **and**
covering-remove-cls2[*simp*]:
‹*covering* (*remove-cls C S*) = *covering S*› **and**
covering-add-learned-cls2[*simp*]:
‹*covering* (*add-learned-cls C S*) = *covering S*› **and**
covering-update-covering-information2[*simp*]:
‹*covering* (*update-weight-information M S*) = *add-mset* (*lit-of* ‘# *mset M*) (*covering S*)›
by (*auto simp*: *update-weight-information-def covering-def*)

sublocale *conflict-driven-clause-learning*$_W$ **where**
state-eq = *state-eq* **and**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
by *unfold-locales*

sublocale *conflict-driven-clause-learning-with-adding-init-clause-bnb*$_W$*-no-state*
where
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**

init-state = *init-state* **and**
weight = *covering* **and**
update-weight-information = *update-weight-information* **and**
is-improving-int = *is-improving-int* **and**
conflicting-clauses = *conflicting-clauses*
by *unfold-locales*

lemma *state-additional-info2'*:
‹*state S* = (*trail S*, *init-clss S*, *learned-clss S*, *conflicting S*, *covering S*, *additional-info' S*)›
unfolding *additional-info'-def* **by** (*cases* ‹*state S*›; *auto simp*: *state-prop covering-def*)

lemma *state-update-weight-information*:
‹*state S* = (*M*, *N*, *U*, *C*, *w*, *other*) ⟹
∃ *w'*. *state* (*update-weight-information T S*) = (*M*, *N*, *U*, *C*, *w'*, *other*)›
unfolding *update-weight-information-def* **by** (*cases* ‹*state S*›; *auto simp*: *state-prop covering-def*)

lemma *conflicting-clss-incl-init-clss*:
‹*atms-of-mm* (*conflicting-clss S*) ⊆ *atms-of-mm* (*init-clss S*)›
unfolding *conflicting-clss-def conflicting-clauses-def*
apply (*auto simp*: *simple-clss-finite*)
by (*auto simp*: *simple-clss-def atms-of-ms-def split*: *if-splits*)

lemma *conflict-clss-update-weight-no-alien*:
‹*atms-of-mm* (*conflicting-clss* (*update-weight-information M S*))
⊆ *atms-of-mm* (*init-clss S*)›
by (*auto simp*: *conflicting-clss-def conflicting-clauses-def atms-of-ms-def*
$cdcl_W$*-restart-mset-state simple-clss-finite*
dest: *simple-clssE*)

lemma *distinct-mset-mset-conflicting-clss2*: ‹*distinct-mset-mset* (*conflicting-clss S*)›
unfolding *conflicting-clss-def conflicting-clauses-def distinct-mset-set-def*
apply (*auto simp*: *simple-clss-finite*)
by (*auto simp*: *simple-clss-def*)

lemma *total-over-m-atms-incl*:
assumes ‹*total-over-m M* (*set-mset N*)›
shows
‹*x* ∈ *atms-of-mm N* ⟹ *x* ∈ *atms-of-s M*›
by (*meson assms contra-subsetD total-over-m-alt-def*)

lemma *negate-ann-lits-simple-clss-iff* [*iff*]:
‹*negate-ann-lits M* ∈ *simple-clss N* ⟷ *lit-of* '# *mset M* ∈ *simple-clss N*›
unfolding *negate-ann-lits-def*
by (*subst uminus-simple-clss-iff* [*symmetric*]) *auto*

lemma *conflicting-clss-update-weight-information-in2*:
assumes ‹*is-improving M M' S*›
shows ‹*negate-ann-lits M'* ∈# *conflicting-clss* (*update-weight-information M' S*)›
proof −
have
[*simp*]: ‹*M'* = *M*› **and**
‹∀ *I*∈#*covering S*. ¬ *model-is-dominated* (*lit-of* '# *mset M*) *I*› **and**
tot: ‹*total-over-m* (*lits-of-l M*) (*set-mset* (*init-clss S*))› **and**

simpI: ‹*lit-of* '# *mset M* ∈ *simple-clss* (*atms-of-mm* (*init-clss S*))› **and**
‹*lit-of* '# *mset M* ∉# *covering S*› **and**
‹*no-dup M*› **and**
‹*M* ⊨*asm init-clss S*›
using *assms* **unfolding** *is-improving-int-def* **by** *auto*
have ‹*pNeg* {#*L* ∈# *lit-of* '# *mset M*. ϱ (*atm-of L*)#}
∈ (λ*x*. *pNeg* {#*L* ∈# *x*. ϱ (*atm-of L*)#}) '
{*x* ∈ *simple-clss* (*atms-of-mm* (*init-clss S*)).
is-dominating (*add-mset* (*lit-of* '# *mset M*) (*covering S*)) *x*}›
using *is-dominating-in*[*of* ‹*lit-of* '# *mset M*› ‹*add-mset* (*lit-of* '# *mset M*) (*covering S*)›]
by (*auto simp*: *simple-clss-finite multiset-filter-mono2 pNeg-mono*
conflicting-clauses-def conflicting-clss-def is-improving-int-def
simpI)
moreover have ‹*atms-of* (*lit-of* '# *mset M*) = *atms-of-mm* (*init-clss S*)›
using *tot simpI*
by (*auto simp*: *simple-clss-finite multiset-filter-mono2 pNeg-mono*
conflicting-clauses-def conflicting-clss-def is-improving-int-def
total-over-m-alt-def atms-of-s-def lits-of-def image-image atms-of-def
simple-clss-def)
ultimately have ‹(∃ *x*. *x* ∈ *simple-clss* (*atms-of-mm* (*init-clss S*)) ∧
is-dominating (*add-mset* (*lit-of* '# *mset M*) (*covering S*)) *x* ∧
atms-of x = *atms-of-mm* (*init-clss S*) ∧
pNeg {#*L* ∈# *lit-of* '# *mset M*. ϱ (*atm-of L*)#} =
pNeg {#*L* ∈# *x*. ϱ (*atm-of L*)#})›
by (*auto intro*: *exI*[*of* - ‹*lit-of* '# *mset M*›] *simp add*: *simpI is-dominating-in*)
then show *?thesis*
using *is-dominating-in*
true-clss-cls-in-susbsuming[*of* ‹*pNeg* {#*L* ∈# *lit-of* '# *mset M*. ϱ (*atm-of L*)#}›
‹*pNeg* (*lit-of* '# *mset M*)› ‹*set-mset* (*conflicting-clauses-ent*
(*init-clss S*) (*covering* (*update-weight-information M' S*)))›]
by (*auto simp*: *simple-clss-finite multiset-filter-mono2 simpI*
conflicting-clauses-def conflicting-clss-def pNeg-mono
negate-ann-lits-pNeg-lit-of image-iff image-mset-subseteq-mono)
qed

lemma *is-improving-conflicting-clss-update-weight-information*: ‹*is-improving M M' S* ⟹
conflicting-clss S ⊆# *conflicting-clss* (*update-weight-information M' S*)›
by (*auto simp*: *is-improving-int-def conflicting-clss-def conflicting-clauses-def*
simp: *multiset-filter-mono2 le-less true-clss-cls-tautology-iff simple-clss-finite*
is-dominating-add-mset filter-disj-eq image-Un
intro!: *image-mset-subseteq-mono*
intro: *true-clss-cls-subsetI*
dest: *simple-clssE*
split: *enat.splits*)

sublocale $state_W$*-no-state*
where
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**

update-conflicting = *update-conflicting* **and**
init-state = *init-state*
by *unfold-locales*

sublocale $state_W$*-no-state* **where**
state-eq = *state-eq* **and**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
by *unfold-locales*

sublocale *conflict-driven-clause-learning*$_W$ **where**
state-eq = *state-eq* **and**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
by *unfold-locales*

sublocale *conflict-driven-clause-learning-with-adding-init-clause-bnb*$_W$*-ops*
where
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state* **and**
weight = *covering* **and**
update-weight-information = *update-weight-information* **and**
is-improving-int = *is-improving-int* **and**
conflicting-clauses = *conflicting-clauses*
apply *unfold-locales*
subgoal by (*rule state-additional-info2'*)
subgoal by (*rule state-update-weight-information*)
subgoal by (*rule conflicting-clss-incl-init-clss*)
subgoal by (*rule distinct-mset-mset-conflicting-clss2*)

subgoal by (*rule is-improving-conflicting-clss-update-weight-information*)
subgoal by (*rule conflicting-clss-update-weight-information-in2*)
done

definition *covering-simple-clss* **where**
‹*covering-simple-clss N S* ⟷ (*set-mset* (*covering S*) ⊆ *simple-clss* (*atms-of-mm N*)) ∧
distinct-mset (*covering S*) ∧
(∀ *M* ∈# *covering S*. *total-over-m* (*set-mset M*) (*set-mset N*))›

lemma [*simp*]: ‹*covering* (*init-state N*) = {#}›
by (*simp add*: *covering-def weight-init-state*)

lemma ‹*covering-simple-clss N* (*init-state N*)›
by (*auto simp*: *covering-simple-clss-def*)

lemma *cdcl-bnb-covering-simple-clss*:
‹*cdcl-bnb S T* ⟹ *init-clss S* = *N* ⟹ *covering-simple-clss N S* ⟹ *covering-simple-clss N T*›
by (*auto simp*: *cdcl-bnb.simps covering-simple-clss-def is-improving-int-def*
model-is-dominated-refl $ocdcl_W$*-o.simps cdcl-bnb-bj.simps*
lits-of-def
elim!: *rulesE improveE conflict-optE obacktrackE*
dest!: *multi-member-split*[*of* - ‹*covering S*›])

lemma *rtranclp-cdcl-bnb-covering-simple-clss*:
‹$cdcl\text{-}bnb^{**}$ *S T* ⟹ *init-clss S* = *N* ⟹ *covering-simple-clss N S* ⟹ *covering-simple-clss N T*›
by (*induction rule*: *rtranclp-induct*)
(*auto simp*: *cdcl-bnb-covering-simple-clss simp*: *rtranclp-cdcl-bnb-no-more-init-clss*
cdcl-bnb-no-more-init-clss)

lemma *wf-cdcl-bnb-fixed*:
‹*wf* {(*T*, *S*). $cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv* (*abs-state S*) ∧ *cdcl-bnb S T*
∧ *covering-simple-clss N S* ∧ *init-clss S* = *N*}›
apply (*rule wf-cdcl-bnb-with-additional-inv*[*of*
‹*covering-simple-clss N*›
N id ‹{(*T*, *S*). (*T*, *S*) ∈ {($\mathcal{M}'$, $\mathcal{M}$). $\mathcal{M}$ ⊂# $\mathcal{M}'$ ∧ *distinct-mset* $\mathcal{M}'$
∧ *set-mset* $\mathcal{M}'$ ⊆ *simple-clss* (*atms-of-mm N*)}}›])
subgoal
by (*auto simp*: *improvep.simps is-improving-int-def covering-simple-clss-def*
add-mset-eq-add-mset model-is-dominated-refl
dest!: *multi-member-split*)
subgoal
apply (*rule wf-bounded-set*[*of* - ‹λ-. *simple-clss* (*atms-of-mm N*)› *set-mset*])
apply (*auto simp*: *distinct-mset-subset-iff-remdups*[*symmetric*] *simple-clss-finite*
simp flip: *remdups-mset-def*)
by (*metis distinct-mset-mono distinct-mset-set-mset-ident*)
subgoal
by (*rule cdcl-bnb-covering-simple-clss*)
done

lemma *can-always-improve*:
assumes
ent: ‹*trail S* ⊨*asm clauses S*› **and**
total: ‹*total-over-m* (*lits-of-l* (*trail S*)) (*set-mset* (*clauses S*))› **and**
n-s: ‹*no-step conflict-opt S*› **and**
confl: ‹*conflicting S* = *None*› **and**

all-struct: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state S*)›
shows ‹*Ex* (*improvep S*)›
proof −
have ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv* (*abs-state S*)› **and**
alien: ‹$cdcl_W$*-restart-mset.no-strange-atm* (*abs-state S*)›
using *all-struct*
unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*
by *fast+*
then have *n-d*: ‹*no-dup* (*trail S*)›
unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv-def*
by *auto*
have [*simp*]:
‹*atms-of-mm* (*CDCL-W-Abstract-State.init-clss* (*abs-state S*)) = *atms-of-mm* (*init-clss S*)›
unfolding *abs-state-def init-clss.simps*
by *auto*
let *?M* = ‹(*lit-of* ‘# *mset* (*trail S*))›
have *trail-simple*: ‹*?M* ∈ *simple-clss* (*atms-of-mm* (*init-clss S*))›
using *n-d alien*
by (*auto simp*: *simple-clss-def* $cdcl_W$*-restart-mset.no-strange-atm-def*
lits-of-def image-image atms-of-def
dest: *distinct-consistent-interp no-dup-not-tautology*
no-dup-distinct)
then have [*simp*]: ‹*atms-of ?M* = *atms-of-mm* (*init-clss S*)›
using *total*
by (*auto simp*: *total-over-m-alt-def simple-clss-def atms-of-def image-image*
lits-of-def atms-of-s-def clauses-def)
then have *K*: ‹*is-dominating* (*covering S*) *?M* ⟹ *pNeg* {#*L* ∈# *lit-of* ‘# *mset* (*trail S*). ϱ (*atm-of*
L)#}
∈ (λ*x*. *pNeg* {#*L* ∈# *x*. ϱ (*atm-of L*)#}) ‘
{*x* ∈ *simple-clss* (*atms-of-mm* (*init-clss S*)).
is-dominating (*covering S*) *x* ∧
atms-of x = *atms-of-mm* (*init-clss S*)}›
by (*auto simp*: *image-iff trail-simple*
intro!: *exI*[*of* - ‹*lit-of* ‘# *mset* (*trail S*)›])
have *H*: ‹*I* ∈# *covering S* ⟹
model-is-dominated ?M I ⟹
pNeg {#*L* ∈# *?M*. ϱ (*atm-of L*)#}
∈ (λ*x*. *pNeg* {#*L* ∈# *x*. ϱ (*atm-of L*)#}) ‘
{*x* ∈ *simple-clss* (*atms-of-mm* (*init-clss S*)).
is-dominating (*covering S*) *x*}› **for** *I*
using *is-dominating-in*[*of* ‹*lit-of* ‘# *mset M*› ‹*add-mset* (*lit-of* ‘# *mset M*) (*covering S*)›]
trail-simple
by (*auto 5 5 simp*: *simple-clss-finite multiset-filter-mono2 pNeg-mono*
conflicting-clauses-def conflicting-clss-def is-improving-int-def
is-dominating-add-mset filter-disj-eq image-Un
dest!: *multi-member-split*)
have ‹*I* ∈# *covering S* ⟹
model-is-dominated ?M I ⟹ *False*› **for** *I*
using *n-s confl H*[*of I*] *K*
true-clss-cls-in-susbsuming[*of* ‹*pNeg* {#*L* ∈# *?M*. ϱ (*atm-of L*)#}›
‹*pNeg ?M*› ‹*set-mset* (*conflicting-clauses-ent*
(*init-clss S*) (*covering S*))›]
by (*auto simp*: *conflict-opt.simps simple-clss-finite*
conflicting-clss-def conflicting-clauses-def is-dominating-def
is-dominating-add-mset filter-disj-eq image-Un pNeg-mono
multiset-filter-mono2 negate-ann-lits-pNeg-lit-of

```
      intro: trail-simple)
  moreover have False if ‹lit-of `# mset (trail S) ∈# covering S›
    using n-s confl that trail-simple by (auto simp: conflict-opt.simps
      conflicting-clauses-insert conflicting-clss-def simple-clss-finite
      negate-ann-lits-pNeg-lit-of
      dest!: multi-member-split)
  ultimately have imp: ‹is-improving (trail S) (trail S) S›
    unfolding is-improving-int-def
    using assms trail-simple n-d by (auto simp: clauses-def)
  show ?thesis
    by (rule exI) (rule improvep.intros[OF imp confl state-eq-ref])
qed

lemma exists-model-with-true-lit-entails-conflicting:
  assumes
    L-I: ‹Pos L ∈ I› and
    L: ‹ϱ L› and
    L-in: ‹L ∈ atms-of-mm (init-clss S)› and
    ent: ‹I ⊨m init-clss S› and
    cons: ‹consistent-interp I› and
    total: ‹total-over-m I (set-mset N)› and
    no-L: ‹¬(∃ J∈# covering S. Pos L ∈# J)› and
    cov: ‹covering-simple-clss N S› and
    NS: ‹atms-of-mm N = atms-of-mm (init-clss S)›
  shows ‹I ⊨m conflicting-clss S› and
    ‹I ⊨m CDCL-W-Abstract-State.init-clss (abs-state S)›
proof −
  show ‹I ⊨m conflicting-clss S›
    unfolding conflicting-clss-def conflicting-clauses-def
      set-mset-filter true-cls-mset-def
  proof
    fix C
    assume ‹C ∈ {a. a ∈# mset-set (simple-clss (atms-of-mm (init-clss S))) ∧
              {#pNeg {#L ∈# x. ϱ (atm-of L)#}.
              x ∈# {#x ∈# mset-set (simple-clss (atms-of-mm (init-clss S))).
                      is-dominating (covering S) x ∧
                      atms-of x = atms-of-mm (init-clss S)#}#} +
              init-clss S ⊨pm
              a}›
    then have simp-C: ‹C ∈ simple-clss (atms-of-mm (init-clss S))› and
      ent-C: ‹(λx. pNeg {#L ∈# x. ϱ (atm-of L)#}) `
           {x ∈ simple-clss (atms-of-mm (init-clss S)). is-dominating (covering S) x ∧
      atms-of x = atms-of-mm (init-clss S)} ∪
          set-mset (init-clss S) ⊨p C›
      by (auto simp: simple-clss-finite)
    have tot-I2: ‹total-over-m I
         ((λx. pNeg {#L ∈# x. ϱ (atm-of L)#}) `
          {x ∈ simple-clss (atms-of-mm (init-clss S)).
           is-dominating (covering S) x ∧
           atms-of x = atms-of-mm (init-clss S)} ∪
          set-mset (init-clss S) ∪
          {C}) ⟷ total-over-m I (set-mset N)› for I
      using simp-C  NS[symmetric]
      by (auto simp: total-over-m-def total-over-set-def
          simple-clss-def atms-of-ms-def atms-of-def pNeg-def
 dest!: multi-member-split)
```

have ‹*I* ⊨*s* (λ*x*. *pNeg* {#*L* ∈# *x*. ϱ (*atm-of L*)#}) ‘
{*x* ∈ *simple-clss* (*atms-of-mm* (*init-clss S*)). *is-dominating* (*covering S*) *x* ∧
atms-of x = *atms-of-mm* (*init-clss S*)}›
unfolding *NS*[*symmetric*]
total-over-m-alt-def true-clss-def
proof
fix *D*
assume ‹*D* ∈ (λ*x*. *pNeg* {#*L* ∈# *x*. ϱ (*atm-of L*)#}) ‘
{*x* ∈ *simple-clss* (*atms-of-mm N*). *is-dominating* (*covering S*) *x* ∧
atms-of x = *atms-of-mm N*}›
then obtain *x* **where**
D: ‹*D* = *pNeg* {#*L* ∈# *x*. ϱ (*atm-of L*)#}› **and**
x: ‹*x* ∈ *simple-clss* (*atms-of-mm N*)› **and**
dom: ‹*is-dominating* (*covering S*) *x*› **and**
tot-x: ‹*atms-of x* = *atms-of-mm N*›
by *auto*
then have ‹*L* ∈ *atms-of x*›
using *cov L-in no-L*
unfolding *NS*[*symmetric*]
by (*auto simp*: *true-clss-def is-dominating-def model-is-dominated-def*
covering-simple-clss-def atms-of-def pNeg-def image-image
total-over-m-alt-def atms-of-s-def
dest!: *multi-member-split*)
then have ‹*Neg L* ∈# *x*›
using *no-L dom L* **unfolding** *atm-iff-pos-or-neg-lit*
by (*auto simp*: *is-dominating-def model-is-dominated-def insert-subset-eq-iff*
dest!: *multi-member-split*)
then have ‹*Pos L* ∈# *D*›
using *L*
by (*auto simp*: *pNeg-def image-image D image-iff*
dest!: *multi-member-split*)
then show ‹*I* ⊨ *D*›
using *L-I* **by** (*auto dest*: *multi-member-split*)
qed
then show ‹*I* ⊨ *C*›
using *total cons ent-C ent*
unfolding *true-clss-cls-def tot-I2*
by *auto*
qed
then show *I-S*: ‹*I* ⊨*m* *CDCL-W-Abstract-State.init-clss* (*abs-state S*)›
using *ent*
by (*auto simp*: *abs-state-def init-clss.simps*)
qed

lemma *exists-model-with-true-lit-still-model*:
assumes
L-I: ‹*Pos L* ∈ *I*› **and**
L: ‹ϱ *L*› **and**
L-in: ‹*L* ∈ *atms-of-mm* (*init-clss S*)› **and**
ent: ‹*I* ⊨*m* *init-clss S*› **and**
cons: ‹*consistent-interp I*› **and**
total: ‹*total-over-m I* (*set-mset N*)› **and**
cdcl: ‹*cdcl-bnb S T*› **and**
no-L-T: ‹¬(∃ *J*∈# *covering T*. *Pos L* ∈# *J*)› **and**
cov: ‹*covering-simple-clss N S*› **and**
NS: ‹*atms-of-mm N* = *atms-of-mm* (*init-clss S*)›

shows ‹*I* ⊨*m* *CDCL-W-Abstract-State.init-clss* (*abs-state* *T*)›
proof −
have *no-L*: ‹¬(∃ *J*∈# *covering* *S*. *Pos* *L* ∈# *J*)›
using *cdcl* *no-L-T*
by (*cases*) (*auto* *elim*!: *rulesE* *improveE* *conflict-optE* *obacktrackE*
simp: $ocdcl_W$*-o.simps* *cdcl-bnb-bj.simps*)
have *I-S*: ‹*I* ⊨*m* *CDCL-W-Abstract-State.init-clss* (*abs-state* *S*)›
by (*rule* *exists-model-with-true-lit-entails-conflicting*[*OF* *assms*(*1*−*6*) *no-L* *assms*(*9*) *NS*])
have *I-T′*: ‹*I* ⊨*m* *conflicting-clss* (*update-weight-information* *M′* *S*)›
if *T*: ‹*T* ∼ *update-weight-information* *M′* *S*› **for** *M′*
unfolding *conflicting-clss-def* *conflicting-clauses-def*
set-mset-filter *true-cls-mset-def*
proof
let *?T* = ‹*update-weight-information* *M′* *S*›
fix *C*
assume ‹*C* ∈ {*a*. *a* ∈# *mset-set* (*simple-clss* (*atms-of-mm* (*init-clss* *?T*))) ∧
{#*pNeg* {#*L* ∈# *x*. ϱ (*atm-of* *L*)#}.
x ∈# {#*x* ∈# *mset-set* (*simple-clss* (*atms-of-mm* (*init-clss* *?T*))).
is-dominating (*covering* *?T*) *x* ∧
atms-of *x* = *atms-of-mm* (*init-clss* *?T*)#}#} +
init-clss *?T* ⊨*pm*
a}›
then have *simp-C*: ‹*C* ∈ *simple-clss* (*atms-of-mm* (*init-clss* *?T*))› **and**
ent-C: ‹(λ*x*. *pNeg* {#*L* ∈# *x*. ϱ (*atm-of* *L*)#}) ‘
{*x* ∈ *simple-clss* (*atms-of-mm* (*init-clss* *?T*)). *is-dominating* (*covering* *?T*) *x* ∧
atms-of *x* = *atms-of-mm* (*init-clss* *?T*)} ∪
set-mset (*init-clss* *?T*) ⊨*p* *C*›
by (*auto* *simp*: *simple-clss-finite*)
have *tot-I2*: ‹*total-over-m* *I*
((λ*x*. *pNeg* {#*L* ∈# *x*. ϱ (*atm-of* *L*)#}) ‘
{*x* ∈ *simple-clss* (*atms-of-mm* (*init-clss* *?T*)).
is-dominating (*covering* *?T*) *x* ∧
atms-of *x* = *atms-of-mm* (*init-clss* *?T*)} ∪
set-mset (*init-clss* *?T*) ∪
{*C*}) ⟷ *total-over-m* *I* (*set-mset* *N*)› **for** *I*
using *simp-C* *NS*[*symmetric*]
by (*auto* *simp*: *total-over-m-def* *total-over-set-def*
simple-clss-def *atms-of-ms-def* *atms-of-def* *pNeg-def*
dest!: *multi-member-split*)
have *H*: ‹*atms-of-mm* (*init-clss* (*update-weight-information* *M′* *S*)) = *atms-of-mm* *N*›
by (*auto* *simp*: *NS*)
have ‹*I* ⊨*s* (λ*x*. *pNeg* {#*L* ∈# *x*. ϱ (*atm-of* *L*)#}) ‘
{*x* ∈ *simple-clss* (*atms-of-mm* (*init-clss* *?T*)). *is-dominating* (*covering* *?T*) *x* ∧
atms-of *x* = *atms-of-mm* (*init-clss* *?T*)}›
unfolding *NS*[*symmetric*] *H*
total-over-m-alt-def *true-clss-def*
proof
fix *D*
assume ‹*D* ∈ (λ*x*. *pNeg* {#*L* ∈# *x*. ϱ (*atm-of* *L*)#}) ‘
{*x* ∈ *simple-clss* (*atms-of-mm* *N*). *is-dominating* (*covering* *?T*) *x* ∧
atms-of *x* = *atms-of-mm* *N*}›
then obtain *x* **where**
D: ‹*D* = *pNeg* {#*L* ∈# *x*. ϱ (*atm-of* *L*)#}› **and**
x: ‹*x* ∈ *simple-clss* (*atms-of-mm* *N*)› **and**
dom: ‹*is-dominating* (*covering* *?T*) *x*› **and**
tot-x: ‹*atms-of* *x* = *atms-of-mm* *N*›

by *auto*
then have ‹$L \in$ *atms-of x*›
using *cov L-in no-L*
unfolding *NS*[*symmetric*]
by (*auto simp*: *true-clss-def is-dominating-def model-is-dominated-def covering-simple-clss-def atms-of-def pNeg-def image-image total-over-m-alt-def atms-of-s-def dest*!: *multi-member-split*)
then have ‹*Neg L* $\in\#$ *x*›
using *no-L-T dom L T* **unfolding** *atm-iff-pos-or-neg-lit*
by (*auto simp*: *is-dominating-def model-is-dominated-def insert-subset-eq-iff dest*!: *multi-member-split*)
then have ‹*Pos L* $\in\#$ *D*›
using *L*
by (*auto simp*: *pNeg-def image-image D image-iff dest*!: *multi-member-split*)
then show ‹$I \models D$›
using *L-I* **by** (*auto dest*: *multi-member-split*)
qed
then show ‹$I \models C$›
using *total cons ent-C ent*
unfolding *true-clss-cls-def tot-I2*
by *auto*
qed
show *?thesis*
using *cdcl*
proof (*cases*)
case *cdcl-conflict*
then show *?thesis* **using** *I-S* **by** (*auto elim*!: *conflictE*)
next
case *cdcl-propagate*
then show *?thesis* **using** *I-S* **by** (*auto elim*!: *rulesE*)
next
case *cdcl-improve*
show *?thesis*
using *I-S cdcl-improve I-T'*
by (*auto simp*: *abs-state-def init-clss.simps elim*!: *improveE*)
next
case *cdcl-conflict-opt*
then show *?thesis* **using** *I-S* **by** (*auto elim*!: *conflict-optE*)
next
case *cdcl-other'*
then show *?thesis* **using** *I-S* **by** (*auto elim*!: *rulesE obacktrackE simp*: $ocdcl_W$*-o.simps cdcl-bnb-bj.simps*)
qed
qed

lemma *rtranclp-exists-model-with-true-lit-still-model*:
assumes
L-I: ‹*Pos L* $\in$ *I*› **and**
L: ‹ϱ *L*› **and**
L-in: ‹$L \in$ *atms-of-mm* (*init-clss S*)› **and**
ent: ‹$I \models m$ *init-clss S*› **and**
cons: ‹*consistent-interp I*› **and**
total: ‹*total-over-m I* (*set-mset N*)› **and**
cdcl: ‹*cdcl-bnb*** *S T*› **and**

cov: ‹*covering-simple-clss N S*› **and**
‹*N* = *init-clss S*›
shows ‹*I* ⊨*m CDCL-W-Abstract-State.init-clss* (*abs-state T*) ∨ (∃ *J*∈# *covering T*. *Pos L* ∈# *J*)›
using *cdcl assms*
apply (*induction rule*: *rtranclp-induct*)
subgoal using *exists-model-with-true-lit-entails-conflicting*[*of L I S N*]
by *auto*
subgoal for *T U*
apply (*rule disjCI*)
apply (*rule exists-model-with-true-lit-still-model*[*OF L-I L - - cons total*, *of T U*])
by (*auto dest*: *rtranclp-cdcl-bnb-no-more-init-clss*
intro: *rtranclp-cdcl-bnb-covering-simple-clss cdcl-bnb-covering-simple-clss*)
done

lemma *is-dominating-nil*[*simp*]: ‹¬*is-dominating* {#} *x*›
by (*auto simp*: *is-dominating-def*)

lemma *atms-of-conflicting-clss-init-state*:
‹*atms-of-mm* (*conflicting-clss* (*init-state N*)) ⊆ *atms-of-mm N*›
by (*auto simp*: *conflicting-clss-def conflicting-clauses-def*
atms-of-ms-def simple-clss-finite
dest!: *simple-clssE*)

lemma *no-step-cdcl-bnb-stgy-empty-conflict2*:
assumes
n-s: ‹*no-step cdcl-bnb S*› **and**
all-struct: ‹$cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv* (*abs-state S*)› **and**
stgy-inv: ‹*cdcl-bnb-stgy-inv S*›
shows ‹*conflicting S* = *Some* {#}›
by (*rule no-step-cdcl-bnb-stgy-empty-conflict*[*OF can-always-improve assms*])

theorem *cdclcm-correctness*:
assumes
full: ‹*full cdcl-bnb-stgy* (*init-state N*) *T*› **and**
dist: ‹*distinct-mset-mset N*›
shows
‹*Pos L* ∈ *I* ⟹ *ϱ L* ⟹ *L* ∈ *atms-of-mm N* ⟹ *total-over-m I* (*set-mset N*) ⟹ *consistent-interp I* ⟹ *I* ⊨*m N* ⟹
∃ *J* ∈# *covering T*. *Pos L* ∈# *J*›
proof −
let *?S* = ‹*init-state N*›
have *ns*: ‹*no-step cdcl-bnb-stgy T*› **and**
st: ‹*cdcl-bnb-stgy*** *?S T*› **and**
st': ‹*cdcl-bnb*** *?S T*›
using *full* **unfolding** *full-def* **by** (*auto intro*: *rtranclp-cdcl-bnb-stgy-cdcl-bnb*)
have *ns'*: ‹*no-step cdcl-bnb T*›
by (*meson cdcl-bnb.cases cdcl-bnb-stgy.simps no-confl-prop-impr.elims*(*3*) *ns*)

have ‹*distinct-mset C*› **if** ‹*C* ∈# *N*› **for** *C*
using *dist that* **by** (*auto simp*: *distinct-mset-set-def dest*: *multi-member-split*)
then have *dist*: ‹*distinct-mset-mset* (*N*)›
by (*auto simp*: *distinct-mset-set-def*)
then have [*simp*]: ‹$cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv* ([], *N*, {#}, *None*)›
unfolding *init-state.simps*[*symmetric*]
by (*auto simp*: $cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv-def*)

have [*iff*]: ‹*cdcl-bnb-struct-invs ?S*›
  using *atms-of-conflicting-clss-init-state*[*of N*]
  by (*auto simp*: *cdcl-bnb-struct-invs-def*)
have *stgy-inv*: ‹*cdcl-bnb-stgy-inv ?S*›
  by (*auto simp*: *cdcl-bnb-stgy-inv-def conflict-is-false-with-level-def*)
have *ent*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init* (*abs-state ?S*)›
  by (*auto simp*: $cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init-def*)
have *all-struct*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state* (*init-state N*))›
  unfolding *CDCL-W-Abstract-State.init-state.simps abs-state-def*
  by (*auto simp*: $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def dist*
    $cdcl_W$*-restart-mset.no-strange-atm-def* $cdcl_W$*-restart-mset-state*
    $cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv-def*
    $cdcl_W$*-restart-mset.distinct-*$cdcl_W$*-state-def*
    $cdcl_W$*-restart-mset.*$cdcl_W$*-conflicting-def distinct-mset-mset-conflicting-clss*
    $cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause-alt-def*)
have *cdcl*: ‹*cdcl-bnb*** *?S T*›
  using *st rtranclp-cdcl-bnb-stgy-cdcl-bnb* **unfolding** *full-def* **by** *blast*
have *cov*: ‹*covering-simple-clss N ?S*›
  by (*auto simp*: *covering-simple-clss-def*)

have *struct-T*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*abs-state T*)›
  using *rtranclp-cdcl-bnb-stgy-all-struct-inv*[*OF st' all-struct*] **.**
have *stgy-T*: ‹*cdcl-bnb-stgy-inv T*›
  using *rtranclp-cdcl-bnb-stgy-stgy-inv*[*OF st all-struct stgy-inv*] **.**
have *confl*: ‹*conflicting T = Some* {#}›
  using *no-step-cdcl-bnb-stgy-empty-conflict2*[*OF ns' struct-T stgy-T*] **.**
have *tot-I*: ‹*total-over-m I* (*set-mset* (*clauses T* + *conflicting-clss T*)) ⟷
  total-over-m I (*set-mset* (*init-clss T* + *conflicting-clss T*))› **for** *I*
  using *struct-T atms-of-conflicting-clss*[*of T*]
  unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*
    $cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause-alt-def satisfiable-def*
    $cdcl_W$*-restart-mset.no-strange-atm-def*
  by (*auto simp*: *clauses-def satisfiable-def total-over-m-alt-def*
    abs-state-def $cdcl_W$*-restart-mset-state*
    $cdcl_W$*-restart-mset.clauses-def*)
have ‹*unsatisfiable* (*set-mset* (*clauses T* + *conflicting-clss T*))›
  using *full-cdcl-bnb-stgy-unsat*[*OF - full all-struct - stgy-inv*]
  by (*auto simp*: *can-always-improve*)
have ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init*
   (*abs-state T*)›
  using *rtranclp-cdcl-bnb-*$cdcl_W$*-learned-clauses-entailed-by-init*[*OF st' ent all-struct*] **.**
then have ‹*init-clss T* + *conflicting-clss T* ⊨*pm* {#}›
  using *struct-T confl*
  unfolding $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*
    $cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause-alt-def*
    $cdcl_W$*-restart-mset.no-strange-atm-def tot-I*
    $cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init-def*
  by (*auto simp*: *clauses-def abs-state-def* $cdcl_W$*-restart-mset-state*
    $cdcl_W$*-restart-mset.clauses-def*
    satisfiable-def dest: *true-clss-clss-left-right*)
then have *unsat*: ‹*unsatisfiable* (*set-mset* (*init-clss T* + *conflicting-clss T*))›
  by (*auto simp*: *clauses-def true-clss-cls-def*
    satisfiable-def)

assume
  L-I: ‹*Pos L* ∈ *I*› **and**

L: ‹*ϱ L*› **and**
L-N: ‹*L* ∈ *atms-of-mm N*› **and**
tot-I: ‹*total-over-m I* (*set-mset N*)› **and**
cons: ‹*consistent-interp I*› **and**
I-N: ‹*I* ⊨*m N*›
show ‹*Multiset.Bex* (*covering T*) ((∈#) (*Pos L*))›
using *rtranclp-exists-model-with-true-lit-still-model*[*OF L-I L - - - - cdcl*, *of N*] *L-N*
I-N tot-I cons cov unsat
by (*auto simp*: *abs-state-def* $cdcl_W$*-restart-mset-state*)
qed

end

Now we instantiate the previous with λ-. *True*: This means that we aim at making all variables that appears at least ones true.

global-interpretation *cover-all-vars*: *covering-models* ‹λ-. *True*›
.

context *conflict-driven-clause-learning*$_W$*-covering-models*
begin

interpretation *cover-all-vars*: *conflict-driven-clause-learning*$_W$*-covering-models* **where**
ϱ = ‹λ-::′*v*. *True*› **and**
state = *state* **and**
trail = *trail* **and**
init-clss = *init-clss* **and**
learned-clss = *learned-clss* **and**
conflicting = *conflicting* **and**
cons-trail = *cons-trail* **and**
tl-trail = *tl-trail* **and**
add-learned-cls = *add-learned-cls* **and**
remove-cls = *remove-cls* **and**
update-conflicting = *update-conflicting* **and**
init-state = *init-state*
by *standard*

lemma
‹*cover-all-vars.model-is-dominated M M*′ ⟷
filter-mset (λ*L*. *is-pos L*) *M* ⊆# *filter-mset* (λ*L*. *is-pos L*) *M*′›
unfolding *cover-all-vars.model-is-dominated-def*
by *auto*

lemma
‹*cover-all-vars.conflicting-clauses N* 𝓜 =
{# *C* ∈# (*mset-set* (*simple-clss* (*atms-of-mm N*))).
(*pNeg* ‘
{*a*. *a* ∈# *mset-set* (*simple-clss* (*atms-of-mm N*)) ∧
(∃ *M*∈#𝓜. ∃ *J*. *a* ⊆# *J* ∧ *cover-all-vars.model-is-dominated J M*) ∧
atms-of a = *atms-of-mm N*} ∪
set-mset N) ⊨*p C*#}›
unfolding *cover-all-vars.conflicting-clauses-def*
cover-all-vars.is-dominating-def
by *auto*

theorem *cdclcm-correctness-all-vars*:
assumes

full: ‹*full cover-all-vars.cdcl-bnb-stgy* (*init-state N*) *T*› **and**
dist: ‹*distinct-mset-mset N*›
shows
‹*Pos L* ∈ *I* ⟹ *L* ∈ *atms-of-mm N* ⟹ *total-over-m I* (*set-mset N*) ⟹ *consistent-interp I* ⟹ *I* ⊨*m N* ⟹
∃ *J* ∈# *covering T*. *Pos L* ∈# *J*›
using *cover-all-vars.cdclcm-correctness*[*OF assms*]
by *blast*

end

end
theory *DPLL-W-BnB*
imports
CDCL-W-Optimal-Model
CDCL.DPLL-W
begin

lemma [*simp*]: ‹*backtrack-split M1* = (*M*′, *L* # *M*) ⟹ *is-decided L*›
by (*metis backtrack-split-snd-hd-decided list.sel*(*1*) *list.simps*(*3*) *snd-conv*)

lemma *funpow-tl-append-skip-ge*:
‹*n* ≥ *length M*′ ⟹ ((*tl* ^^ *n*) (*M*′ @ *M*)) = (*tl* ^^ (*n* − *length M*′)) *M*›
apply (*induction n arbitrary*: *M*′)
subgoal by *auto*
subgoal for *n M*′
by (*cases M*′)
(*auto simp del*: *funpow.simps*(*2*) *simp*: *funpow-Suc-right*)
done

The following version is more suited than ∃ *l*∈*set ?M*. *is-decided l* ⟹ ∃ *M*′ *L*′ *M*″. *backtrack-split ?M* = (*M*″, *L*′ # *M*′) for direct use.

lemma *backtrack-split-some-is-decided-then-snd-has-hd*′:
‹*l*∈*set M* ⟹ *is-decided l* ⟹ ∃ *M*′ *L*′ *M*″. *backtrack-split M* = (*M*″, *L*′ # *M*′)›
by (*metis backtrack-snd-empty-not-decided list.exhaust prod.collapse*)

lemma *total-over-m-entailed-or-conflict*:
shows ‹*total-over-m M N* ⟹ *M* ⊨*s N* ∨ (∃ *C* ∈ *N*. *M* ⊨*s CNot C*)›
by (*metis Set.set-insert total-not-true-cls-true-clss-CNot total-over-m-empty total-over-m-insert true-clss-def*)

The locales on DPLL should eventually be moved to the DPLL theory, but currently it is only a discount version (in particular, we cheat and don't use $S \sim T$ in the transition system below, even if it would be cleaner to do as as we de for CDCL).

locale *dpll-ops* =
fixes
trail :: ‹′*st* ⇒ ′*v* $dpll_W$*-ann-lits*› **and**
clauses :: ‹′*st* ⇒ ′*v clauses*› **and**
tl-trail :: ‹′*st* ⇒ ′*st*› **and**
cons-trail :: ‹′*v* $dpll_W$*-ann-lit* ⇒ ′*st* ⇒ ′*st*› **and**
state-eq :: ‹′*st* ⇒ ′*st* ⇒ *bool*› (**infix** ‹∼› *50*) **and**
state :: ‹′*st* ⇒ ′*v* $dpll_W$*-ann-lits* × ′*v clauses* × ′*b*›
begin

definition *additional-info* :: ‹′*st* ⇒ ′*b*› **where**
‹*additional-info S* = (λ(*M*, *N*, *w*). *w*) (*state S*)›

definition *reduce-trail-to* :: ‹'v dpll$_W$-ann-lits ⇒ 'st ⇒ 'st› **where**
‹*reduce-trail-to M S* = (*tl-trail* ^^ (*length* (*trail S*) − *length M*)) *S*›

end

locale *bnb-ops* =
fixes
trail :: ‹'st ⇒ 'v dpll$_W$-ann-lits› **and**
clauses :: ‹'st ⇒ 'v clauses› **and**
tl-trail :: ‹'st ⇒ 'st› **and**
cons-trail :: ‹'v dpll$_W$-ann-lit ⇒ 'st ⇒ 'st› **and**
state-eq :: ‹'st ⇒ 'st ⇒ bool› (**infix** ‹∼› *50*) **and**
state :: ‹'st ⇒ 'v dpll$_W$-ann-lits × 'v clauses × 'a × 'b› **and**
weight :: ‹'st ⇒ 'a› **and**
update-weight-information :: ‹'v dpll$_W$-ann-lits ⇒ 'st ⇒ 'st› **and**
is-improving-int :: ‹'v dpll$_W$-ann-lits ⇒ 'v dpll$_W$-ann-lits ⇒ 'v clauses ⇒ 'a ⇒ bool› **and**
conflicting-clauses :: ‹'v clauses ⇒ 'a ⇒ 'v clauses›
begin

interpretation *dpll*: *dpll-ops* **where**
trail = *trail* **and**
clauses = *clauses* **and**
tl-trail = *tl-trail* **and**
cons-trail = *cons-trail* **and**
state-eq = *state-eq* **and**
state = *state*
.

definition *conflicting-clss* :: ‹'st ⇒ 'v literal multiset multiset› **where**
‹*conflicting-clss S* = *conflicting-clauses* (*clauses S*) (*weight S*)›

definition *abs-state* **where**
‹*abs-state S* = (*trail S*, *clauses S* + *conflicting-clss S*)›

abbreviation *is-improving* **where**
‹*is-improving M M' S* ≡ *is-improving-int M M'* (*clauses S*) (*weight S*)›

definition *state'* :: ‹'st ⇒ 'v dpll$_W$-ann-lits × 'v clauses × 'a × 'v clauses› **where**
‹*state' S* = (*trail S*, *clauses S*, *weight S*, *conflicting-clss S*)›

definition *additional-info* :: ‹'st ⇒ 'b› **where**
‹*additional-info S* = (λ(*M*, *N*, -, *w*). *w*) (*state S*)›

end

locale *dpll$_W$-state* =
dpll-ops trail clauses
tl-trail cons-trail state-eq state
for
trail :: ‹'st ⇒ 'v dpll$_W$-ann-lits› **and**

clauses :: ‹′*st* ⇒ ′*v clauses*› **and**
tl-trail :: ‹′*st* ⇒ ′*st*› **and**
cons-trail :: ‹′*v* $dpll_W$*-ann-lit* ⇒ ′*st* ⇒ ′*st*› **and**
state-eq :: ‹′*st* ⇒ ′*st* ⇒ *bool*› (**infix** ‹∼› *50*) **and**
state :: ‹′*st* ⇒ ′*v* $dpll_W$*-ann-lits* × ′*v clauses* × ′*b*› +
assumes
state-eq-ref[*simp*, *intro*]: ‹*S* ∼ *S*› **and**
state-eq-sym: ‹*S* ∼ *T* ⟷ *T* ∼ *S*› **and**
state-eq-trans: ‹*S* ∼ *T* ⟹ *T* ∼ *U*′ ⟹ *S* ∼ *U*′› **and**
state-eq-state: ‹*S* ∼ *T* ⟹ *state S* = *state T*› **and**

cons-trail:
⋀*S*′. *state st* = (*M*, *S*′) ⟹
state (*cons-trail L st*) = (*L* # *M*, *S*′) **and**

tl-trail:
‹⋀*S*′. *state st* = (*M*, *S*′) ⟹ *state* (*tl-trail st*) = (*tl M*, *S*′)› **and**
state:
‹*state S* = (*trail S*, *clauses S*, *additional-info S*)›
begin

lemma [*simp*]:
‹*clauses* (*cons-trail uu S*) = *clauses S*›
‹*trail* (*cons-trail uu S*) = *uu* # *trail S*›
‹*trail* (*tl-trail S*) = *tl* (*trail S*)›
‹*clauses* (*tl-trail S*) = *clauses* (*S*)›
‹*additional-info* (*cons-trail L S*) = *additional-info S*›
‹*additional-info* (*tl-trail S*) = *additional-info S*›
using
cons-trail[*of S*]
tl-trail[*of S*]
by (*auto simp*: *state*)

lemma *state-simp*[*simp*]:
‹*T* ∼ *S* ⟹ *trail T* = *trail S*›
‹*T* ∼ *S* ⟹ *clauses T* = *clauses S*›
by (*auto dest*!: *state-eq-state simp*: *state*)

lemma *state-tl-trail*: ‹*state* (*tl-trail S*) = (*tl* (*trail S*), *clauses S*, *additional-info S*)›
by (*auto simp*: *state*)

lemma *state-tl-trail-comp-pow*: ‹*state* ((*tl-trail* ^^ *n*) *S*) = ((*tl* ^^ *n*) (*trail S*), *clauses S*, *additional-info S*)›
apply (*induction n*)
using *state* **apply** *fastforce*
apply (*auto simp*: *state-tl-trail state*)[]
done

lemma *reduce-trail-to-simps*[*simp*]:
‹*backtrack-split* (*trail S*) = (*M*′, *L* # *M*) ⟹ *trail* (*reduce-trail-to M S*) = *M*›
‹*clauses* (*reduce-trail-to M S*) = *clauses S*›
‹*additional-info* (*reduce-trail-to M S*) = *additional-info S*›
using *state-tl-trail-comp-pow*[*of* ‹*Suc* (*length M*′)› *S*] *backtrack-split-list-eq*[*of* ‹*trail S*›, *symmetric*]
unfolding *reduce-trail-to-def*

apply (*auto simp*: *state funpow-tl-append-skip-ge*)
using *state state-tl-trail-comp-pow* **apply** *auto*
done

inductive *dpll-backtrack* :: ‹′*st* ⇒ ′*st* ⇒ *bool*› **where**
‹*dpll-backtrack S T*›
if
‹*D* ∈# *clauses S*› **and**
‹*trail S* ⊨*as CNot D*› **and**
‹*backtrack-split* (*trail S*) = (*M*′, *L* # *M*)› **and**
‹*T* ∼*cons-trail* (*Propagated* (−*lit-of L*) ()) (*reduce-trail-to M S*)›

inductive *dpll-propagate* :: ‹′*st* ⇒ ′*st* ⇒ *bool*› **where**
‹*dpll-propagate S T*›
if
‹*add-mset L D* ∈# *clauses S*› **and**
‹*trail S* ⊨*as CNot D*› **and**
‹*undefined-lit* (*trail S*) *L*›
‹*T* ∼ *cons-trail* (*Propagated L* ()) *S*›

inductive *dpll-decide* :: ‹′*st* ⇒ ′*st* ⇒ *bool*› **where**
‹*dpll-decide S T*›
if
‹*undefined-lit* (*trail S*) *L*›
‹*T* ∼ *cons-trail* (*Decided L*) *S*›
‹*atm-of L* ∈ *atms-of-mm* (*clauses S*)›

inductive *dpll* :: ‹′*st* ⇒ ′*st* ⇒ *bool*› **where**
‹*dpll S T*› **if** ‹*dpll-decide S T*› |
‹*dpll S T*› **if** ‹*dpll-propagate S T*› |
‹*dpll S T*› **if** ‹*dpll-backtrack S T*›

lemma *dpll-is-dpll*$_W$:
‹*dpll S T* ⟹ *dpll*$_W$ (*trail S*, *clauses S*) (*trail T*, *clauses T*)›
apply (*induction rule*: *dpll.induct*)
subgoal for *S T*
apply (*auto simp*: *dpll.simps dpll*$_W$*.simps dpll-decide.simps dpll-backtrack.simps dpll-propagate.simps*
dest!: *multi-member-split*[*of* - ‹*clauses S*›])
done
subgoal for *S T*
unfolding *dpll.simps dpll*$_W$*.simps dpll-decide.simps dpll-backtrack.simps dpll-propagate.simps*
by *auto*
subgoal for *S T*
unfolding *dpll*$_W$*.simps dpll-decide.simps dpll-backtrack.simps dpll-propagate.simps*
by (*auto simp*: *state*)
done

end

locale *bnb* =
bnb-ops trail clauses
tl-trail cons-trail state-eq state weight update-weight-information is-improving-int conflicting-clauses
for
weight :: ‹′*st* ⇒ ′*a*› **and**
update-weight-information :: ‹′*v dpll*$_W$*-ann-lits* ⇒ ′*st* ⇒ ′*st*› **and**

is-improving-int :: ‹′*v* *dpll*$_W$-*ann-lits* ⇒ ′*v* *dpll*$_W$-*ann-lits* ⇒ ′*v clauses* ⇒ ′*a* ⇒ *bool*› **and**
trail :: ‹′*st* ⇒ ′*v* *dpll*$_W$-*ann-lits*› **and**
clauses :: ‹′*st* ⇒ ′*v clauses*› **and**
tl-trail :: ‹′*st* ⇒ ′*st*› **and**
cons-trail :: ‹′*v* *dpll*$_W$-*ann-lit* ⇒ ′*st* ⇒ ′*st*› **and**
state-eq :: ‹′*st* ⇒ ′*st* ⇒ *bool*› (**infix** ‹∼› *50*) **and**
conflicting-clauses :: ‹′*v clauses* ⇒ ′*a* ⇒ ′*v clauses*›**and**
state :: ‹′*st* ⇒ ′*v* *dpll*$_W$-*ann-lits* × ′*v clauses* × ′*a* × ′*b*› +
assumes
state-eq-ref[*simp*, *intro*]: ‹*S* ∼ *S*› **and**
state-eq-sym: ‹*S* ∼ *T* ⟷ *T* ∼ *S*› **and**
state-eq-trans: ‹*S* ∼ *T* ⟹ *T* ∼ *U′* ⟹ *S* ∼ *U′*› **and**
state-eq-state: ‹*S* ∼ *T* ⟹ *state S* = *state T*› **and**

cons-trail:
⋀*S′*. *state st* = (*M*, *S′*) ⟹
state (*cons-trail L st*) = (*L* # *M*, *S′*) **and**

tl-trail:
‹⋀*S′*. *state st* = (*M*, *S′*) ⟹ *state* (*tl-trail st*) = (*tl M*, *S′*)› **and**
update-weight-information:
‹*state S* = (*M*, *N*, *w*, *oth*) ⟹
∃ *w′*. *state* (*update-weight-information M′ S*) = (*M*, *N*, *w′*, *oth*)› **and**

conflicting-clss-update-weight-information-mono:
‹*dpll*$_W$-*all-inv* (*abs-state S*) ⟹ *is-improving M M′ S* ⟹
conflicting-clss S ⊆# *conflicting-clss* (*update-weight-information M′ S*)› **and**
conflicting-clss-update-weight-information-in:
‹*is-improving M M′ S* ⟹ *negate-ann-lits M′* ∈# *conflicting-clss* (*update-weight-information M′ S*)› **and**
atms-of-conflicting-clss:
‹*atms-of-mm* (*conflicting-clss S*) ⊆ *atms-of-mm* (*clauses S*)› **and**
state:
‹*state S* = (*trail S*, *clauses S*, *weight S*, *additional-info S*)›
begin

lemma [*simp*]: ‹*DPLL-W.clauses* (*abs-state S*) = *clauses S* + *conflicting-clss S*›
‹*DPLL-W.trail* (*abs-state S*) = *trail S*›
by (*auto simp*: *abs-state-def*)

lemma [*simp*]: ‹*trail* (*update-weight-information M′ S*) = *trail S*›
using *update-weight-information*[*of S*]
by (*auto simp*: *state*)

lemma [*simp*]:
‹*clauses* (*update-weight-information M′ S*) = *clauses S*›
‹*weight* (*cons-trail uu S*) = *weight S*›
‹*clauses* (*cons-trail uu S*) = *clauses S*›
‹*conflicting-clss* (*cons-trail uu S*) = *conflicting-clss S*›
‹*trail* (*cons-trail uu S*) = *uu* # *trail S*›
‹*trail* (*tl-trail S*) = *tl* (*trail S*)›
‹*clauses* (*tl-trail S*) = *clauses* (*S*)›
‹*weight* (*tl-trail S*) = *weight* (*S*)›
‹*conflicting-clss* (*tl-trail S*) = *conflicting-clss* (*S*)›
‹*additional-info* (*cons-trail L S*) = *additional-info S*›

‹*additional-info* (*tl-trail S*) = *additional-info S*›
‹*additional-info* (*update-weight-information M′ S*) = *additional-info S*›
using *update-weight-information*[*of S*]
cons-trail[*of S*]
tl-trail[*of S*]
by (*auto simp*: *state conflicting-clss-def*)

lemma *state-simp*[*simp*]:
‹*T* ∼ *S* ⟹ *trail T* = *trail S*›
‹*T* ∼ *S* ⟹ *clauses T* = *clauses S*›
‹*T* ∼ *S* ⟹ *weight T* = *weight S*›
‹*T* ∼ *S* ⟹ *conflicting-clss T* = *conflicting-clss S*›
by (*auto dest*!: *state-eq-state simp*: *state conflicting-clss-def*)

interpretation *dpll*: *dpll-ops trail clauses tl-trail cons-trail state-eq state*
.

interpretation *dpll*: $dpll_W$*-state trail clauses tl-trail cons-trail state-eq state*
apply *standard*
apply (*auto dest*: *state-eq-sym*[*THEN iffD1*] *intro*: *state-eq-trans dest*: *state-eq-state*)
apply (*auto simp*: *state cons-trail dpll.additional-info-def*)
done

lemma [*simp*]:
‹*conflicting-clss* (*dpll.reduce-trail-to M S*) = *conflicting-clss S*›
‹*weight* (*dpll.reduce-trail-to M S*) = *weight S*›
using *dpll.reduce-trail-to-simps*(*2*−)[*of M S*] *state*[*of S*]
unfolding *dpll.additional-info-def*
apply (*auto simp*:)
by (*smt conflicting-clss-def dpll.reduce-trail-to-simps*(*2*) *dpll.state dpll-ops.additional-info-def old.prod.inject state*)+

inductive *backtrack-opt* :: ‹′*st* ⇒ ′*st* ⇒ *bool*› **where**
backtrack-opt: *backtrack-split* (*trail S*) = (*M′*, *L* # *M*) ⟹ *is-decided L* ⟹ *D* ∈# *conflicting-clss S*
⟹ *trail S* ⊨*as CNot D*
⟹ *T* ∼*cons-trail* (*Propagated* (−*lit-of L*) ()) (*dpll.reduce-trail-to M S*)
⟹ *backtrack-opt S T*

In the definition below the *state′ T* = (*Propagated L* () # *trail S*, *clauses S*, *weight S*, *conflicting-clss S*) are not necessary, but avoids to change the DPLL formalisation with proper locales, as we did for CDCL.

The DPLL calculus looks slightly more general than the CDCL calculus because we can take any conflicting clause from *conflicting-clss S*. However, this does not make a difference for the trail, as we backtrack to the last decision independantly of the conflict.

inductive $dpll_W$*-core* :: ‹′*st* ⇒ ′*st* ⇒ *bool*› **for** *S T* **where**
propagate: ‹*dpll.dpll-propagate S T* ⟹ $dpll_W$*-core S T*› |
decided: ‹*dpll.dpll-decide S T* ⟹ $dpll_W$*-core S T* › |
backtrack: ‹*dpll.dpll-backtrack S T* ⟹ $dpll_W$*-core S T*› |
backtrack-opt: ‹*backtrack-opt S T* ⟹ $dpll_W$*-core S T*›

inductive-cases $dpll_W$*-coreE*: ‹$dpll_W$*-core S T*›

inductive $dpll_W$*-bound* :: ‹′*st* ⇒ ′*st* ⇒ *bool*› **where**
update-info:
‹*is-improving M M′ S* ⟹ *T* ∼ (*update-weight-information M′ S*)

```
  ⟹ dpll_W-bound S T›

inductive dpll_W-bnb :: ‹'st ⇒ 'st ⇒ bool› where
dpll:
  ‹dpll_W-bnb S T›
  if ‹dpll_W-core S T› |
bnb:
  ‹dpll_W-bnb S T›
  if ‹dpll_W-bound S T›


inductive-cases dpll_W-bnbE: ‹dpll_W-bnb S T›

lemma dpll_W-core-is-dpll_W:
  ‹dpll_W-core S T ⟹ dpll_W (abs-state S) (abs-state T)›
  supply abs-state-def[simp] state'-def[simp]
  apply (induction rule: dpll_W-core.induct)
  subgoal
    by (auto simp: dpll_W.simps dpll.dpll-propagate.simps)
  subgoal
    by (auto simp: dpll_W.simps dpll.dpll-decide.simps)
  subgoal
    by (auto simp: dpll_W.simps dpll.dpll-backtrack.simps)
  subgoal
    by (auto simp: dpll_W.simps backtrack-opt.simps)
  done

lemma dpll_W-core-abs-state-all-inv:
  ‹dpll_W-core S T ⟹ dpll_W-all-inv (abs-state S) ⟹ dpll_W-all-inv (abs-state T)›
  by (auto dest!: dpll_W-core-is-dpll_W intro: dpll_W-all-inv)

lemma dpll_W-core-same-weight:
  ‹dpll_W-core S T ⟹ weight S = weight T›
  supply abs-state-def[simp] state'-def[simp]
  apply (induction rule: dpll_W-core.induct)
  subgoal
    by (auto simp: dpll_W.simps dpll.dpll-propagate.simps)
  subgoal
    by (auto simp: dpll_W.simps dpll.dpll-decide.simps)
  subgoal
    by (auto simp: dpll_W.simps dpll.dpll-backtrack.simps)
  subgoal
    by (auto simp: dpll_W.simps backtrack-opt.simps)
  done

lemma dpll_W-bound-trail:
    ‹dpll_W-bound S T ⟹ trail S = trail T› and
   dpll_W-bound-clauses:
    ‹dpll_W-bound S T ⟹ clauses S = clauses T› and
  dpll_W-bound-conflicting-clss:
    ‹dpll_W-bound S T ⟹ dpll_W-all-inv (abs-state S) ⟹ conflicting-clss S ⊆# conflicting-clss T›
  subgoal
    by (induction rule: dpll_W-bound.induct)
     (auto simp: dpll_W-all-inv-def state dest!: conflicting-clss-update-weight-information-mono)
  subgoal
    by (induction rule: dpll_W-bound.induct)
```

(*auto simp*: $dpll_W$*-all-inv-def state dest*!: *conflicting-clss-update-weight-information-mono*)
subgoal
by (*induction rule*: $dpll_W$*-bound.induct*)
(*auto simp*: *state conflicting-clss-def*
dest!: *conflicting-clss-update-weight-information-mono*)
done

lemma $dpll_W$*-bound-abs-state-all-inv*:
‹$dpll_W$*-bound S T* ⟹ $dpll_W$*-all-inv* (*abs-state S*) ⟹ $dpll_W$*-all-inv* (*abs-state T*)›
using $dpll_W$*-bound-conflicting-clss*[*of S T*] $dpll_W$*-bound-clauses*[*of S T*]
atms-of-conflicting-clss[*of T*] *atms-of-conflicting-clss*[*of S*]
apply (*auto simp*: $dpll_W$*-all-inv-def* $dpll_W$*-bound-trail lits-of-def image-image*
intro: *all-decomposition-implies-mono*[*OF set-mset-mono*] *dest*: $dpll_W$*-bound-conflicting-clss*)
by (*blast intro*: *all-decomposition-implies-mono*)

lemma $dpll_W$*-bnb-abs-state-all-inv*:
‹$dpll_W$*-bnb S T* ⟹ $dpll_W$*-all-inv* (*abs-state S*) ⟹ $dpll_W$*-all-inv* (*abs-state T*)›
by (*auto elim*!: $dpll_W$*-bnb.cases intro*: $dpll_W$*-bound-abs-state-all-inv* $dpll_W$*-core-abs-state-all-inv*)

lemma *rtranclp-*$dpll_W$*-bnb-abs-state-all-inv*:
‹$dpll_W$*-bnb*** *S T* ⟹ $dpll_W$*-all-inv* (*abs-state S*) ⟹ $dpll_W$*-all-inv* (*abs-state T*)›
by (*induction rule*: *rtranclp-induct*)
(*auto simp*: $dpll_W$*-bnb-abs-state-all-inv*)

lemma $dpll_W$*-core-clauses*:
‹$dpll_W$*-core S T* ⟹ *clauses S* = *clauses T*›
supply *abs-state-def*[*simp*] *state'-def*[*simp*]
apply (*induction rule*: $dpll_W$*-core.induct*)
subgoal
by (*auto simp*: $dpll_W$*.simps dpll.dpll-propagate.simps*)
subgoal
by (*auto simp*: $dpll_W$*.simps dpll.dpll-decide.simps*)
subgoal
by (*auto simp*: $dpll_W$*.simps dpll.dpll-backtrack.simps*)
subgoal
by (*auto simp*: $dpll_W$*.simps backtrack-opt.simps*)
done

lemma $dpll_W$*-bnb-clauses*:
‹$dpll_W$*-bnb S T* ⟹ *clauses S* = *clauses T*›
by (*auto elim*!: $dpll_W$*-bnbE simp*: $dpll_W$*-bound-clauses* $dpll_W$*-core-clauses*)

lemma *rtranclp-*$dpll_W$*-bnb-clauses*:
‹$dpll_W$*-bnb*** *S T* ⟹ *clauses S* = *clauses T*›
by (*induction rule*: *rtranclp-induct*)
(*auto simp*: $dpll_W$*-bnb-clauses*)

lemma *atms-of-clauses-conflicting-clss*[*simp*]:
‹*atms-of-mm* (*clauses S*) ∪ *atms-of-mm* (*conflicting-clss S*) = *atms-of-mm* (*clauses S*)›
using *atms-of-conflicting-clss*[*of S*] **by** *blast*

lemma *wf-*$dpll_W$*-bnb-bnb*:
assumes *improve*: ‹⋀*S T*. $dpll_W$*-bound S T* ⟹ *clauses S* = *N* ⟹ (ν (*weight T*), ν (*weight S*)) ∈ *R*› **and**
wf-R: ‹*wf R*›

shows ‹*wf* {(*T*, *S*). *dpll*$_W$*-all-inv* (*abs-state S*) ∧ *dpll*$_W$*-bnb S T* ∧
clauses S = *N*}›
(**is** ‹*wf ?A*›)
proof −
let *?R* = ‹{(*T*, *S*). (*ν* (*weight T*), *ν* (*weight S*)) ∈ *R*}›

have ‹*wf* {(*T*, *S*). *dpll*$_W$*-all-inv S* ∧ *dpll*$_W$ *S T*}›
by (*rule wf-dpll*$_W$)
from *wf-if-measure-f*[*OF this*, *of abs-state*]
have *wf*: ‹*wf* {(*T*, *S*). *dpll*$_W$*-all-inv* (*abs-state S*) ∧
dpll$_W$ (*abs-state S*) (*abs-state T*) ∧ *weight S* = *weight T*}›
(**is** ‹*wf ?CDCL*›)
by (*rule wf-subset*) *auto*
have ‹*wf* (*?R* ∪ *?CDCL*)›
apply (*rule wf-union-compatible*)
subgoal by (*rule wf-if-measure-f*[*OF wf-R*, *of* ‹*λx*. *ν* (*weight x*)›])
subgoal by (*rule wf*)
subgoal by (*auto simp*: *dpll*$_W$*-core-same-weight*)
done

moreover have ‹*?A* ⊆ *?R* ∪ *?CDCL*›
by (*auto elim!*: *dpll*$_W$*-bnbE dest*: *dpll*$_W$*-core-abs-state-all-inv dpll*$_W$*-core-is-dpll*$_W$
simp: *dpll*$_W$*-core-same-weight improve*)
ultimately show *?thesis*
by (*rule wf-subset*)
qed

lemma [*simp*]:
‹*weight* ((*tl-trail* ^^ *n*) *S*) = *weight S*›
‹*trail* ((*tl-trail* ^^ *n*) *S*) = (*tl* ^^ *n*) (*trail S*)›
‹*clauses* ((*tl-trail* ^^ *n*) *S*) = *clauses S*›
‹*conflicting-clss* ((*tl-trail* ^^ *n*) *S*) = *conflicting-clss S*›
using *dpll.state-tl-trail-comp-pow*[*of n S*]
apply (*auto simp*: *state conflicting-clss-def*)
apply (*metis* (*mono-tags*, *lifting*) *Pair-inject dpll.state state*)+
done

lemma *dpll*$_W$*-core-Ex-propagate*:
‹*add-mset L C* ∈# *clauses S* ⟹ *trail S* ⊨*as CNot C* ⟹ *undefined-lit* (*trail S*) *L* ⟹
Ex (*dpll*$_W$*-core S*)› **and**
dpll$_W$*-core-Ex-decide*:
undefined-lit (*trail S*) *L* ⟹ *atm-of L* ∈ *atms-of-mm* (*clauses S*) ⟹
Ex (*dpll*$_W$*-core S*) **and**
dpll$_W$*-core-Ex-backtrack*: *backtrack-split* (*trail S*) = (*M'*, *L'* # *M*) ⟹ *is-decided L'* ⟹ *D* ∈#
clauses S ⟹
trail S ⊨*as CNot D* ⟹ *Ex* (*dpll*$_W$*-core S*) **and**
dpll$_W$*-core-Ex-backtrack-opt*: *backtrack-split* (*trail S*) = (*M'*, *L'* # *M*) ⟹ *is-decided L'* ⟹ *D* ∈#
conflicting-clss S
⟹ *trail S* ⊨*as CNot D* ⟹
Ex (*dpll*$_W$*-core S*)
subgoal
by (*rule exI*[*of* - ‹*cons-trail* (*Propagated L* ()) *S*›])
(*fastforce simp*: *dpll*$_W$*-core.simps state-eq-ref dpll.dpll-propagate.simps*)
subgoal
by (*rule exI*[*of* - ‹*cons-trail* (*Decided L*) *S*›])

(*auto simp*: $dpll_W$*-core.simps state′-def dpll.dpll-decide.simps dpll.dpll-backtrack.simps backtrack-opt.simps dpll.dpll-propagate.simps*)
subgoal
using *backtrack-split-list-eq*[*of* ‹*trail S*›, *symmetric*] **apply** −
apply (*rule exI*[*of* - ‹*cons-trail* (*Propagated* (−*lit-of L′*) ()) (*dpll.reduce-trail-to M S*)›])
apply (*auto simp*: $dpll_W$*-core.simps state′-def funpow-tl-append-skip-ge dpll.dpll-decide.simps dpll.dpll-backtrack.simps backtrack-opt.simps dpll.dpll-propagate.simps*)
done
subgoal
using *backtrack-split-list-eq*[*of* ‹*trail S*›, *symmetric*] **apply** −
apply (*rule exI*[*of* - ‹*cons-trail* (*Propagated* (−*lit-of L′*) ()) (*dpll.reduce-trail-to M S*)›])
apply (*auto simp*: $dpll_W$*-core.simps state′-def funpow-tl-append-skip-ge dpll.dpll-decide.simps dpll.dpll-backtrack.simps backtrack-opt.simps dpll.dpll-propagate.simps*)
done
done

Unlike the CDCL case, we do not need assumptions on improve. The reason behind it is that we do not need any strategy on propagation and decisions.

lemma *no-step-dpll-bnb-*$dpll_W$:
assumes
ns: ‹*no-step* $dpll_W$*-bnb S*› **and**
struct-invs: ‹$dpll_W$*-all-inv* (*abs-state S*)›
shows ‹*no-step* $dpll_W$ (*abs-state S*)›
proof −
have *no-decide*: ‹*atm-of L* ∈ *atms-of-mm* (*clauses S*) ⟹ *defined-lit* (*trail S*) *L*› **for** *L*
using *spec*[*OF ns*, *of* ‹*cons-trail* - *S*›]
apply (*fastforce simp*: $dpll_W$*-bnb.simps total-over-m-def total-over-set-def* $dpll_W$*-core.simps state′-def dpll.dpll-decide.simps dpll.dpll-backtrack.simps backtrack-opt.simps dpll.dpll-propagate.simps*)
done
have [*intro*]: ‹*is-decided L* ⟹ *backtrack-split* (*trail S*) = (*M′*, *L* # *M*) ⟹ *trail S* ⊨*as CNot D* ⟹ *D* ∈# *clauses S* ⟹ *False*› **for** *M′ L M D*
using $dpll_W$*-core-Ex-backtrack*[*of S M′ L M D*] *ns*
by (*auto simp*: $dpll_W$*-bnb.simps*)
have [*intro*]: ‹*is-decided L* ⟹ *backtrack-split* (*trail S*) = (*M′*, *L* # *M*) ⟹ *trail S* ⊨*as CNot D* ⟹ *D* ∈# *conflicting-clss S* ⟹ *False*› **for** *M′ L M D*
using $dpll_W$*-core-Ex-backtrack-opt*[*of S M′ L M D*] *ns*
by (*auto simp*: $dpll_W$*-bnb.simps*)
have *tot*: ‹*total-over-m* (*lits-of-l* (*trail S*)) (*set-mset* (*clauses S*))›
using *no-decide*
by (*force simp*: *total-over-m-def total-over-set-def state′-def Decided-Propagated-in-iff-in-lits-of-l*)
have [*simp*]: ‹*add-mset L C* ∈# *clauses S* ⟹ *defined-lit* (*trail S*) *L*› **for** *L C*
using *no-decide*
by (*auto dest*!: *multi-member-split*)
have [*simp*]: ‹*add-mset L C* ∈# *conflicting-clss S* ⟹ *defined-lit* (*trail S*) *L*› **for** *L C*
using *no-decide atms-of-conflicting-clss*[*of S*]
by (*auto dest*!: *multi-member-split*)
show *?thesis*
by (*auto simp*: $dpll_W$*.simps no-decide*)

qed

context
assumes *can-always-improve*:
‹⋀*S*. *trail S* ⊨*asm clauses S* ⟹ (∀ *C* ∈# *conflicting-clss S*. ¬ *trail S* ⊨*as CNot C*) ⟹
$dpll_W$*-all-inv* (*abs-state S*) ⟹
total-over-m (*lits-of-l* (*trail S*)) (*set-mset* (*clauses S*)) ⟹ *Ex* ($dpll_W$*-bound S*)›
begin

lemma *no-step-*$dpll_W$*-bnb-conflict*:
assumes
ns: ‹*no-step* $dpll_W$*-bnb S*› **and**
invs: ‹$dpll_W$*-all-inv* (*abs-state S*)›
shows ‹∃ *C* ∈# *clauses S* + *conflicting-clss S*. *trail S* ⊨*as CNot C*› (**is** *?A*) **and**
‹*count-decided* (*trail S*) = *0*› **and**
‹*unsatisfiable* (*set-mset* (*clauses S* + *conflicting-clss S*))›
proof (*rule ccontr*)
have *no-decide*: ‹*atm-of L* ∈ *atms-of-mm* (*clauses S*) ⟹ *defined-lit* (*trail S*) *L*› **for** *L*
using *spec*[*OF ns*, *of* ‹*cons-trail - S*›]
apply (*fastforce simp*: $dpll_W$*-bnb.simps total-over-m-def total-over-set-def*
$dpll_W$*-core.simps state'-def*
dpll.dpll-decide.simps dpll.dpll-backtrack.simps backtrack-opt.simps
dpll.dpll-propagate.simps)
done
have *tot*: ‹*total-over-m* (*lits-of-l* (*trail S*)) (*set-mset* (*clauses S*))›
using *no-decide*
by (*force simp*: *total-over-m-def total-over-set-def state'-def*
Decided-Propagated-in-iff-in-lits-of-l)
have *dec0*: ‹*count-decided* (*trail S*) = *0*› **if** *ent*: ‹*?A*›
proof −
obtain *C* **where**
‹*C* ∈# *clauses S* + *conflicting-clss S*› **and**
‹*trail S* ⊨*as CNot C*›
using *ent tot ns invs*
by (*auto simp*: $dpll_W$*-bnb.simps*)
then show ‹*count-decided* (*trail S*) = *0*›
using *ns* $dpll_W$*-core-Ex-backtrack*[*of S - - - C*] $dpll_W$*-core-Ex-backtrack-opt*[*of S - - - C*]
unfolding *count-decided-0-iff*
apply *clarify*
apply (*frule backtrack-split-some-is-decided-then-snd-has-hd'*[*of -* ‹*trail S*›], *assumption*)
apply (*auto simp*: $dpll_W$*-bnb.simps count-decided-0-iff*)
done
qed

show *A*: *False* **if** ‹¬*?A*›
proof −
have ‹*trail S* ⊨*a C*› **if** ‹*C* ∈# *clauses S* + *conflicting-clss S*› **for** *C*
proof −
have ‹¬ *trail S* ⊨*as CNot C*›
using ‹¬*?A*› *that* **by** (*auto dest*!: *multi-member-split*)
then show ‹*?thesis*›
using *tot that*
total-not-true-cls-true-clss-CNot[*of* ‹*lits-of-l* (*trail S*)› *C*]
apply (*auto simp*: *true-annots-def simp del*: *true-clss-def-iff-negation-in-model dest*!: *multi-member-split*
)

using *true-annot-def* **apply** *blast*
using *true-annot-def* **apply** *blast*
by (*metis Decided-Propagated-in-iff-in-lits-of-l atms-of-clauses-conflicting-clss atms-of-ms-union in-m-in-literals no-decide set-mset-union that true-annot-def true-cls-add-mset*)
qed
then have ‹*trail S* ⊨*asm clauses S* + *conflicting-clss S*›
by (*auto simp*: *true-annots-def dest*!: *multi-member-split*)
then show *?thesis*
using *can-always-improve*[*of S*] ‹¬*?A*› *tot invs ns* **by** (*auto simp*: $dpll_W$*-bnb.simps*)
qed
then show ‹*count-decided* (*trail S*) = *0*›
using *dec0* **by** *blast*
moreover have *?A*
using *A* **by** *blast*
ultimately show ‹*unsatisfiable* (*set-mset* (*clauses S* + *conflicting-clss S*))›
using *only-propagated-vars-unsat*[*of* ‹*trail S*› - ‹*set-mset* (*clauses S* + *conflicting-clss S*)›] *invs*
unfolding $dpll_W$*-all-inv-def count-decided-0-iff*
by *auto*
qed

end

inductive $dpll_W$*-core-stgy* :: ‹*'st* ⇒ *'st* ⇒ *bool*› **for** *S T* **where**
propagate: ‹*dpll.dpll-propagate S T* ⟹ $dpll_W$*-core-stgy S T*› |
decided: ‹*dpll.dpll-decide S T* ⟹ *no-step dpll.dpll-propagate S* ⟹ $dpll_W$*-core-stgy S T* › |
backtrack: ‹*dpll.dpll-backtrack S T* ⟹ $dpll_W$*-core-stgy S T*› |
backtrack-opt: ‹*backtrack-opt S T* ⟹ $dpll_W$*-core-stgy S T*›

lemma $dpll_W$*-core-stgy-*$dpll_W$*-core*: ‹$dpll_W$*-core-stgy S T* ⟹ $dpll_W$*-core S T*›
by (*induction rule*: $dpll_W$*-core-stgy.induct*)
(*auto intro*: $dpll_W$*-core.intros*)

lemma *rtranclp-*$dpll_W$*-core-stgy-*$dpll_W$*-core*: ‹$dpll_W$*-core-stgy*** *S T* ⟹ $dpll_W$*-core*** *S T*›
by (*induction rule*: *rtranclp-induct*)
(*auto dest*: $dpll_W$*-core-stgy-*$dpll_W$*-core*)

lemma *no-step-stgy-iff*: ‹*no-step* $dpll_W$*-core-stgy S* ⟷ *no-step* $dpll_W$*-core S*›
by (*auto simp*: $dpll_W$*-core-stgy.simps* $dpll_W$*-core.simps*)

lemma *full-*$dpll_W$*-core-stgy-*$dpll_W$*-core*: ‹*full* $dpll_W$*-core-stgy S T* ⟹ *full* $dpll_W$*-core S T*›
unfolding *full-def* **by** (*simp add*: *no-step-stgy-iff rtranclp-*$dpll_W$*-core-stgy-*$dpll_W$*-core*)

lemma $dpll_W$*-core-stgy-clauses*:
‹$dpll_W$*-core-stgy S T* ⟹ *clauses T* = *clauses S*›
by (*induction rule*: $dpll_W$*-core-stgy.induct*)
(*auto simp*: *dpll.dpll-propagate.simps dpll.dpll-decide.simps dpll.dpll-backtrack.simps backtrack-opt.simps*)

lemma *rtranclp-*$dpll_W$*-core-stgy-clauses*:
‹$dpll_W$*-core-stgy*** *S T* ⟹ *clauses T* = *clauses S*›
by (*induction rule*: *rtranclp-induct*)
(*auto dest*: $dpll_W$*-core-stgy-clauses*)

end

end
theory *DPLL-W-Optimal-Model*
imports
  DPLL-W-BnB
begin

locale $dpll_W$*-state-optimal-weight* =
  $dpll_W$*-state trail clauses*
    tl-trail cons-trail state-eq state +
  ocdcl-weight ϱ
  for
    trail :: ‹′*st* ⇒ ′*v* $dpll_W$*-ann-lits*› **and**
    clauses :: ‹′*st* ⇒ ′*v clauses*› **and**
    tl-trail :: ‹′*st* ⇒ ′*st*› **and**
    cons-trail :: ‹′*v* $dpll_W$*-ann-lit* ⇒ ′*st* ⇒ ′*st*› **and**
    state-eq :: ‹′*st* ⇒ ′*st* ⇒ *bool*› (**infix** ‹∼› *50*) **and**
    state :: ‹′*st* ⇒ ′*v* $dpll_W$*-ann-lits* × ′*v clauses* × ′*v clause option* × ′*b*› **and**
    ϱ :: ‹′*v clause* ⇒ ′*a* :: {*linorder*}› +
  fixes
    update-additional-info :: ‹′*v clause option* × ′*b* ⇒ ′*st* ⇒ ′*st*›
  assumes
    update-additional-info:
      ‹*state S* = (*M*, *N*, *K*) ⟹ *state* (*update-additional-info K′ S*) = (*M*, *N*, *K′*)›
begin

definition *update-weight-information* :: ‹(′*v literal*, ′*v literal*, *unit*) *annotated-lits* ⇒ ′*st* ⇒ ′*st*› **where**
  ‹*update-weight-information M S* =
    update-additional-info (*Some* (*lit-of* ‘# *mset M*), *snd* (*additional-info S*)) *S*›

lemma [*simp*]:
  ‹*trail* (*update-weight-information M′ S*) = *trail S*›
  ‹*clauses* (*update-weight-information M′ S*) = *clauses S*›
  ‹*clauses* (*update-additional-info c S*) = *clauses S*›
  ‹*additional-info* (*update-additional-info* (*w*, *oth*) *S*) = (*w*, *oth*)›
  using *update-additional-info*[*of S*] **unfolding** *update-weight-information-def*
  by (*auto simp*: *state*)

lemma *state-update-weight-information*: ‹*state S* = (*M*, *N*, *w*, *oth*) ⟹
      ∃ *w′*. *state* (*update-weight-information M′ S*) = (*M*, *N*, *w′*, *oth*)›
  apply (*auto simp*: *state*)
  apply (*auto simp*: *update-weight-information-def*)
  done

definition *weight* **where**
  ‹*weight S* = *fst* (*additional-info S*)›

lemma [*simp*]: ‹(*weight* (*update-weight-information M′ S*)) = *Some* (*lit-of* ‘# *mset M′*)›
  unfolding *weight-def* **by** (*auto simp*: *update-weight-information-def*)

We test here a slightly different decision. In the CDCL version, we renamed *additional-info* from the BNB version to avoid collisions. Here instead of renaming, we add the prefix *bnb.* to every name.

sublocale *bnb*: *bnb-ops* **where**
  trail = *trail* **and**

clauses = *clauses* **and**
tl-trail = *tl-trail* **and**
cons-trail = *cons-trail* **and**
state-eq = *state-eq* **and**
state = *state* **and**
weight = *weight* **and**
conflicting-clauses = *conflicting-clauses* **and**
is-improving-int = *is-improving-int* **and**
update-weight-information = *update-weight-information*
by *unfold-locales*

lemma *atms-of-mm-conflicting-clss-incl-init-clauses*:
‹*atms-of-mm* (*bnb.conflicting-clss S*) ⊆ *atms-of-mm* (*clauses S*)›
using *conflicting-clss-incl-init-clauses*[*of* ‹*clauses S*› ‹*weight S*›]
unfolding *bnb.conflicting-clss-def*
by *auto*

lemma *is-improving-conflicting-clss-update-weight-information*: ‹*bnb.is-improving M M′ S* ⟹
bnb.conflicting-clss S ⊆# *bnb.conflicting-clss* (*update-weight-information M′ S*)›
using *is-improving-conflicting-clss-update-weight-information*[*of M M′* ‹*clauses S*› ‹*weight S*›]
unfolding *bnb.conflicting-clss-def*
by (*auto simp*: *update-weight-information-def weight-def*)

lemma *conflicting-clss-update-weight-information-in2*:
assumes ‹*bnb.is-improving M M′ S*›
shows ‹*negate-ann-lits M′* ∈# *bnb.conflicting-clss* (*update-weight-information M′ S*)›
using *conflicting-clss-update-weight-information-in2*[*of M M′* ‹*clauses S*› ‹*weight S*›] *assms*
unfolding *bnb.conflicting-clss-def*
unfolding *bnb.conflicting-clss-def*
by (*auto simp*: *update-weight-information-def weight-def*)

lemma *state-additional-info′*:
‹*state S* = (*trail S*, *clauses S*, *weight S*, *bnb.additional-info S*)›
unfolding *additional-info-def* **by** (*cases* ‹*state S*›; *auto simp*: *state weight-def bnb.additional-info-def*)

sublocale *bnb*: *bnb* **where**
trail = *trail* **and**
clauses = *clauses* **and**
tl-trail = *tl-trail* **and**
cons-trail = *cons-trail* **and**
state-eq = *state-eq* **and**
state = *state* **and**
weight = *weight* **and**
conflicting-clauses = *conflicting-clauses* **and**
is-improving-int = *is-improving-int* **and**
update-weight-information = *update-weight-information*
apply *unfold-locales*
subgoal by *auto*
subgoal by (*rule state-eq-sym*)
subgoal by (*rule state-eq-trans*)
subgoal by (*auto dest*!: *state-eq-state*)
subgoal by (*rule cons-trail*)
subgoal by (*rule tl-trail*)
subgoal by (*rule state-update-weight-information*)

subgoal by (rule is-improving-conflicting-clss-update-weight-information)
subgoal by (rule conflicting-clss-update-weight-information-in2; assumption)
subgoal by (rule atms-of-mm-conflicting-clss-incl-init-clauses)
subgoal by (rule state-additional-info′)
done

lemma improve-model-still-model:
assumes
‹bnb.dpll$_W$-bound S T› **and**
all-struct: ‹dpll$_W$-all-inv (bnb.abs-state S)› **and**
ent: ‹set-mset I ⊨sm clauses S› ‹set-mset I ⊨sm bnb.conflicting-clss S› **and**
dist: ‹distinct-mset I› **and**
cons: ‹consistent-interp (set-mset I)› **and**
tot: ‹atms-of I = atms-of-mm (clauses S)› **and**
le: ‹Found (ϱ I) < ϱ′ (weight T)›
shows
‹set-mset I ⊨sm clauses T ∧ set-mset I ⊨sm bnb.conflicting-clss T›
using assms(1)
proof (cases rule: bnb.dpll$_W$-bound.cases)
case (update-info M M′) **note** imp = this(1) **and** T = this(2)
have atm-trail: ‹atms-of (lit-of ‘# mset (trail S)) ⊆ atms-of-mm (clauses S)› **and**
dist2: ‹distinct-mset (lit-of ‘# mset (trail S))› **and**
taut2: ‹¬ tautology (lit-of ‘# mset (trail S))›
using all-struct **unfolding** dpll$_W$-all-inv-def **by** (auto simp: lits-of-def atms-of-def
dest: no-dup-distinct no-dup-not-tautology)

have tot2: ‹total-over-m (set-mset I) (set-mset (clauses S))›
using tot[symmetric]
by (auto simp: total-over-m-def total-over-set-def atm-iff-pos-or-neg-lit)
have atm-trail: ‹atms-of (lit-of ‘# mset M′) ⊆ atms-of-mm (clauses S)› **and**
dist2: ‹distinct-mset (lit-of ‘# mset M′)› **and**
taut2: ‹¬ tautology (lit-of ‘# mset M′)›
using imp **by** (auto simp: lits-of-def atms-of-def is-improving-int-def
simple-clss-def)

have tot2: ‹total-over-m (set-mset I) (set-mset (clauses S))›
using tot[symmetric]
by (auto simp: total-over-m-def total-over-set-def atm-iff-pos-or-neg-lit)
have
‹set-mset I ⊨m conflicting-clauses (clauses S) (weight (update-weight-information M′ S))›
using entails-conflicting-clauses-if-le[of I ‹clauses S› M′ M ‹weight S›]
using T dist cons tot le imp **by** auto
then have ‹set-mset I ⊨m bnb.conflicting-clss (update-weight-information M′ S)›
by (auto simp: update-weight-information-def bnb.conflicting-clss-def)
then show ?thesis
using ent T **by** (auto simp: bnb.conflicting-clss-def state)
qed

lemma cdcl-bnb-still-model:
assumes
‹bnb.dpll$_W$-bnb S T› **and**
all-struct: ‹dpll$_W$-all-inv (bnb.abs-state S)› **and**
ent: ‹set-mset I ⊨sm clauses S› ‹set-mset I ⊨sm bnb.conflicting-clss S› **and**
dist: ‹distinct-mset I› **and**
cons: ‹consistent-interp (set-mset I)› **and**
tot: ‹atms-of I = atms-of-mm (clauses S)›

shows
‹(*set-mset I* ⊨*sm clauses T* ∧ *set-mset I* ⊨*sm bnb.conflicting-clss T*) ∨ *Found* (ϱ *I*) ≥ ϱ′ (*weight T*)›
using *assms*
proof (*induction rule*: *bnb.dpll*$_W$*-bnb.induct*)
case (*dpll S T*)
then show *?case* **using** *ent* **by** (*auto elim*!: *bnb.dpll*$_W$*-coreE simp*: *bnb.state*′*-def dpll-decide.simps dpll-backtrack.simps bnb.backtrack-opt.simps dpll-propagate.simps*)
next
case (*bnb S T*)
then show *?case*
using *improve-model-still-model*[*of S T I*] **using** *assms*(*2*−) **by** *auto*
qed

lemma *cdcl-bnb-larger-still-larger*:
assumes
‹*bnb.dpll*$_W$*-bnb S T*›
shows ‹ϱ′ (*weight S*) ≥ ϱ′ (*weight T*)›
using *assms* **apply** (*cases rule*: *bnb.dpll*$_W$*-bnb.cases*)
by (*auto simp*: *bnb.dpll*$_W$*-bound.simps is-improving-int-def bnb.dpll*$_W$*-core-same-weight*)

lemma *rtranclp-cdcl-bnb-still-model*:
assumes
st: ‹*bnb.dpll*$_W$*-bnb*** *S T*› **and**
all-struct: ‹*dpll*$_W$*-all-inv* (*bnb.abs-state S*)› **and**
ent: ‹(*set-mset I* ⊨*sm clauses S* ∧ *set-mset I* ⊨*sm bnb.conflicting-clss S*) ∨ *Found* (ϱ *I*) ≥ ϱ′ (*weight S*)› **and**
dist: ‹*distinct-mset I*› **and**
cons: ‹*consistent-interp* (*set-mset I*)› **and**
tot: ‹*atms-of I* = *atms-of-mm* (*clauses S*)›
shows
‹(*set-mset I* ⊨*sm clauses T* ∧ *set-mset I* ⊨*sm bnb.conflicting-clss T*) ∨ *Found* (ϱ *I*) ≥ ϱ′ (*weight T*)›
using *st*
proof (*induction rule*: *rtranclp-induct*)
case *base*
then show *?case*
using *ent* **by** *auto*
next
case (*step T U*) **note** *star* = *this*(*1*) **and** *st* = *this*(*2*) **and** *IH* = *this*(*3*)
have *1*: ‹*dpll*$_W$*-all-inv* (*bnb.abs-state T*)›
using *bnb.rtranclp-dpll*$_W$*-bnb-abs-state-all-inv*[*OF star all-struct*] **.**
have *3*: ‹*atms-of I* = *atms-of-mm* (*clauses T*)›
using *bnb.rtranclp-dpll*$_W$*-bnb-clauses*[*OF star*] *tot* **by** *auto*
show *?case*
using *cdcl-bnb-still-model*[*OF st 1 - - dist cons 3*] *IH*
cdcl-bnb-larger-still-larger[*OF st*]
order.trans **by** *blast*
qed

lemma *simple-clss-entailed-by-too-heavy-in-conflicting*:
‹*C* ∈# *mset-set* (*simple-clss* (*atms-of-mm* (*clauses S*))) ⟹
too-heavy-clauses (*clauses S*) (*weight S*) ⊨*pm*
(*C*) ⟹ *C* ∈# *bnb.conflicting-clss S*›
by (*auto simp*: *conflicting-clauses-def bnb.conflicting-clss-def*)

```
lemma can-always-improve:
  assumes
    ent: ‹trail S ⊨asm clauses S› and
    total: ‹total-over-m (lits-of-l (trail S)) (set-mset (clauses S))› and
    n-s: ‹(∀ C ∈# bnb.conflicting-clss S. ¬ trail S ⊨as CNot C)› and
    all-struct: ‹dpll_W-all-inv (bnb.abs-state S)›
   shows ‹Ex (bnb.dpll_W-bound S)›
proof –
  have H: ‹(lit-of `# mset (trail S)) ∈# mset-set (simple-clss (atms-of-mm (clauses S)))›
    ‹(lit-of `# mset (trail S)) ∈ simple-clss (atms-of-mm (clauses S))›
    ‹no-dup (trail S)›
    apply (subst finite-set-mset-mset-set[OF simple-clss-finite])
    using all-struct by (auto simp: simple-clss-def
        dpll_W-all-inv-def atms-of-def lits-of-def image-image clauses-def
      dest: no-dup-not-tautology no-dup-distinct)
  moreover have ‹trail S ⊨as CNot (pNeg (lit-of `# mset (trail S)))›
    by (auto simp: pNeg-def true-annots-true-cls-def-iff-negation-in-model lits-of-def)

  ultimately have le: ‹Found (ϱ (lit-of `# mset (trail S))) < ϱ' (weight S)›
    using n-s total not-entailed-too-heavy-clauses-ge[of ‹lit-of `# mset (trail S)› ‹clauses S› ‹weight S›]
     simple-clss-entailed-by-too-heavy-in-conflicting[of ‹pNeg (lit-of `# mset (trail S))› S]
    by (cases ‹¬ too-heavy-clauses (clauses S) (weight S) ⊨pm
       pNeg (lit-of `# mset (trail S))›)
     (auto simp:  lits-of-def
         conflicting-clauses-def clauses-def negate-ann-lits-pNeg-lit-of image-iff
         simple-clss-finite subset-iff
       dest: bspec[of - - ‹(lit-of `# mset (trail S))›] dest: total-over-m-atms-incl
          true-clss-cls-in too-heavy-clauses-contains-itself
          dest!: multi-member-split)
  have tr: ‹trail S ⊨asm clauses S›
    using ent by (auto simp: clauses-def)
  have tot': ‹total-over-m (lits-of-l (trail S)) (set-mset (clauses S))›
    using total all-struct by (auto simp: total-over-m-def total-over-set-def)
  have M': ‹ϱ (lit-of `# mset M') = ϱ (lit-of `# mset (trail S))›
    if ‹total-over-m (lits-of-l M') (set-mset (clauses S))› and
      incl: ‹mset (trail S) ⊆# mset M'› and
      ‹lit-of `# mset M' ∈ simple-clss (atms-of-mm (clauses S))›
      for M'
    proof –
      have [simp]: ‹lits-of-l M' = set-mset (lit-of `# mset M')›
        by (auto simp: lits-of-def)
      obtain A where A: ‹mset M' = A + mset (trail S)›
        using incl by (auto simp: mset-subset-eq-exists-conv)
      have M': ‹lits-of-l M' = lit-of ` set-mset A ∪ lits-of-l (trail S)›
        unfolding lits-of-def
        by (metis A image-Un set-mset-mset set-mset-union)
      have ‹mset M' = mset (trail S)›
        using that tot' total unfolding A total-over-m-alt-def
          apply (case-tac A)
        apply (auto simp: A simple-clss-def distinct-mset-add M' image-Un
            tautology-union mset-inter-empty-set-mset atms-of-def atms-of-s-def
            atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set image-image
            tautology-add-mset)
          by (metis (no-types, lifting) atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set
          lits-of-def subsetCE)
```

then show *?thesis*
using *total* **by** *auto*
qed
have ‹*bnb.is-improving* (*trail S*) (*trail S*) *S*›
if ‹*Found* (ϱ (*lit-of* '# *mset* (*trail S*))) < ϱ' (*weight S*)›
using *that total H tr tot' M'* **unfolding** *is-improving-int-def lits-of-def*
by *fast*
then show *?thesis*
using *bnb.dpll*$_W$*-bound.intros*[*of* ‹*trail S*› - *S* ‹*update-weight-information* (*trail S*) *S*›] *total H le*
by *fast*
qed

lemma *no-step-dpll*$_W$*-bnb-conflict*:
assumes
ns: ‹*no-step bnb.dpll*$_W$*-bnb S*› **and**
invs: ‹*dpll*$_W$*-all-inv* (*bnb.abs-state S*)›
shows ‹$\exists$ *C* $\in$# *clauses S* + *bnb.conflicting-clss S*. *trail S* $\models$*as CNot C*› (**is** *?A*) **and**
‹*count-decided* (*trail S*) = *0*› **and**
‹*unsatisfiable* (*set-mset* (*clauses S* + *bnb.conflicting-clss S*))›
apply (*rule bnb.no-step-dpll*$_W$*-bnb-conflict*[*OF* - *assms*])
subgoal using *can-always-improve* **by** *blast*
apply (*rule bnb.no-step-dpll*$_W$*-bnb-conflict*[*OF* - *assms*])
subgoal using *can-always-improve* **by** *blast*
apply (*rule bnb.no-step-dpll*$_W$*-bnb-conflict*[*OF* - *assms*])
subgoal using *can-always-improve* **by** *blast*
done

lemma *full-cdcl-bnb-stgy-larger-or-equal-weight*:
assumes
st: ‹*full bnb.dpll*$_W$*-bnb S T*› **and**
all-struct: ‹*dpll*$_W$*-all-inv* (*bnb.abs-state S*)› **and**
ent: ‹(*set-mset I* $\models$*sm clauses S* $\wedge$ *set-mset I* $\models$*sm bnb.conflicting-clss S*) $\vee$ *Found* (ϱ *I*) $\geq$ ϱ' (*weight S*)› **and**
dist: ‹*distinct-mset I*› **and**
cons: ‹*consistent-interp* (*set-mset I*)› **and**
tot: ‹*atms-of I* = *atms-of-mm* (*clauses S*)›
shows
‹*Found* (ϱ *I*) $\geq$ ϱ' (*weight T*)› **and**
‹*unsatisfiable* (*set-mset* (*clauses T* + *bnb.conflicting-clss T*))›
proof −
have *ns*: ‹*no-step bnb.dpll*$_W$*-bnb T*› **and**
st: ‹*bnb.dpll*$_W$*-bnb*** *S T*›
using *st* **unfolding** *full-def* **by** (*auto intro*:)
have *struct-T*: ‹*dpll*$_W$*-all-inv* (*bnb.abs-state T*)›
using *bnb.rtranclp-dpll*$_W$*-bnb-abs-state-all-inv*[*OF st all-struct*] **.**

have *atms-eq*: ‹*atms-of I* $\cup$ *atms-of-mm* (*bnb.conflicting-clss T*) = *atms-of-mm* (*clauses T*)›
using *atms-of-mm-conflicting-clss-incl-init-clauses*[*of T*]
bnb.rtranclp-dpll$_W$*-bnb-clauses*[*OF st*] *tot*
by *auto*

show ‹*unsatisfiable* (*set-mset* (*clauses T* + *bnb.conflicting-clss T*))›
using *no-step-dpll*$_W$*-bnb-conflict*[*of T*] *ns struct-T*
by *fast*
then have ‹¬*set-mset I* $\models$*sm clauses T* + *bnb.conflicting-clss T*›

using *dist cons* **by** *auto*
then have ‹*False*› **if** ‹*Found* (ϱ *I*) < ϱ' (*weight T*)›
using *ent that rtranclp-cdcl-bnb-still-model*[*OF st assms*(*2*−)]
bnb.rtranclp-dpll$_W$-bnb-clauses[*OF st*]
apply *simp*
using *leD* **by** *blast*

then show ‹*Found* (ϱ *I*) ≥ ϱ' (*weight T*)›
by *force*
qed

end

end
theory *DPLL-W-Partial-Encoding*
imports
DPLL-W-Optimal-Model
CDCL-W-Partial-Encoding
begin

context *optimal-encoding-ops*
begin

We use the following list to generate an upper bound of the derived trails by ODPLL: using lists makes it possible to use recursion. Using *inductive-set* does not work, because it is not possible to recurse on the arguments of a predicate.

The idea is similar to an earlier definition of *simple-clss*, although in that case, we went for recursion over the set of literals directly, via a choice in the recursive call.

definition *list-new-vars* :: ‹*'v list*› **where**
‹*list-new-vars* = (*SOME v. set v* = ΔΣ ∧ *distinct v*)›

lemma
set-list-new-vars: ‹*set list-new-vars* = ΔΣ› **and**
distinct-list-new-vars: ‹*distinct list-new-vars*› **and**
length-list-new-vars: ‹*length list-new-vars* = *card* ΔΣ›
using *someI*[*of* ‹λ*v. set v* = ΔΣ ∧ *distinct v*›]
unfolding *list-new-vars-def*[*symmetric*]
using *finite-Σ finite-distinct-list* **apply** *blast*
using *someI*[*of* ‹λ*v. set v* = ΔΣ ∧ *distinct v*›]
unfolding *list-new-vars-def*[*symmetric*]
using *finite-Σ finite-distinct-list* **apply** *blast*
using *someI*[*of* ‹λ*v. set v* = ΔΣ ∧ *distinct v*›]
unfolding *list-new-vars-def*[*symmetric*]
by (*metis distinct-card finite-Σ finite-distinct-list*)

fun *all-sound-trails* **where**
‹*all-sound-trails* [] = *simple-clss* (Σ − ΔΣ)› |
‹*all-sound-trails* (*L* # *M*) =
all-sound-trails M ∪ *add-mset* (*Pos* (*replacement-pos L*)) ‘ *all-sound-trails M*
∪ *add-mset* (*Pos* (*replacement-neg L*)) ‘ *all-sound-trails M*›

lemma *all-sound-trails-atms*:

```
  ‹set xs ⊆ ΔΣ ⟹
   C ∈ all-sound-trails xs ⟹
     atms-of C ⊆ Σ − ΔΣ ∪ replacement-pos ‘ set xs ∪ replacement-neg ‘ set xs›
  apply (induction xs arbitrary: C)
  subgoal by (auto simp: simple-clss-def)
  subgoal for x xs C
    apply (auto simp: tautology-add-mset)
    apply blast+
    done
  done

lemma all-sound-trails-distinct-mset:
  ‹set xs ⊆ ΔΣ ⟹ distinct xs ⟹
   C ∈ all-sound-trails xs ⟹
     distinct-mset C›
  using all-sound-trails-atms[of xs C]
  apply (induction xs arbitrary: C)
  subgoal by (auto simp: simple-clss-def)
  subgoal for x xs C
    apply clarsimp
    apply (auto simp: tautology-add-mset)
    apply (simp add: all-sound-trails-atms; fail)+
    apply (frule all-sound-trails-atms, assumption)
    apply (auto dest!: multi-member-split simp: subsetD)
    apply (simp add: all-sound-trails-atms; fail)+
    apply (frule all-sound-trails-atms, assumption)
    apply (auto dest!: multi-member-split simp: subsetD)
    apply (simp add: all-sound-trails-atms; fail)+
    done
  done

lemma all-sound-trails-tautology:
  ‹set xs ⊆ ΔΣ ⟹ distinct xs ⟹
   C ∈ all-sound-trails xs ⟹
     ¬tautology C›
  using all-sound-trails-atms[of xs C]
  apply (induction xs arbitrary: C)
  subgoal by (auto simp: simple-clss-def)
  subgoal for x xs C
    apply clarsimp
    apply (auto simp: tautology-add-mset)
    apply (simp add: all-sound-trails-atms; fail)+
    apply (frule all-sound-trails-atms, assumption)
    apply (auto dest!: multi-member-split simp: subsetD)
    apply (simp add: all-sound-trails-atms; fail)+
    apply (frule all-sound-trails-atms, assumption)
    apply (auto dest!: multi-member-split simp: subsetD)
    done
  done

lemma all-sound-trails-simple-clss:
  ‹set xs ⊆ ΔΣ ⟹ distinct xs ⟹
   all-sound-trails xs ⊆ simple-clss (Σ − ΔΣ ∪ replacement-pos ‘ set xs ∪ replacement-neg ‘ set xs)›
   using all-sound-trails-tautology[of xs]
     all-sound-trails-distinct-mset[of xs]
     all-sound-trails-atms[of xs]
```

by (*fastforce simp*: *simple-clss-def*)

lemma *in-all-sound-trails-inD*:
‹*set xs* ⊆ ΔΣ ⟹ *distinct xs* ⟹ *a* ∈ ΔΣ ⟹
add-mset (*Pos* ($a^{\mapsto 0}$)) *xa* ∈ *all-sound-trails xs* ⟹ *a* ∈ *set xs*›
using *all-sound-trails-simple-clss*[*of xs*]
apply (*auto simp*: *simple-clss-def*)
apply (*rotate-tac 3*)
apply (*frule set-mp*, *assumption*)
apply *auto*
done

lemma *in-all-sound-trails-inD'*:
‹*set xs* ⊆ ΔΣ ⟹ *distinct xs* ⟹ *a* ∈ ΔΣ ⟹
add-mset (*Pos* ($a^{\mapsto 1}$)) *xa* ∈ *all-sound-trails xs* ⟹ *a* ∈ *set xs*›
using *all-sound-trails-simple-clss*[*of xs*]
apply (*auto simp*: *simple-clss-def*)
apply (*rotate-tac 3*)
apply (*frule set-mp*, *assumption*)
apply *auto*
done

context
assumes [*simp*]: ‹*finite* Σ›
begin

lemma *all-sound-trails-finite*[*simp*]:
‹*finite* (*all-sound-trails xs*)›
by (*induction xs*)
(*auto intro*!: *simple-clss-finite finite*-Σ)

lemma *card-all-sound-trails*:
assumes ‹*set xs* ⊆ ΔΣ› **and** ‹*distinct xs*›
shows ‹*card* (*all-sound-trails xs*) = *card* (*simple-clss* (Σ − ΔΣ)) ∗ *3* ^ (*length xs*)›
using *assms*
apply (*induction xs*)
apply *auto*
apply (*subst card-Un-disjoint*)
apply (*auto simp*: *add-mset-eq-add-mset dest*: *in-all-sound-trails-inD*)
apply (*subst card-Un-disjoint*)
apply (*auto simp*: *add-mset-eq-add-mset dest*: *in-all-sound-trails-inD'*)
apply (*subst card-image*)
apply (*auto simp*: *inj-on-def*)
apply (*subst card-image*)
apply (*auto simp*: *inj-on-def*)
done

end

lemma *simple-clss-all-sound-trails*: ‹*simple-clss* (Σ − ΔΣ) ⊆ *all-sound-trails ys*›
apply (*induction ys*)
apply *auto*
done

lemma *all-sound-trails-decomp-in*:
assumes

‹C ⊆ ΔΣ› ‹C′ ⊆ ΔΣ› ‹C ∩ C′ = {}› ‹C ∪ C′ ⊆ *set xs*›
‹D ∈ *simple-clss* (Σ − ΔΣ)›
shows
‹(*Pos o replacement-pos*) ‘# *mset-set C* + (*Pos o replacement-neg*) ‘# *mset-set C′* + D ∈ *all-sound-trails xs*›
using *assms*
apply (*induction xs arbitrary*: *C C′ D*)
subgoal
using *simple-clss-all-sound-trails*[*of* ‹[]›]
by *auto*
subgoal premises *p* **for** *a xs C C′ D*
apply (*cases* ‹*a* ∈# *mset-set C*›)
subgoal
using *p*(*1*)[*of* ‹*C* − {*a*}› *C′ D*] *p*(*2*−)
finite-subset[*OF p*(*3*)]
apply −
apply (*subgoal-tac* ‹*finite C* ∧ *C* − {*a*} ⊆ ΔΣ ∧ *C′* ⊆ ΔΣ ∧ (*C* − {*a*}) ∩ *C′* = {} ∧ *C* − {*a*} ∪ *C′* ⊆ *set xs*›)
defer
apply (*auto simp*: *disjoint-iff-not-equal finite-subset*)[]
apply (*auto dest*!: *multi-member-split*)
by (*simp add*: *mset-set.remove*)
apply (*cases* ‹*a* ∈# *mset-set C′*›)
subgoal
using *p*(*1*)[*of C* ‹*C′* − {*a*}› *D*] *p*(*2*−)
finite-subset[*OF p*(*3*)]
apply −
apply (*subgoal-tac* ‹*finite C* ∧ *C* ⊆ ΔΣ ∧ *C′*− {*a*} ⊆ ΔΣ ∧ (*C*) ∩ (*C′*− {*a*}) = {} ∧ *C* ∪ *C′*− {*a*} ⊆ *set xs* ∧
C ⊆ *set xs* ∧ *C′* − {*a*} ⊆ *set xs*›)
defer
apply (*auto simp*: *disjoint-iff-not-equal finite-subset*)[]
apply (*auto dest*!: *multi-member-split*)
by (*simp add*: *mset-set.remove*)
subgoal
using *p*(*1*)[*of C C′ D*] *p*(*2*−)
finite-subset[*OF p*(*3*)]
apply −
apply (*subgoal-tac* ‹*finite C* ∧ *C* ⊆ ΔΣ ∧ *C′* ⊆ ΔΣ ∧ (*C*) ∩ (*C′*) = {} ∧ *C* ∪ *C′* ⊆ *set xs* ∧
C ⊆ *set xs* ∧ *C′* ⊆ *set xs*›)
defer
apply (*auto simp*: *disjoint-iff-not-equal finite-subset*)[]
by (*auto dest*!: *multi-member-split*)
done
done

lemma (**in** −)*image-union-subset-decomp*:
‹*f* ‘ (*C*) ⊆ *A* ∪ *B* ⟷ (∃ *A′ B′*. *f* ‘ *A′* ⊆ *A* ∧ *f* ‘ *B′* ⊆ *B* ∧ *C* = *A′* ∪ *B′* ∧ *A′* ∩ *B′* = {})›
apply (*rule iffI*)
apply (*rule exI*[*of* - ‹{*x* ∈ *C*. *f x* ∈ *A*}›])
apply (*rule exI*[*of* - ‹{*x* ∈ *C*. *f x* ∈ *B* ∧ *f x* ∉ *A*}›])
apply *auto*
done

lemma *in-all-sound-trails*:
assumes

‹⋀L. L ∈ ΔΣ ⟹ *Neg* (*replacement-pos L*) ∉# *C*›
‹⋀L. L ∈ ΔΣ ⟹ *Neg* (*replacement-neg L*) ∉# *C*›
‹⋀L. L ∈ ΔΣ ⟹ *Pos* (*replacement-pos L*) ∈# *C* ⟹ *Pos* (*replacement-neg L*) ∉# *C*›
‹*C* ∈ *simple-clss* (Σ − ΔΣ ∪ *replacement-pos* ‘ *set xs* ∪ *replacement-neg* ‘ *set xs*)› **and**
xs: ‹*set xs* ⊆ ΔΣ›
shows
‹*C* ∈ *all-sound-trails xs*›
proof −
have
atms: ‹*atms-of C* ⊆ (Σ − ΔΣ ∪ *replacement-pos* ‘ *set xs* ∪ *replacement-neg* ‘ *set xs*)› **and**
taut: ‹¬*tautology C*› **and**
dist: ‹*distinct-mset C*›
using *assms* **unfolding** *simple-clss-def*
by *blast*+

obtain *A′ B′ A′a B″* **where**
A′a: ‹*atm-of* ‘ *A′a* ⊆ Σ − ΔΣ› **and**
B″: ‹*atm-of* ‘ *B″* ⊆ *replacement-pos* ‘ *set xs*› **and**
‹*A′* = *A′a* ∪ *B″*› **and**
B′: ‹*atm-of* ‘ *B′* ⊆ *replacement-neg* ‘ *set xs*› **and**
C: ‹*set-mset C* = *A′a* ∪ *B″* ∪ *B′*› **and**
inter:
‹*B″* ∩ *B′* = {}›
‹*A′a* ∩ *B′* = {}›
‹*A′a* ∩ *B″* = {}›
using *atms* **unfolding** *atms-of-def*
apply (*subst* (*asm*)*image-union-subset-decomp*)
apply (*subst* (*asm*)*image-union-subset-decomp*)
by (*auto simp*: *Int-Un-distrib2*)

have *H*: ‹*f* ‘ *A* ⊆ *B* ⟹ *x* ∈ *A* ⟹ *f x* ∈ *B*› **for** *x A B f*
by *auto*
have [*simp*]: ‹*finite A′a*› ‹*finite B″*› ‹*finite B′*›
by (*metis C finite-Un finite-set-mset*)+
obtain *CB″ CB′* **where**
CB: ‹*CB′* ⊆ *set xs*› ‹*CB″* ⊆ *set xs*› **and**
decomp:
‹*atm-of* ‘ *B″* = *replacement-pos* ‘ *CB″*›
‹*atm-of* ‘ *B′* = *replacement-neg* ‘ *CB′*›
using *B′ B″* **by** (*auto simp*: *subset-image-iff*)
have *C*: ‹*C* =*mset-set B″* + *mset-set B′* + *mset-set A′a*›
using *inter*
apply (*subst distinct-set-mset-eq-iff*[*symmetric*, *OF dist*])
apply (*auto simp*: *C distinct-mset-mset-set simp flip*: *mset-set-Union*)
apply (*subst mset-set-Union*[*symmetric*])
using *inter*
apply *auto*
done
have *B″*: ‹*B″* = (*Pos*) ‘ (*atm-of* ‘ *B″*)›
using *assms*(*1*−*3*) *B″ xs A′a B″* **unfolding** *C*
apply (*auto simp*:)
apply (*frule H*, *assumption*)
apply (*case-tac x*)
apply *auto*
apply (*rule-tac x* = ‹*replacement-pos A*› **in** *imageI*)
apply (*auto simp add*: *rev-image-eqI*)

```
    apply (frule H, assumption)
    apply (case-tac xb)
    apply auto
    done
  have B': ‹B' = (Pos) ‘ (atm-of ‘ B')›
    using assms(1−3) B' xs A'a B' unfolding C
    apply (auto simp: )
    apply (frule H, assumption)
    apply (case-tac x)
    apply auto
    apply (rule-tac x = ‹replacement-neg A› in imageI)
    apply (auto simp add: rev-image-eqI)
    apply (frule H, assumption)
    apply (case-tac xb)
    apply auto
    done

  have simple: ‹mset-set A'a ∈ simple-clss (Σ − ΔΣ)›
    using assms A'a
    by (auto simp: simple-clss-def C atms-of-def image-Un tautology-decomp distinct-mset-mset-set)

  have [simp]: ‹finite (Pos ‘ replacement-pos ‘ CB'')›  ‹finite (Pos ‘ replacement-neg ‘ CB')›
    using B'' ‹finite B''› decomp ‹finite B'› B' apply auto
    by (meson CB(1) finite-Σ finite-imageI finite-subset xs)
  show ?thesis
    unfolding C
    apply (subst B'', subst B')
    unfolding decomp image-image
    apply (subst image-mset-mset-set[symmetric])
    subgoal
      using decomp xs B' B'' inter CB
      by (auto simp: C inj-on-def subset-iff)
    apply (subst image-mset-mset-set[symmetric])
    subgoal
      using decomp xs B' B'' inter CB
      by (auto simp: C inj-on-def subset-iff)
    apply (rule all-sound-trails-decomp-in[unfolded comp-def])
      using decomp xs B' B'' inter CB assms(3) simple
      unfolding C
      apply (auto simp: image-image)
      subgoal for x
        apply (subgoal-tac ‹x ∈ ΔΣ›)
        using assms(3)[of x]
        apply auto
        by (metis (mono-tags, lifting) B' ‹finite (Pos ‘ replacement-neg ‘ CB')› ‹finite B''› decomp(2)
         finite-set-mset-mset-set image-iff)
    done
qed

end


locale dpll-optimal-encoding-opt =
  dpll_W-state-optimal-weight trail clauses
    tl-trail cons-trail state-eq state ϱ update-additional-info +
  optimal-encoding-opt-ops Σ ΔΣ new-vars
```

for
trail :: ‹'st ⇒ 'v $dpll_W$-ann-lits› **and**
clauses :: ‹'st ⇒ 'v clauses› **and**
tl-trail :: ‹'st ⇒ 'st› **and**
cons-trail :: ‹'v $dpll_W$-ann-lit ⇒ 'st ⇒ 'st› **and**
state-eq :: ‹'st ⇒ 'st ⇒ bool› (**infix** ‹∼› *50*) **and**
state :: ‹'st ⇒ 'v $dpll_W$-ann-lits × 'v clauses × 'v clause option × 'b› **and**
update-additional-info :: ‹'v clause option × 'b ⇒ 'st ⇒ 'st› **and**
Σ ΔΣ :: ‹'v set› **and**
ϱ :: ‹'v clause ⇒ 'a :: {linorder}› **and**
new-vars :: ‹'v ⇒ 'v × 'v›
begin

end

locale *dpll-optimal-encoding* =
dpll-optimal-encoding-opt trail clauses
tl-trail cons-trail state-eq state
update-additional-info Σ ΔΣ ϱ *new-vars* +
optimal-encoding-ops
Σ ΔΣ
new-vars ϱ
for
trail :: ‹'st ⇒ 'v $dpll_W$-ann-lits› **and**
clauses :: ‹'st ⇒ 'v clauses› **and**
tl-trail :: ‹'st ⇒ 'st› **and**
cons-trail :: ‹'v $dpll_W$-ann-lit ⇒ 'st ⇒ 'st› **and**
state-eq :: ‹'st ⇒ 'st ⇒ bool› (**infix** ‹∼› *50*) **and**
state :: ‹'st ⇒ 'v $dpll_W$-ann-lits × 'v clauses × 'v clause option × 'b› **and**
update-additional-info :: ‹'v clause option × 'b ⇒ 'st ⇒ 'st› **and**
Σ ΔΣ :: ‹'v set› **and**
ϱ :: ‹'v clause ⇒ 'a :: {linorder}› **and**
new-vars :: ‹'v ⇒ 'v × 'v›
begin

inductive *odecide* :: ‹'st ⇒ 'st ⇒ bool› **where**
odecide-noweight: ‹*odecide S T*›
if
‹*undefined-lit* (*trail S*) *L*› **and**
‹*atm-of L* ∈ *atms-of-mm* (*clauses S*)› **and**
‹*T* ∼ *cons-trail* (*Decided L*) *S*› **and**
‹*atm-of L* ∈ Σ − ΔΣ› |
odecide-replacement-pos: ‹*odecide S T*›
if
‹*undefined-lit* (*trail S*) (*Pos* (*replacement-pos L*))› **and**
‹*T* ∼ *cons-trail* (*Decided* (*Pos* (*replacement-pos L*))) *S*› **and**
‹*L* ∈ ΔΣ› |
odecide-replacement-neg: ‹*odecide S T*›
if
‹*undefined-lit* (*trail S*) (*Pos* (*replacement-neg L*))› **and**
‹*T* ∼ *cons-trail* (*Decided* (*Pos* (*replacement-neg L*))) *S*› **and**
‹*L* ∈ ΔΣ›

inductive-cases *odecideE*: ‹*odecide S T*›

inductive *dpll-conflict* :: ‹'st ⇒ 'st ⇒ bool› **where**
‹dpll-conflict S S›
if ‹C ∈# clauses S› **and**
‹trail S ⊨as CNot C›

inductive odpll$_W$-core-stgy :: ‹'st ⇒ 'st ⇒ bool› **for** S T **where**
propagate: ‹dpll-propagate S T ⟹ odpll$_W$-core-stgy S T› |
decided: ‹odecide S T ⟹ no-step dpll-propagate S ⟹ odpll$_W$-core-stgy S T › |
backtrack: ‹dpll-backtrack S T ⟹ odpll$_W$-core-stgy S T› |
backtrack-opt: ‹bnb.backtrack-opt S T ⟹ odpll$_W$-core-stgy S T›

lemma odpll$_W$-core-stgy-clauses:
‹odpll$_W$-core-stgy S T ⟹ clauses T = clauses S›
by (induction rule: odpll$_W$-core-stgy.induct)
(auto simp: dpll-propagate.simps odecide.simps dpll-backtrack.simps
bnb.backtrack-opt.simps)

lemma rtranclp-odpll$_W$-core-stgy-clauses:
‹odpll$_W$-core-stgy** S T ⟹ clauses T = clauses S›
by (induction rule: rtranclp-induct)
(auto dest: odpll$_W$-core-stgy-clauses)

inductive odpll$_W$-bnb-stgy :: ‹'st ⇒ 'st ⇒ bool› **for** S T :: 'st **where**
dpll:
‹odpll$_W$-bnb-stgy S T›
if ‹odpll$_W$-core-stgy S T› |
bnb:
‹odpll$_W$-bnb-stgy S T›
if ‹bnb.dpll$_W$-bound S T›

lemma odpll$_W$-bnb-stgy-clauses:
‹odpll$_W$-bnb-stgy S T ⟹ clauses T = clauses S›
by (induction rule: odpll$_W$-bnb-stgy.induct)
(auto simp: bnb.dpll$_W$-bound.simps dest: odpll$_W$-core-stgy-clauses)

lemma rtranclp-odpll$_W$-bnb-stgy-clauses:
‹odpll$_W$-bnb-stgy** S T ⟹ clauses T = clauses S›
by (induction rule: rtranclp-induct)
(auto dest: odpll$_W$-bnb-stgy-clauses)

lemma odecide-dpll-decide-iff:
assumes ‹clauses S = penc N› ‹atms-of-mm N = Σ›
shows ‹odecide S T ⟹ dpll-decide S T›
‹dpll-decide S T ⟹ Ex(odecide S)›
using assms atms-of-mm-penc-subset2[of N] ΔΣ-Σ
unfolding odecide.simps dpll-decide.simps
apply (auto simp: odecide.simps dpll-decide.simps)
apply (metis defined-lit-Pos-atm-iff state-eq-ref)+
done

lemma
assumes ‹clauses S = penc N› ‹atms-of-mm N = Σ›
shows
odpll$_W$-core-stgy-dpll$_W$-core-stgy: ‹odpll$_W$-core-stgy S T ⟹ bnb.dpll$_W$-core-stgy S T›

using *odecide-dpll-decide-iff*[*OF assms*]
by (*auto simp*: *odpll*$_W$*-core-stgy.simps bnb.dpll*$_W$*-core-stgy.simps*)

lemma
assumes ‹*clauses S* = *penc N*› ‹*atms-of-mm N* = Σ›
shows
odpll$_W$*-bnb-stgy-dpll*$_W$*-bnb-stgy*: ‹*odpll*$_W$*-bnb-stgy S T* ⟹ *bnb.dpll*$_W$*-bnb S T*›
using *odecide-dpll-decide-iff*[*OF assms*]
by (*auto simp*: *odpll*$_W$*-bnb-stgy.simps bnb.dpll*$_W$*-bnb.simps dest*: *odpll*$_W$*-core-stgy-dpll*$_W$*-core-stgy*[*OF assms*]
bnb.dpll$_W$*-core-stgy-dpll*$_W$*-core*)

lemma
assumes ‹*clauses S* = *penc N*› **and** [*simp*]: ‹*atms-of-mm N* = Σ›
shows
rtranclp-odpll$_W$*-bnb-stgy-dpll*$_W$*-bnb-stgy*: ‹*odpll*$_W$*-bnb-stgy*** *S T* ⟹ *bnb.dpll*$_W$*-bnb*** *S T*›
using *assms*(*1*) **apply** −
apply (*induction rule*: *rtranclp-induct*)
subgoal by *auto*
subgoal for *T U*
using *odpll*$_W$*-bnb-stgy-dpll*$_W$*-bnb-stgy*[*of T N U*] *rtranclp-odpll*$_W$*-bnb-stgy-clauses*[*of S T*]
by *auto*
done

lemma *no-step-odpll*$_W$*-core-stgy-no-step-dpll*$_W$*-core-stgy*:
assumes ‹*clauses S* = *penc N*› **and** [*simp*]:‹*atms-of-mm N* = Σ›
shows
‹*no-step odpll*$_W$*-core-stgy S* ⟷ *no-step bnb.dpll*$_W$*-core-stgy S*›
using *odecide-dpll-decide-iff*[*of S*, *OF assms*]
by (*auto simp*: *odpll*$_W$*-core-stgy.simps bnb.dpll*$_W$*-core-stgy.simps*)

lemma *no-step-odpll*$_W$*-bnb-stgy-no-step-dpll*$_W$*-bnb*:
assumes ‹*clauses S* = *penc N*› **and** [*simp*]:‹*atms-of-mm N* = Σ›
shows
‹*no-step odpll*$_W$*-bnb-stgy S* ⟷ *no-step bnb.dpll*$_W$*-bnb S*›
using *no-step-odpll*$_W$*-core-stgy-no-step-dpll*$_W$*-core-stgy*[*of S*, *OF assms*] *bnb.no-step-stgy-iff*
by (*auto simp*: *odpll*$_W$*-bnb-stgy.simps bnb.dpll*$_W$*-bnb.simps dest*: *odpll*$_W$*-core-stgy-dpll*$_W$*-core-stgy*[*OF assms*]
bnb.dpll$_W$*-core-stgy-dpll*$_W$*-core*)

lemma *full-odpll*$_W$*-core-stgy-full-dpll*$_W$*-core-stgy*:
assumes ‹*clauses S* = *penc N*› **and** [*simp*]:‹*atms-of-mm N* = Σ›
shows
‹*full odpll*$_W$*-bnb-stgy S T* ⟹ *full bnb.dpll*$_W$*-bnb S T*›
using *no-step-odpll*$_W$*-bnb-stgy-no-step-dpll*$_W$*-bnb*[*of T*, *OF - assms*(*2*)]
rtranclp-odpll$_W$*-bnb-stgy-clauses*[*of S T*, *symmetric*, *unfolded assms*]
rtranclp-odpll$_W$*-bnb-stgy-dpll*$_W$*-bnb-stgy*[*of S N T*, *OF assms*]
by (*auto simp*: *full-def*)

lemma *decided-cons-eq-append-decide-cons*:
Decided L # *Ms* = *M'* @ *Decided K* # *M* ⟷
(*L* = *K* ∧ *Ms* = *M* ∧ *M'* = []) ∨
(*hd M'* = *Decided L* ∧ *Ms* = *tl M'* @ *Decided K* # *M* ∧ *M'* ≠ [])
by (*cases M'*)
auto

lemma *no-step-dpll-backtrack-iff*:
‹*no-step dpll-backtrack S* ⟷ (*count-decided* (*trail S*) = *0* ∨ (∀ *C* ∈# *clauses S*. ¬*trail S* ⊨*as CNot C*))›
using *backtrack-snd-empty-not-decided*[*of* ‹*trail S*›] *backtrack-split-list-eq*[*of* ‹*trail S*›, *symmetric*]
apply (*cases* ‹*backtrack-split* (*trail S*)›; *cases* ‹*snd*(*backtrack-split* (*trail S*))›)
by (*auto simp*: *dpll-backtrack.simps count-decided-0-iff*)

lemma *no-step-dpll-conflict*:
‹*no-step dpll-conflict S* ⟷ (∀ *C* ∈# *clauses S*. ¬*trail S* ⊨*as CNot C*)›
by (*auto simp*: *dpll-conflict.simps*)

definition *no-smaller-propa* :: ‹′*st* ⇒ *bool*› **where**
no-smaller-propa (*S* ::′*st*) ⟷
(∀ *M K M′ D L*. *trail S* = *M′* @ *Decided K* # *M* ⟶ *add-mset L D* ∈# *clauses S* ⟶ *undefined-lit M L* ⟶ ¬*M* ⊨*as CNot D*)

lemma [*simp*]: ‹*T* ∼ *S* ⟹ *no-smaller-propa T* = *no-smaller-propa S*›
by (*auto simp*: *no-smaller-propa-def*)

lemma *no-smaller-propa-cons-trail*[*simp*]:
‹*no-smaller-propa* (*cons-trail* (*Propagated L C*) *S*) ⟷ *no-smaller-propa S*›
‹*no-smaller-propa* (*update-weight-information M′ S*) ⟷ *no-smaller-propa S*›
by (*force simp*: *no-smaller-propa-def* $cdcl_W$*-restart-mset.propagated-cons-eq-append-decide-cons*)+

lemma *no-smaller-propa-cons-trail-decided*[*simp*]:
‹*no-smaller-propa S* ⟹ *no-smaller-propa* (*cons-trail* (*Decided L*) *S*) ⟷ (∀ *L C*. *add-mset L C* ∈# *clauses S* ⟶ *undefined-lit* (*trail S*)*L* ⟶ ¬*trail S* ⊨*as CNot C*)›
by (*auto simp*: *no-smaller-propa-def* $cdcl_W$*-restart-mset.propagated-cons-eq-append-decide-cons decided-cons-eq-append-decide-cons*)

lemma *no-step-dpll-propagate-iff*:
‹*no-step dpll-propagate S* ⟷ (∀ *L C*. *add-mset L C* ∈# *clauses S* ⟶ *undefined-lit* (*trail S*)*L* ⟶ ¬*trail S* ⊨*as CNot C*)›
by (*auto simp*: *dpll-propagate.simps*)

lemma *count-decided-0-no-smaller-propa*: ‹*count-decided* (*trail S*) = *0* ⟹ *no-smaller-propa S*›
by (*auto simp*: *no-smaller-propa-def*)

lemma *no-smaller-propa-backtrack-split*:
‹*no-smaller-propa S* ⟹
backtrack-split (*trail S*) = (*M′*, *L* # *M*) ⟹
no-smaller-propa (*reduce-trail-to M S*)›
using *backtrack-split-list-eq*[*of* ‹*trail S*›, *symmetric*]
by (*auto simp*: *no-smaller-propa-def*)

lemma $odpll_W$*-core-stgy-no-smaller-propa*:
‹$odpll_W$*-core-stgy S T* ⟹ *no-smaller-propa S* ⟹ *no-smaller-propa T*›
using *no-step-dpll-backtrack-iff*[*of S*] **apply** −
by (*induction rule*: $odpll_W$*-core-stgy.induct*)
(*auto 5 5 simp*: $cdcl_W$*-restart-mset.propagated-cons-eq-append-decide-cons count-decided-0-no-smaller-propa dpll-propagate.simps dpll-decide.simps odecide.simps decided-cons-eq-append-decide-cons bnb.backtrack-opt.simps dpll-backtrack.simps no-step-dpll-conflict no-smaller-propa-backtrack-split*)

lemma $odpll_W$*-bound-stgy-no-smaller-propa*: ‹*bnb.*$dpll_W$*-bound S T* ⟹ *no-smaller-propa S* ⟹ *no-smaller-propa T*›

by (*auto simp*: *cdcl*$_W$*-restart-mset.propagated-cons-eq-append-decide-cons count-decided-0-no-smaller-propa dpll-propagate.simps dpll-decide.simps odecide.simps decided-cons-eq-append-decide-cons bnb.dpll*$_W$*-bound.simps bnb.backtrack-opt.simps dpll-backtrack.simps no-step-dpll-conflict no-smaller-propa-backtrack-split*)

lemma *odpll*$_W$*-bnb-stgy-no-smaller-propa*:
‹*odpll*$_W$*-bnb-stgy S T* ⟹ *no-smaller-propa S* ⟹ *no-smaller-propa T*›
by (*induction rule*: *odpll*$_W$*-bnb-stgy.induct*)
(*auto simp*: *odpll*$_W$*-core-stgy-no-smaller-propa odpll*$_W$*-bound-stgy-no-smaller-propa*)

lemma *filter-disjount-union*:
‹(⋀*x*. *x* ∈ *set xs* ⟹ *P x* ⟹ ¬*Q x*) ⟹
length (*filter P xs*) + *length* (*filter Q xs*) =
length (*filter* (λ*x*. *P x* ∨ *Q x*) *xs*)›
by (*induction xs*) *auto*

lemma *Collect-req-remove1*:
‹{*a* ∈ *A*. *a* ≠ *b* ∧ *P a*} = (*if P b then Set.remove b* {*a* ∈ *A*. *P a*} *else* {*a* ∈ *A*. *P a*})› **and**
Collect-req-remove2:
‹{*a* ∈ *A*. *b* ≠ *a* ∧ *P a*} = (*if P b then Set.remove b* {*a* ∈ *A*. *P a*} *else* {*a* ∈ *A*. *P a*})›
by *auto*

lemma *card-remove*:
‹*card* (*Set.remove a A*) = (*if a* ∈ *A then card A* − *1 else card A*)›
by (*auto simp*: *Set.remove-def*)

lemma *isabelle-should-do-that-automatically*: ‹*Suc* (*a* − *Suc 0*) = *a* ⟷ *a* ≥ *1*›
by *auto*
lemma *distinct-count-list-if*: ‹*distinct xs* ⟹ *count-list xs x* = (*if x* ∈ *set xs then 1 else 0*)›
by (*induction xs*) *auto*

abbreviation (*input*) *cut-and-complete-trail* :: ‹*'st* ⇒ *-*› **where**
‹*cut-and-complete-trail S* ≡ *trail S*›

inductive *odpll*$_W$*-core-stgy-count* :: ‹*'st* × *-* ⇒ *'st* × *-* ⇒ *bool*› **where**
propagate: ‹*dpll-propagate S T* ⟹ *odpll*$_W$*-core-stgy-count* (*S*, *C*) (*T*, *C*)› |
decided: ‹*odecide S T* ⟹ *no-step dpll-propagate S* ⟹ *odpll*$_W$*-core-stgy-count* (*S*, *C*) (*T*, *C*) › |
backtrack: ‹*dpll-backtrack S T* ⟹ *odpll*$_W$*-core-stgy-count* (*S*, *C*) (*T*, *add-mset* (*cut-and-complete-trail S*) *C*)› |
backtrack-opt: ‹*bnb.backtrack-opt S T* ⟹ *odpll*$_W$*-core-stgy-count* (*S*, *C*) (*T*, *add-mset* (*cut-and-complete-trail S*) *C*)›

inductive *odpll*$_W$*-bnb-stgy-count* :: ‹*'st* × *-* ⇒ *'st* × *-* ⇒ *bool*› **where**
dpll:
‹*odpll*$_W$*-bnb-stgy-count S T*›
if ‹*odpll*$_W$*-core-stgy-count S T*› |
bnb:
‹*odpll*$_W$*-bnb-stgy-count* (*S*, *C*) (*T*, *C*)›
if ‹*bnb.dpll*$_W$*-bound S T*›

lemma *odpll*$_W$*-core-stgy-countD*:
‹*odpll*$_W$*-core-stgy-count S T* ⟹ *odpll*$_W$*-core-stgy* (*fst S*) (*fst T*)›

‹$odpll_W$-core-stgy-count S T ⟹ snd S ⊆# snd T›
by (induction rule: $odpll_W$-core-stgy-count.induct; auto intro: $odpll_W$-core-stgy.intros)+

lemma $odpll_W$-bnb-stgy-countD:
‹$odpll_W$-bnb-stgy-count S T ⟹ $odpll_W$-bnb-stgy (fst S) (fst T)›
‹$odpll_W$-bnb-stgy-count S T ⟹ snd S ⊆# snd T›
by (induction rule: $odpll_W$-bnb-stgy-count.induct; auto dest: $odpll_W$-core-stgy-countD intro: $odpll_W$-bnb-stgy.intros)+

lemma rtranclp-$odpll_W$-bnb-stgy-countD:
‹$odpll_W$-bnb-stgy-count** S T ⟹ $odpll_W$-bnb-stgy** (fst S) (fst T)›
‹$odpll_W$-bnb-stgy-count** S T ⟹ snd S ⊆# snd T›
by (induction rule: rtranclp-induct; auto dest: $odpll_W$-bnb-stgy-countD)+

lemmas $odpll_W$-core-stgy-count-induct = $odpll_W$-core-stgy-count.induct[of ‹(S, n)› ‹(T, m)› **for** S n T m, split-format(complete), OF dpll-optimal-encoding-axioms,
consumes 1]

definition conflict-clauses-are-entailed :: ‹'st × - ⇒ bool› **where**
‹conflict-clauses-are-entailed =
(λ(S, Cs). ∀ C ∈# Cs. (∃ M' K M M''. trail S = M' @ Propagated K () # M ∧ C = M'' @ Decided (−K) # M))›

definition conflict-clauses-are-entailed2 :: ‹'st × ('v literal, 'v literal, unit) annotated-lits multiset ⇒ bool› **where**
‹conflict-clauses-are-entailed2 =
(λ(S, Cs). ∀ C ∈# Cs. ∀ C' ∈# remove1-mset C Cs. (∃ L. Decided L ∈ set C ∧ Propagated (−L) () ∈ set C') ∨
(∃ L. Propagated (L) () ∈ set C ∧ Decided (−L) ∈ set C'))›

lemma propagated-cons-eq-append-propagated-cons:
‹Propagated L () # M = M' @ Propagated K () # Ma ⟷
(M' = [] ∧ K = L ∧ M = Ma) ∨
(M' ≠ [] ∧ hd M' = Propagated L () ∧ M = tl M' @ Propagated K () # Ma)›
by (cases M')
auto

lemma $odpll_W$-core-stgy-count-conflict-clauses-are-entailed:
assumes
‹$odpll_W$-core-stgy-count S T› **and**
‹conflict-clauses-are-entailed S›
shows
‹conflict-clauses-are-entailed T›
using assms
apply (induction rule: $odpll_W$-core-stgy-count.induct)
subgoal
apply (auto simp: dpll-propagate.simps conflict-clauses-are-entailed-def
$cdcl_W$-restart-mset.propagated-cons-eq-append-decide-cons)
by (metis append-Cons)
subgoal for S T
apply (auto simp: odecide.simps conflict-clauses-are-entailed-def
dest!: multi-member-split intro: exI[of - ‹Decided - # -›])
by (metis append-Cons)+
subgoal for S T C
using backtrack-split-list-eq[of ‹trail S›, symmetric]

backtrack-split-snd-hd-decided[*of* ‹*trail S*›]
apply (*auto simp*: *dpll-backtrack.simps conflict-clauses-are-entailed-def*
propagated-cons-eq-append-propagated-cons is-decided-def append-eq-append-conv2
eq-commute[*of* - ‹*Propagated* - () # -›] *conj-disj-distribR ex-disj-distrib*
$cdcl_W$*-restart-mset.propagated-cons-eq-append-decide-cons* $dpll_W$*-all-inv-def*
dest!: *multi-member-split*
simp del: *backtrack-split-list-eq*
)
apply (*case-tac us*)
by *force*+
subgoal for *S T C*
using *backtrack-split-list-eq*[*of* ‹*trail S*›, *symmetric*]
backtrack-split-snd-hd-decided[*of* ‹*trail S*›]
apply (*auto simp*: *bnb.backtrack-opt.simps conflict-clauses-are-entailed-def*
propagated-cons-eq-append-propagated-cons is-decided-def append-eq-append-conv2
eq-commute[*of* - ‹*Propagated* - () # -›] *conj-disj-distribR ex-disj-distrib*
$cdcl_W$*-restart-mset.propagated-cons-eq-append-decide-cons*
$dpll_W$*-all-inv-def*
dest!: *multi-member-split*
simp del: *backtrack-split-list-eq*
)
apply (*case-tac us*)
by *force*+
done

lemma *odpll*$_W$*-bnb-stgy-count-conflict-clauses-are-entailed*:
assumes
‹*odpll*$_W$*-bnb-stgy-count S T*› **and**
‹*conflict-clauses-are-entailed S*›
shows
‹*conflict-clauses-are-entailed T*›
using *assms odpll*$_W$*-core-stgy-count-conflict-clauses-are-entailed*[*of S T*]
apply (*auto simp*: *odpll*$_W$*-bnb-stgy-count.simps*)
apply (*auto simp*: *conflict-clauses-are-entailed-def*
bnb.dpll$_W$*-bound.simps*)
done

lemma *odpll*$_W$*-core-stgy-count-no-dup-clss*:
assumes
‹*odpll*$_W$*-core-stgy-count S T*› **and**
‹∀ *C* ∈# *snd S*. *no-dup C*› **and**
invs: ‹*dpll*$_W$*-all-inv* (*bnb.abs-state* (*fst S*))›
shows
‹∀ *C* ∈# *snd T*. *no-dup C*›
using *assms*
by (*induction rule*: *odpll*$_W$*-core-stgy-count.induct*)
(*auto simp*: *dpll*$_W$*-all-inv-def*)

lemma *odpll*$_W$*-bnb-stgy-count-no-dup-clss*:
assumes
‹*odpll*$_W$*-bnb-stgy-count S T*› **and**
‹∀ *C* ∈# *snd S*. *no-dup C*› **and**
invs: ‹*dpll*$_W$*-all-inv* (*bnb.abs-state* (*fst S*))›
shows
‹∀ *C* ∈# *snd T*. *no-dup C*›

```
  using assms
  by (induction rule: odpll_W-bnb-stgy-count.induct)
    (auto simp: dpll_W-all-inv-def
      bnb.dpll_W-bound.simps dest!: odpll_W-core-stgy-count-no-dup-clss)

lemma backtrack-split-conflict-clauses-are-entailed-itself:
  assumes
    ‹backtrack-split (trail S) = (M', L # M)› and
    invs: ‹dpll_W-all-inv (bnb.abs-state S)›
  shows ‹¬ conflict-clauses-are-entailed
           (S, add-mset (trail S) C)› (is ‹¬ ?A›)
proof
  assume ?A
  then obtain M' K Ma where
    tr: ‹trail S = M' @ Propagated K () # Ma› and
    ‹add-mset (− K) (lit-of `# mset Ma) ⊆#
       add-mset (lit-of L) (lit-of `# mset M)›
    by (clarsimp simp: conflict-clauses-are-entailed-def)

  then have ‹−K ∈# add-mset (lit-of L) (lit-of `# mset M)›
    by (meson member-add-mset mset-subset-eqD)
  then have ‹−K ∈# lit-of `# mset (trail S)›
    using backtrack-split-list-eq[of ‹trail S›, symmetric] assms(1)
    by auto
  moreover have ‹K ∈# lit-of `# mset (trail S)›
    by (auto simp: tr)
  ultimately show False using invs unfolding dpll_W-all-inv-def
    by (auto simp add: no-dup-cannot-not-lit-and-uminus uminus-lit-swap)
qed



lemma odpll_W-core-stgy-count-distinct-mset:
  assumes
    ‹odpll_W-core-stgy-count S T› and
    ‹conflict-clauses-are-entailed S› and
    ‹distinct-mset (snd S)› and
    invs: ‹dpll_W-all-inv (bnb.abs-state (fst S))›
  shows
    ‹distinct-mset (snd T)›
  using assms(1,2,3,4) odpll_W-core-stgy-count-conflict-clauses-are-entailed[OF assms(1,2)]
  apply (induction rule: odpll_W-core-stgy-count.induct)
  subgoal
    by (auto simp: dpll-propagate.simps conflict-clauses-are-entailed-def
      cdcl_W-restart-mset.propagated-cons-eq-append-decide-cons)
  subgoal
    by (auto simp:)
  subgoal for S T C
    by (clarsimp simp: dpll-backtrack.simps backtrack-split-conflict-clauses-are-entailed-itself
      dest!: multi-member-split)
  subgoal for S T C
    by (clarsimp simp: bnb.backtrack-opt.simps backtrack-split-conflict-clauses-are-entailed-itself
      dest!: multi-member-split)
  done

lemma odpll_W-bnb-stgy-count-distinct-mset:
```

assumes
‹$odpll_W$*-bnb-stgy-count S T*› **and**
‹*conflict-clauses-are-entailed S*› **and**
‹*distinct-mset* (*snd S*)› **and**
invs: ‹$dpll_W$*-all-inv* (*bnb.abs-state* (*fst S*))›
shows
‹*distinct-mset* (*snd T*)›
using *assms* $odpll_W$*-core-stgy-count-distinct-mset*[*OF - assms*(*2*−), *of T*]
by (*auto simp*: $odpll_W$*-bnb-stgy-count.simps*)

lemma $odpll_W$*-core-stgy-count-conflict-clauses-are-entailed2*:
assumes
‹$odpll_W$*-core-stgy-count S T*› **and**
‹*conflict-clauses-are-entailed S*› **and**
‹*conflict-clauses-are-entailed2 S*› **and**
‹*distinct-mset* (*snd S*)› **and**
invs: ‹$dpll_W$*-all-inv* (*bnb.abs-state* (*fst S*))›
shows
‹*conflict-clauses-are-entailed2 T*›
using *assms*
proof (*induction rule*: $odpll_W$*-core-stgy-count.induct*)
case (*propagate S T C*)
then show *?case*
by (*auto simp*: *dpll-propagate.simps conflict-clauses-are-entailed2-def*)
next
case (*decided S T C*)
then show *?case*
by (*auto simp*: *dpll-decide.simps conflict-clauses-are-entailed2-def*)
next
case (*backtrack S T C*) **note** *bt* = *this*(*1*) **and** *ent* = *this*(*2*) **and** *ent2* = *this*(*3*) **and** *dist* = *this*(*4*)
and *invs* = *this*(*5*)
let *?M* = ‹(*cut-and-complete-trail S*)›
have ‹*conflict-clauses-are-entailed* (*T*, *add-mset ?M C*)› **and**
dist′: ‹*distinct-mset* (*add-mset ?M C*)›
using $odpll_W$*-core-stgy-count-conflict-clauses-are-entailed*[*OF - ent*, *of* ‹(*T*, *add-mset ?M C*)›]
$odpll_W$*-core-stgy-count-distinct-mset*[*OF - ent dist invs*, *of* ‹(*T*, *add-mset ?M C*)›]
bt **by** (*auto dest*!: $odpll_W$*-core-stgy-count.intros*(*3*)[*of S T C*])

obtain *M1 K M2* **where**
spl: ‹*backtrack-split* (*trail S*) = (*M2*, *Decided K* # *M1*)›
using *bt backtrack-split-snd-hd-decided*[*of* ‹*trail S*›]
by (*cases* ‹*hd* (*snd* (*backtrack-split* (*trail S*)))›) (*auto simp*: *dpll-backtrack.simps*)
have *has-dec*: ‹∃ *l*∈*set* (*trail S*). *is-decided l*›
using *bt* **apply** (*auto simp*: *dpll-backtrack.simps*)
using *bt count-decided-0-iff no-step-dpll-backtrack-iff* **by** *blast*

let *?P* = ‹λ*Ca C′*.
(∃ *L*. *Decided L* ∈ *set Ca* ∧ *Propagated* (− *L*) () ∈ *set C′*) ∨
(∃ *L*. *Propagated L* () ∈ *set Ca* ∧ *Decided* (− *L*) ∈ *set C′*)›
have ‹∀ *C′*∈#*remove1-mset ?M C*. *?P ?M C′*›
proof
fix *C′*
assume ‹*C′*∈#*remove1-mset ?M C*›
then have ‹*C′* ∈# *C*› **and** ‹*C′* ≠ *?M*›
using *dist′* **by** *auto*

then obtain *M' L M M''* **where**
‹*trail S* = *M'* @ *Propagated L* () # *M*› **and**
‹*C'* = *M''* @ *Decided* (− *L*) # *M*›
using *ent* **unfolding** *conflict-clauses-are-entailed-def*
by *auto*
then show ‹*?P ?M C'*›
using *backtrack-split-some-is-decided-then-snd-has-hd*[*of* ‹*trail S*›, *OF has-dec*]
spl backtrack-split-list-eq[*of* ‹*trail S*›, *symmetric*]
by (*clarsimp simp*: *conj-disj-distribR ex-disj-distrib decided-cons-eq-append-decide-cons*
$cdcl_W$*-restart-mset.propagated-cons-eq-append-decide-cons propagated-cons-eq-append-propagated-cons*
append-eq-append-conv2)
qed
moreover have *H*: ‹*?case* ⟷ (∀ *Ca*∈#*add-mset ?M C*.
∀ *C'*∈#*remove1-mset Ca C*. *?P Ca C'*)›
unfolding *conflict-clauses-are-entailed2-def prod.case*
apply (*intro conjI iffI impI ballI*)
subgoal for *Ca C'*
by (*auto dest*: *multi-member-split dest*: *in-diffD*)
subgoal for *Ca C'*
using *dist'*
by (*auto 5 3 dest*!: *multi-member-split*[*of Ca*] *dest*: *in-diffD*)
done
moreover have ‹(∀ *Ca*∈#*C*. ∀ *C'*∈#*remove1-mset Ca C*. *?P Ca C'*)›
using *ent2* **unfolding** *conflict-clauses-are-entailed2-def*
by *auto*
ultimately show *?case*
unfolding *H*
by *auto*
next
case (*backtrack-opt S T C*) **note** *bt* = *this*(*1*) **and** *ent* = *this*(*2*) **and** *ent2* = *this*(*3*) **and** *dist* = *this*(*4*)
and *invs* = *this*(*5*)
let *?M* = ‹(*cut-and-complete-trail S*)›
have ‹*conflict-clauses-are-entailed* (*T*, *add-mset ?M C*)› **and**
dist': ‹*distinct-mset* (*add-mset ?M C*)›
using $odpll_W$*-core-stgy-count-conflict-clauses-are-entailed*[*OF* - *ent*, *of* ‹(*T*, *add-mset ?M C*)›]
$odpll_W$*-core-stgy-count-distinct-mset*[*OF* - *ent dist invs*, *of* ‹(*T*, *add-mset ?M C*)›]
bt **by** (*auto dest*!: $odpll_W$*-core-stgy-count.intros*(*4*)[*of S T C*])

obtain *M1 K M2* **where**
spl: ‹*backtrack-split* (*trail S*) = (*M2*, *Decided K* # *M1*)›
using *bt backtrack-split-snd-hd-decided*[*of* ‹*trail S*›]
by (*cases* ‹*hd* (*snd* (*backtrack-split* (*trail S*)))›) (*auto simp*: *bnb.backtrack-opt.simps*)
have *has-dec*: ‹∃ *l*∈*set* (*trail S*). *is-decided l*›
using *bt* **apply** (*auto simp*: *bnb.backtrack-opt.simps*)
by (*metis annotated-lit.disc*(*1*) *backtrack-split-list-eq in-set-conv-decomp snd-conv spl*)

let *?P* = ‹λ*Ca C'*.
(∃ *L*. *Decided L* ∈ *set Ca* ∧ *Propagated* (− *L*) () ∈ *set C'*) ∨
(∃ *L*. *Propagated L* () ∈ *set Ca* ∧ *Decided* (− *L*) ∈ *set C'*)›
have ‹∀ *C'*∈#*remove1-mset ?M C*. *?P ?M C'*›
proof
fix *C'*
assume ‹*C'*∈#*remove1-mset ?M C*›
then have ‹*C'* ∈# *C*› **and** ‹*C'* ≠ *?M*›
using *dist'* **by** *auto*

then obtain *M′ L M M′′* **where**
‹*trail S* = *M′* @ *Propagated L* () # *M*› **and**
‹*C′* = *M′′* @ *Decided* (− *L*) # *M*›
using *ent* **unfolding** *conflict-clauses-are-entailed-def*
by *auto*
then show ‹*?P ?M C′*›
using *backtrack-split-some-is-decided-then-snd-has-hd*[*of* ‹*trail S*›, *OF has-dec*]
spl backtrack-split-list-eq[*of* ‹*trail S*›, *symmetric*]
by (*clarsimp simp*: *conj-disj-distribR ex-disj-distrib decided-cons-eq-append-decide-cons*
cdcl$_W$*-restart-mset.propagated-cons-eq-append-decide-cons propagated-cons-eq-append-propagated-cons*
append-eq-append-conv2)
qed
moreover have *H*: ‹*?case* ⟷ (∀ *Ca*∈#*add-mset ?M C*.
∀ *C′*∈#*remove1-mset Ca C*. *?P Ca C′*)›
unfolding *conflict-clauses-are-entailed2-def prod.case*
apply (*intro conjI iffI impI ballI*)
subgoal for *Ca C′*
by (*auto dest*: *multi-member-split dest*: *in-diffD*)
subgoal for *Ca C′*
using *dist′*
by (*auto 5 3 dest*!: *multi-member-split*[*of Ca*] *dest*: *in-diffD*)
done
moreover have ‹(∀ *Ca*∈#*C*. ∀ *C′*∈#*remove1-mset Ca C*. *?P Ca C′*)›
using *ent2* **unfolding** *conflict-clauses-are-entailed2-def*
by *auto*
ultimately show *?case*
unfolding *H*
by *auto*
qed

lemma *odpll*$_W$*-bnb-stgy-count-conflict-clauses-are-entailed2*:
assumes
‹*odpll*$_W$*-bnb-stgy-count S T*› **and**
‹*conflict-clauses-are-entailed S*› **and**
‹*conflict-clauses-are-entailed2 S*› **and**
‹*distinct-mset* (*snd S*)› **and**
invs: ‹*dpll*$_W$*-all-inv* (*bnb.abs-state* (*fst S*))›
shows
‹*conflict-clauses-are-entailed2 T*›
using *assms odpll*$_W$*-core-stgy-count-conflict-clauses-are-entailed2*[*of S T*]
apply (*auto simp*: *odpll*$_W$*-bnb-stgy-count.simps*)
apply (*auto simp*: *conflict-clauses-are-entailed2-def*
bnb.dpll$_W$*-bound.simps*)
done

definition *no-complement-set-lit* :: ‹′*v dpll*$_W$*-ann-lits* ⇒ *bool*› **where**
‹*no-complement-set-lit M* ⟷
(∀ *L* ∈ ΔΣ. *Decided* (*Pos* (*replacement-pos L*)) ∈ *set M* ⟶ *Decided* (*Pos* (*replacement-neg L*)) ∉ *set M*) ∧
(∀ *L* ∈ ΔΣ. *Decided* (*Neg* (*replacement-pos L*)) ∉ *set M*) ∧
(∀ *L* ∈ ΔΣ. *Decided* (*Neg* (*replacement-neg L*)) ∉ *set M*) ∧
atm-of ‘ *lits-of-l M* ⊆ Σ − ΔΣ ∪ *replacement-pos* ‘ ΔΣ ∪ *replacement-neg* ‘ ΔΣ›

definition *no-complement-set-lit-st* :: ‹′*st* × ′*v dpll*$_W$*-ann-lits multiset* ⇒ *bool*› **where**
‹*no-complement-set-lit-st* = (λ(*S*, *Cs*). (∀ *C*∈#*Cs*. *no-complement-set-lit C*) ∧ *no-complement-set-lit*

(*trail S*))›

lemma *backtrack-no-complement-set-lit*: ‹*no-complement-set-lit* (*trail S*) $\Longrightarrow$
backtrack-split (*trail S*) = (*M'*, *L* # *M*) $\Longrightarrow$
no-complement-set-lit (*Propagated* (− *lit-of L*) () # *M*)›
using *backtrack-split-list-eq*[*of* ‹*trail S*›, *symmetric*]
by (*auto simp*: *no-complement-set-lit-def*)

lemma *odpll*$_W$*-core-stgy-count-no-complement-set-lit-st*:
assumes
‹*odpll*$_W$*-core-stgy-count S T*› **and**
‹*conflict-clauses-are-entailed S*› **and**
‹*conflict-clauses-are-entailed2 S*› **and**
‹*distinct-mset* (*snd S*)› **and**
invs: ‹*dpll*$_W$*-all-inv* (*bnb.abs-state* (*fst S*))› **and**
‹*no-complement-set-lit-st S*› **and**
atms: ‹*clauses* (*fst S*) = *penc N*› ‹*atms-of-mm N* = Σ› **and**
‹*no-smaller-propa* (*fst S*)›
shows
‹*no-complement-set-lit-st T*›
using *assms*
proof (*induction rule*: *odpll*$_W$*-core-stgy-count.induct*)
case (*propagate S T C*)
then show *?case*
using *atms-of-mm-penc-subset2*[*of N*] ΔΣ-Σ
apply (*auto simp*: *dpll-propagate.simps no-complement-set-lit-st-def no-complement-set-lit-def*
dpll$_W$*-all-inv-def dest*!: *multi-member-split*)
apply *blast*
apply *blast*
apply *auto*
done
next
case (*decided S T C*)
have *H1*: *False* **if** ‹*Decided* (*Pos* ($L^{\mapsto 0}$)) ∈ *set* (*trail S*)›
‹*undefined-lit* (*trail S*) (*Pos* ($L^{\mapsto 1}$))› ‹*L* ∈ ΔΣ› **for** *L*
proof −
have ‹{#*Neg* ($L^{\mapsto 0}$), *Neg* ($L^{\mapsto 1}$)#} ∈# *clauses S*›
using *decided that*
by (*fastforce simp*: *penc-def additional-constraints-def additional-constraint-def*)
then show *False*
using *decided*(*2*) *that*
apply (*auto 7 4 simp*: *dpll-propagate.simps add-mset-eq-add-mset all-conj-distrib*
imp-conjR imp-conjL remove1-mset-empty-iff defined-lit-Neg-Pos-iff lits-of-def
dest!: *multi-member-split dest*: *in-lits-of-l-defined-litD*)
apply (*metis* (*full-types*) *image-iff lit-of.simps*(*1*))
apply *auto*
apply (*metis* (*full-types*) *image-iff lit-of.simps*(*1*))
done
qed
have *H2*: *False* **if** ‹*Decided* (*Pos* ($L^{\mapsto 1}$)) ∈ *set* (*trail S*)›
‹*undefined-lit* (*trail S*) (*Pos* ($L^{\mapsto 0}$))› ‹*L* ∈ ΔΣ› **for** *L*
proof −
have ‹{#*Neg* ($L^{\mapsto 0}$), *Neg* ($L^{\mapsto 1}$)#} ∈# *clauses S*›
using *decided that*
by (*fastforce simp*: *penc-def additional-constraints-def additional-constraint-def*)
then show *False*

using *decided*(*2*) *that*
apply (*auto 7 4 simp*: *dpll-propagate.simps add-mset-eq-add-mset all-conj-distrib imp-conjR imp-conjL remove1-mset-empty-iff defined-lit-Neg-Pos-iff lits-of-def dest*!: *multi-member-split dest*: *in-lits-of-l-defined-litD*)
apply (*metis* (*full-types*) *image-iff lit-of.simps*(*1*))
apply *auto*
apply (*metis* (*full-types*) *image-iff lit-of.simps*(*1*))
done
qed
have ‹*?case* ⟷ *no-complement-set-lit* (*trail T*)›
using *decided*(*1*,*7*) **unfolding** *no-complement-set-lit-st-def*
by (*auto simp*: *odecide.simps*)
moreover have ‹*no-complement-set-lit* (*trail T*)›
proof −
have *H*: ‹$L \in \Delta\Sigma \Longrightarrow$
Decided (*Pos* ($L^{\mapsto 1}$)) ∈ *set* (*trail S*) ⟹
Decided (*Pos* ($L^{\mapsto 0}$)) ∈ *set* (*trail S*) ⟹ *False*›
‹$L \in \Delta\Sigma \Longrightarrow$ *Decided* (*Neg* ($L^{\mapsto 1}$)) ∈ *set* (*trail S*) ⟹ *False*›
‹$L \in \Delta\Sigma \Longrightarrow$ *Decided* (*Neg* ($L^{\mapsto 0}$)) ∈ *set* (*trail S*) ⟹ *False*›
‹*atm-of* ‘ *lits-of-l* (*trail S*) ⊆ Σ − ΔΣ ∪ *replacement-pos* ‘ ΔΣ ∪ *replacement-neg* ‘ ΔΣ›
for *L*
using *decided*(*7*) **unfolding** *no-complement-set-lit-st-def no-complement-set-lit-def*
by *blast*+
have ‹$L \in \Delta\Sigma \Longrightarrow$
Decided (*Pos* ($L^{\mapsto 1}$)) ∈ *set* (*trail T*) ⟹
Decided (*Pos* ($L^{\mapsto 0}$)) ∈ *set* (*trail T*) ⟹ *False*› **for** *L*
using *decided*(*1*) *H*(*1*)[*of L*] *H1*[*of L*] *H2*[*of L*]
by (*auto simp*: *odecide.simps no-complement-set-lit-def*)
moreover have ‹$L \in \Delta\Sigma \Longrightarrow$ *Decided* (*Neg* ($L^{\mapsto 1}$)) ∈ *set* (*trail T*) ⟹ *False*› **for** *L*
using *decided*(*1*) *H*(*2*)[*of L*]
by (*auto simp*: *odecide.simps no-complement-set-lit-def*)
moreover have ‹$L \in \Delta\Sigma \Longrightarrow$ *Decided* (*Neg* ($L^{\mapsto 0}$)) ∈ *set* (*trail T*) ⟹ *False*› **for** *L*
using *decided*(*1*) *H*(*3*)[*of L*]
by (*auto simp*: *odecide.simps no-complement-set-lit-def*)
moreover have ‹*atm-of* ‘ *lits-of-l* (*trail T*) ⊆ Σ − ΔΣ ∪ *replacement-pos* ‘ ΔΣ ∪ *replacement-neg* ‘ ΔΣ›
using *decided*(*1*) *H*(*4*)
by (*auto 5 3 simp*: *odecide.simps no-complement-set-lit-def lits-of-def image-image*)

ultimately show *?thesis*
by (*auto simp*: *no-complement-set-lit-def*)
qed
ultimately show *?case*
by *fast*

next
case (*backtrack S T C*) **note** *bt* = *this*(*1*) **and** *ent* = *this*(*2*) **and** *ent2* = *this*(*3*) **and** *dist* = *this*(*4*) **and** *invs* = *this*(*6*)
show *?case*
using *bt invs*
by (*auto simp*: *dpll-backtrack.simps no-complement-set-lit-st-def backtrack-no-complement-set-lit*)

next
case (*backtrack-opt S T C*) **note** *bt* = *this*(*1*) **and** *ent* = *this*(*2*) **and** *ent2* = *this*(*3*) **and** *dist* = *this*(*4*)

and *invs* = *this*(*6*)
show *?case*
using *bt invs*
by (*auto simp*: *bnb.backtrack-opt.simps no-complement-set-lit-st-def backtrack-no-complement-set-lit*)
qed

lemma *odpll*$_W$*-bnb-stgy-count-no-complement-set-lit-st*:
assumes
‹*odpll*$_W$*-bnb-stgy-count S T*› **and**
‹*conflict-clauses-are-entailed S*› **and**
‹*conflict-clauses-are-entailed2 S*› **and**
‹*distinct-mset* (*snd S*)› **and**
invs: ‹*dpll*$_W$*-all-inv* (*bnb.abs-state* (*fst S*))› **and**
‹*no-complement-set-lit-st S*› **and**
atms: ‹*clauses* (*fst S*) = *penc N*› ‹*atms-of-mm N* = Σ› **and**
‹*no-smaller-propa* (*fst S*)›
shows
‹*no-complement-set-lit-st T*›
using *odpll*$_W$*-core-stgy-count-no-complement-set-lit-st*[*of S T*, *OF* - *assms*(*2*−)] *assms*(*1*,*6*)
by (*auto simp*: *odpll*$_W$*-bnb-stgy-count.simps no-complement-set-lit-st-def bnb.dpll*$_W$*-bound.simps*)

definition *stgy-invs* :: ‹′*v clauses* ⇒ ′*st* × - ⇒ *bool*› **where**
‹*stgy-invs N S* ⟷
no-smaller-propa (*fst S*) ∧
conflict-clauses-are-entailed S ∧
conflict-clauses-are-entailed2 S ∧
distinct-mset (*snd S*) ∧
(∀ *C* ∈# *snd S*. *no-dup C*) ∧
dpll$_W$*-all-inv* (*bnb.abs-state* (*fst S*)) ∧
no-complement-set-lit-st S ∧
clauses (*fst S*) = *penc N* ∧
atms-of-mm N = Σ
›

lemma *odpll*$_W$*-bnb-stgy-count-stgy-invs*:
assumes
‹*odpll*$_W$*-bnb-stgy-count S T*› **and**
‹*stgy-invs N S*›
shows ‹*stgy-invs N T*›
using *odpll*$_W$*-bnb-stgy-count-conflict-clauses-are-entailed2*[*of S T*]
odpll$_W$*-bnb-stgy-count-conflict-clauses-are-entailed*[*of S T*]
odpll$_W$*-bnb-stgy-no-smaller-propa*[*of* ‹*fst S*› ‹*fst T*›]
odpll$_W$*-bnb-stgy-countD*[*of S T*]
odpll$_W$*-bnb-stgy-clauses*[*of* ‹*fst S*› ‹*fst T*›]
odpll$_W$*-core-stgy-count-distinct-mset*[*of S T*]
odpll$_W$*-bnb-stgy-count-no-dup-clss*[*of S T*]
odpll$_W$*-bnb-stgy-count-distinct-mset*[*of S T*]
assms
odpll$_W$*-bnb-stgy-dpll*$_W$*-bnb-stgy*[*of* ‹*fst S*› *N* ‹*fst T*›]
odpll$_W$*-bnb-stgy-count-no-complement-set-lit-st*[*of S T*]
using *local.bnb.dpll*$_W$*-bnb-abs-state-all-inv*
unfolding *stgy-invs-def*
by *auto*

```
lemma stgy-invs-size-le:
  assumes ‹stgy-invs N S›
  shows ‹size (snd S) ≤ 3 ^ (card Σ)›
proof -
  have ‹no-smaller-propa (fst S)› and
    ‹conflict-clauses-are-entailed S› and
    ent2: ‹conflict-clauses-are-entailed2 S› and
    dist: ‹distinct-mset (snd S)› and
    n-d: ‹(∀ C ∈# snd S. no-dup C)› and
    ‹dpll_W-all-inv (bnb.abs-state (fst S))› and
    nc: ‹no-complement-set-lit-st S› and
    Σ: ‹atms-of-mm N = Σ›
    using assms unfolding stgy-invs-def by fast+

  let ?f = ‹(filter-mset is-decided o mset)›
  have ‹distinct-mset (?f `# (snd S))›
    apply (subst distinct-image-mset-inj)
    subgoal
      using ent2 n-d
      apply (auto simp: conflict-clauses-are-entailed2-def
        inj-on-def add-mset-eq-add-mset dest!: multi-member-split split-list)
      using n-d apply auto
      apply (metis defined-lit-def multiset-partition set-mset-mset union-iff union-single-eq-member)+
      done
    subgoal
      using dist by auto
    done
  have H: ‹lit-of `# ?f C ∈ all-sound-trails list-new-vars› if ‹C ∈# (snd S)› for C
  proof -
    have nc: ‹no-complement-set-lit C› and n-d: ‹no-dup C›
      using nc that n-d unfolding no-complement-set-lit-st-def
      by (auto dest!: multi-member-split)
    have taut: ‹¬tautology (lit-of `# mset C)›
      using n-d no-dup-not-tautology by blast
    have taut: ‹¬tautology (lit-of `# ?f C)›
      apply (rule not-tautology-mono[OF - taut])
      by (simp add: image-mset-subseteq-mono)
    have dist: ‹distinct-mset (lit-of `# mset C)›
      using n-d no-dup-distinct by blast
    have dist: ‹distinct-mset (lit-of `# ?f C)›
      apply (rule distinct-mset-mono[OF - dist])
      by (simp add: image-mset-subseteq-mono)

    show ?thesis
      apply (rule in-all-sound-trails)
      subgoal
        using nc unfolding no-complement-set-lit-def
        by (auto dest!: multi-member-split simp: is-decided-def)
      subgoal
        using nc unfolding no-complement-set-lit-def
        by (auto dest!: multi-member-split simp: is-decided-def)
      subgoal
        using nc unfolding no-complement-set-lit-def
        by (auto dest!: multi-member-split simp: is-decided-def)
      subgoal
        using nc n-d taut dist unfolding no-complement-set-lit-def set-list-new-vars
```

by (*auto dest*!: *multi-member-split simp*: *set-list-new-vars*
is-decided-def simple-clss-def atms-of-def lits-of-def
image-image dest!: *split-list*)
subgoal
by (*auto simp*: *set-list-new-vars*)
done
qed
then have *incl*: ‹*set-mset* ((*image-mset lit-of o ?f*) '# (*snd S*)) ⊆ *all-sound-trails list-new-vars*›
by *auto*
have *K*: ‹*xs* ≠ [] ⟹ ∃ *y ys*. *xs* = *y* # *ys*› **for** *xs*
by (*cases xs*) *auto*
have *K2*: ‹*Decided La* # *zsb* = *us* @ *Propagated* (*L*) () # *zsa* ⟷
(*us* ≠ [] ∧ *hd us* = *Decided La* ∧ *zsb* = *tl us* @ *Propagated* (*L*) () # *zsa*)› **for** *La zsb us L zsa*
apply (*cases us*)
apply *auto*
done
have *inj*: ‹*inj-on* (('#) *lit-of* ∘ (*filter-mset is-decided* ∘ *mset*))
(*set-mset* (*snd S*))›
unfolding *inj-on-def*
proof (*intro ballI impI*, *rule ccontr*)
fix *x y*
assume *x*: ‹*x* ∈# *snd S*› **and**
y: ‹*y* ∈# *snd S*› **and**
eq: ‹(('#) *lit-of* ∘ (*filter-mset is-decided* ∘ *mset*)) *x* =
(('#) *lit-of* ∘ (*filter-mset is-decided* ∘ *mset*)) *y*› **and**
neq: ‹*x* ≠ *y*›
consider
L **where** ‹*Decided L* ∈ *set x*› ‹*Propagated* (− *L*) () ∈ *set y*› |
L **where** ‹*Decided L* ∈ *set y*› ‹*Propagated* (− *L*) () ∈ *set x*›
using *ent2 n-d x y* **unfolding** *conflict-clauses-are-entailed2-def*
by (*auto dest*!: *multi-member-split simp*: *add-mset-eq-add-mset neq*)
then show *False*
proof *cases*
case *1*
show *False*
using *eq 1*(*1*) *multi-member-split*[*of* ‹*Decided L*› ‹*mset x*›]
apply *auto*
by (*smt 1*(*2*) *lit-of.simps*(*2*) *msed-map-invR multiset-partition n-d*
no-dup-cannot-not-lit-and-uminus set-mset-mset union-mset-add-mset-left union-single-eq-member
y)
next
case *2*
show *False*
using *eq 2 multi-member-split*[*of* ‹*Decided L*› ‹*mset y*›]
apply *auto*
by (*smt lit-of.simps*(*2*) *msed-map-invR multiset-partition n-d*
no-dup-cannot-not-lit-and-uminus set-mset-mset union-mset-add-mset-left union-single-eq-member
x)
qed
qed

have [*simp*]: ‹*finite* Σ›
unfolding Σ[*symmetric*]
by *auto*
have [*simp*]: ‹Σ ∪ ΔΣ = Σ›
using ΔΣ-Σ **by** *blast*

```
  have ‹size (snd S) = size (((image-mset lit-of o ?f) `# (snd S)))›
    by auto
  also have ‹... = card (set-mset ((image-mset lit-of o ?f) `# (snd S)))›
    supply [[goals-limit=1]]
    apply (subst distinct-mset-size-eq-card)
    apply (subst distinct-image-mset-inj[OF inj])
    using dist by auto
  also have ‹... ≤ card (all-sound-trails list-new-vars)›
    by (rule card-mono[OF - incl]) simp
  also have ‹... ≤ card (simple-clss (Σ − ΔΣ)) ∗ 3 ^ card ΔΣ›
    using card-all-sound-trails[of list-new-vars]
    by (auto simp: set-list-new-vars distinct-list-new-vars
      length-list-new-vars)
  also have ‹... ≤ 3 ^ card (Σ − ΔΣ) ∗ 3 ^ card ΔΣ›
    using simple-clss-card[of ‹Σ − ΔΣ›]
    unfolding set-list-new-vars distinct-list-new-vars
      length-list-new-vars
    by (auto simp: set-list-new-vars distinct-list-new-vars
      length-list-new-vars)
  also have ‹... = (3 :: nat) ^ (card Σ)›
    unfolding comm-semiring-1-class.semiring-normalization-rules(26)
    by (subst card-Un-disjoint[symmetric])
      auto
  finally show ‹size (snd S) ≤ 3 ^ card Σ›
    .
qed

lemma rtranclp-odpll_W-bnb-stgy-count-stgy-invs: ‹odpll_W-bnb-stgy-count** S T ⟹ stgy-invs N S ⟹
stgy-invs N T›
  apply (induction rule: rtranclp-induct)
  apply (auto dest!: odpll_W-bnb-stgy-count-stgy-invs)
  done

theorem
  assumes ‹clauses S = penc N› ‹atms-of-mm N = Σ› and
    ‹odpll_W-bnb-stgy-count** (S, {#}) (T, D)› and
    tr: ‹trail S = []›
  shows ‹size D ≤ 3 ^ (card Σ)›
proof −
  have i: ‹stgy-invs N (S, {#})›
    using tr unfolding no-smaller-propa-def
      stgy-invs-def conflict-clauses-are-entailed-def
      conflict-clauses-are-entailed2-def assms(1,2)
      no-complement-set-lit-st-def no-complement-set-lit-def
      dpll_W-all-inv-def
    by (auto simp: assms(1))
  show ?thesis
    using rtranclp-odpll_W-bnb-stgy-count-stgy-invs[OF assms(3) i]
      stgy-invs-size-le[of N ‹(T, D)›]
    by auto
qed

end

end
```