

Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

December 6, 2019

Contents

```

theory Model-Enumeration
  imports Entailment-Definition.Partial-Annotated-Herbrand-Interpretation
    Weidenbach-Book-Base.Wellfounded-More
begin

lemma Ex-sat-model:
  assumes  $\langle \text{satisfiable } (\text{set-mset } N) \rangle$ 
  shows  $\langle \exists M. \text{ set } M \models_{sm} N \wedge$ 
     $\text{distinct } M \wedge$ 
     $\text{consistent-interp } (\text{set } M) \wedge$ 
     $\text{atm-of } ' \text{ set } M \subseteq \text{atms-of-mm } N \rangle$ 
proof –
  from assms obtain I where
    I-N:  $\langle I \models_{sm} N \rangle$  and
    consistent:  $\langle \text{consistent-interp } I \rangle$  and
     $\langle \text{total-over-m } I (\text{set-mset } N) \rangle$  and
    atms-I-N:  $\langle \text{atm-of } ' I = \text{atms-of-mm } N \rangle$ 
  unfolding satisfiable-def-min by blast
  have  $\langle I \subseteq \text{Pos } ' (\text{atms-of-mm } N) \cup \text{Neg } ' (\text{atms-of-mm } N) \rangle$ 
  using atms-I-N
  by (smt in-set-image-subsetD literal.exhaust-sel subsetI sup-ge1 sup-ge2)
  then have  $\langle \text{finite } I \rangle$ 
  using infinite-super by fastforce
  then obtain I' where  $I' : \langle I = \text{set } I' \rangle$  and dist:  $\langle \text{distinct } I' \rangle$ 
  using finite-distinct-list by force
  show ?thesis
  apply (rule exI[of - I'])
  using I-N dist consistent atms-I-N by (auto simp: I')
qed

```

```

definition all-models where
   $\langle \text{all-models } N = \{M. \text{ set } M \models_{sm} N \wedge \text{consistent-interp } (\text{set } M) \wedge$ 
     $\text{distinct } M \wedge \text{atm-of } ' \text{ set } M \subseteq \text{atms-of-mm } N\} \rangle$ 

```

```

lemma finite-all-models:
   $\langle \text{finite } (\text{all-models } N) \rangle$ 
proof –
  let ?n =  $\langle \text{Pos } ' (\text{atms-of-mm } N) \cup \text{Neg } ' (\text{atms-of-mm } N) \rangle$ 
  have H:  $\langle \text{all-models } N \subseteq \{M. \text{ set } M \subseteq ?n \wedge \text{length } M \leq \text{card } ?n\} \rangle$ 
  unfolding all-models-def
  apply (auto dest: imageI[of - - atm-of])
  apply (metis contra-subsetD image-eqI literal.exhaust-sel)
  by (smt atms-of-ms-finite card-mono distinct-card finite-Un finite-imageI
    finite-set-mset image-subset-iff literal.exhaust-sel subsetI sup-ge1 sup-ge2)

```

```

show ?thesis
  apply (rule finite-subset)
  apply (rule H)
  apply (rule finite-lists-length-le)
  apply auto
done
qed

```

inductive *next-model* **where**

```

⟨set M ⊨sm N ⟹ distinct M ⟹ consistent-interp (set M) ⟹
  atm-of 'set M ⊆ atms-of-mm N ⟹ next-model M N⟩

```

lemma *image-mset-uminus-eq-image-mset-uminus-literals[simp]*:

```

⟨image-mset uminus M' = image-mset uminus M ⟷ M = M'⟩ for M :: ⟨'v clause⟩
by (auto simp: inj-image-mset-eq-iff inj-def)

```

context

```

fixes P :: ⟨'v literal set ⟹ bool⟩

```

begin

inductive *next-model-filtered* :: ⟨'v literal list option × 'v literal multiset multiset
 ⇒ 'v literal list option × 'v literal multiset multiset
 ⇒ bool⟩ **where**

```

⟨next-model M N ⟹ P (set M) ⟹ next-model-filtered (None, N) (Some M, N)⟩ |
⟨next-model M N ⟹ ¬P (set M) ⟹ next-model-filtered (None, N) (None, add-mset (image-mset
uminus (mset M)) N)⟩

```

lemma *next-model-filtered-mono*:

```

⟨next-model-filtered a b ⟹ snd a ⊆# snd b⟩
by (induction rule: next-model-filtered.induct) auto

```

lemma *rtrancplp-next-model-filtered-mono*:

```

⟨next-model-filtered** a b ⟹ snd a ⊆# snd b⟩
by (induction rule: rtrancplp-induct) (auto dest: next-model-filtered-mono)

```

lemma *next-filtered-same-atoms*:

```

⟨next-model-filtered a b ⟹ atms-of-mm (snd b) = atms-of-mm (snd a)⟩
by (induction rule: next-model-filtered.induct) (auto simp: next-model.simps atms-of-def)

```

lemma *rtrancplp-next-filtered-same-atoms*:

```

⟨next-model-filtered** a b ⟹ atms-of-mm (snd b) = atms-of-mm (snd a)⟩
by (induction rule: rtrancplp-induct) (auto simp: next-filtered-same-atoms)

```

lemma *next-model-filtered-next-modelD*:

```

⟨next-model-filtered a b ⟹ M ∈# snd b - snd a ⟹ M = image-mset uminus (mset M') ⟹
  next-model M' (snd a)⟩
by (induction arbitrary: M M' rule: next-model-filtered.induct)
  (auto simp: next-model.simps distinct-mset-mset-distinct[symmetric]
    dest: mset-eq-setD
    simp del: distinct-mset-mset-distinct)

```

lemma *rtrancplp-next-model-filtered-next-modelD*:

```

⟨next-model-filtered** a b ⟹ M ∈# snd b - snd a ⟹ M = image-mset uminus (mset M') ⟹
  next-model M' (snd a)⟩

```

proof (induction arbitrary: M M' rule: rtrancplp-induct)

```

case base
then show ?case by auto
next
case (step y z) note star = this(1) and step = this(2) and IH = this(3) and M-in = this(4) and
  M = this(5)
consider
   $\langle M \in \# \text{ snd } y - \text{snd } a \rangle \mid$ 
   $\langle M \in \# \text{ snd } z - \text{snd } y \rangle$ 
using step star M-in
by (smt rtrancpl-next-model-filtered-mono add-diff-cancel-right
    in-multiset-minus-notin-snd rtrancpl.rtrancpl-into-rtrancpl subset-mset.diff-add)
then show ?case
proof cases
  case 1
  show ?thesis
    by (rule IH[OF 1 M])
next
  case 2
  then show ?thesis
    using step rtrancpl-next-model-filtered-mono[OF star] rtrancpl-next-filtered-same-atoms[OF star]
    unfolding subset-mset.le-iff-add
    by (force simp: next-model-filtered.simps M next-model.simps
      distinct-mset-mset-distinct[symmetric]
      dest: mset-eq-setD
      simp del: distinct-mset-mset-distinct)
qed
qed

```

lemma *rtrancpl-next-model-filtered-next-false:*

$\langle \text{next-model-filtered}^{**} a b \implies M \in \# \text{ snd } b - \text{snd } a \implies M = \text{image-mset } \text{uminus } (\text{mset } M') \implies$
 $\neg P (\text{uminus } \text{'set-mset } M) \rangle$

proof (*induction arbitrary: M M' rule: rtrancpl-induct*)

case *base*

then show ?*case* **by** *auto*

next

case (*step y z*) **note** *star = this(1)* **and** *step = this(2)* **and** *IH = this(3)* **and** *M-in = this(4)* **and**
M = this(5)

consider

$\langle M \in \# \text{ snd } y - \text{snd } a \rangle \mid$

$\langle M \in \# \text{ snd } z - \text{snd } y \rangle$

using *step star M-in*

by (*smt rtrancpl-next-model-filtered-mono add-diff-cancel-right*
in-multiset-minus-notin-snd rtrancpl.rtrancpl-into-rtrancpl subset-mset.diff-add)

then show ?*case*

proof cases

case 1

show ?*thesis*

by (*rule IH[OF 1 M]*)

next

case 2

then show ?*thesis*

using *step rtrancpl-next-model-filtered-mono[OF star] rtrancpl-next-filtered-same-atoms[OF star]*

unfolding *subset-mset.le-iff-add*

by (*force simp: next-model-filtered.simps M next-model.simps*
distinct-mset-mset-distinct[symmetric] image-image)

```

    dest: mset-eq-setD
    simp del: distinct-mset-mset-distinct)
qed
qed

lemma next-model-decreasing:
  assumes
    ⟨next-model M N⟩
  shows ⟨(add-mset (image-mset uminus (mset M)) N, N)
    ∈ measure (λN. card (all-models N))⟩
proof -
  have ⟨M ∈ all-models N⟩
    using assms unfolding all-models-def
    by (auto simp: true-clss-def true-cls-mset-def next-model.simps)
  moreover {
    have ⟨¬ set M ⊨ image-mset uminus (mset M)⟩
      using assms unfolding true-cls-def all-models-def
      by (auto simp: true-clss-def consistent-interp-def next-model.simps)
    then have ⟨M ∉ all-models (add-mset (image-mset uminus (mset M)) N)⟩
      unfolding all-models-def by (auto elim!: simp: true-clss-def)
  }
  moreover {
    have ⟨atm-of ' uminus ' set M ∪ atms-of-ms (set-mset N) = atms-of-ms (set-mset N)⟩
      using assms unfolding true-cls-def all-models-def
      by (auto simp: true-clss-def consistent-interp-def atms-of-def next-model.simps)
    then have ⟨all-models (add-mset (image-mset uminus (mset M)) N) ⊆ all-models N⟩
      using assms unfolding all-models-def
      by (auto simp: atms-of-def)
  }
  ultimately have ⟨all-models (add-mset (image-mset uminus (mset M)) N) ⊂ all-models N⟩
    by auto
  then show ?thesis
    by (auto simp: finite-all-models psubset-card-mono)
qed

```

```

lemma next-model-decreasing':
  assumes
    ⟨next-model M N⟩
  shows ⟨((P, add-mset (image-mset uminus (mset M)) N), P, N)
    ∈ measure (λ(P, N). card (all-models N))⟩
  using next-model-decreasing[OF assms] by auto

```

```

lemma wf-next-model-filtered:
  ⟨wf {(y, x). next-model-filtered x y}⟩
proof -
  have ⟨wf {(y, x). True ∧ next-model-filtered x y}⟩
    by (rule wfP-if-measure[of ⟨λ-. True⟩ next-model-filtered
      ⟨λN. (if fst N = None then 1 else 0) + card (all-models (snd N))⟩])
      (auto dest: next-model-decreasing simp: next-model-filtered.simps)
  then show ?thesis
    unfolding wfP-def
    by simp
qed

```

```

lemma no-step-next-model-filtered-unsat:
  assumes ⟨no-step next-model-filtered (None, N)⟩

```

shows $\langle \text{unsatisfiable } (\text{set-mset } N) \rangle$
by $(\text{metis } \text{Ex-sat-model } \text{Model-Enumeration.next-model-filtered.simps}$
 $\text{assms next-model.intros})$

lemma *unsat-no-step-next-model-filtered*:
assumes $\langle \text{unsatisfiable } (\text{set-mset } N) \rangle$
shows $\langle \text{no-step next-model-filtered } (None, N) \rangle$
by $(\text{metis } (\text{no-types, lifting}) \text{ next-model-filtered.simps assms}$
 $\text{next-model.cases satisfiable-carac' snd-conv})$

lemma *full-next-model-filtered-no-distinct-model*:
assumes
 $\text{no-model: } \langle \text{full next-model-filtered } (None, N) (None, N') \rangle$ **and**
 $\text{filter-mono: } \langle \bigwedge M M'. \text{ set } M \models_{sm} N \implies \text{consistent-interp } (\text{set } M) \implies \text{set } M' \models_{sm} N \implies$
 $\text{distinct } M \implies \text{distinct } M' \implies \text{set } M \subseteq \text{set } M' \implies P (\text{set } M) \longleftrightarrow P (\text{set } M') \rangle$
shows
 $\langle \nexists M. \text{ set } M \models_{sm} N \wedge P (\text{set } M) \wedge \text{consistent-interp } (\text{set } M) \wedge \text{distinct } M \rangle$

proof *clarify*

fix M

assume

$M-N: \langle \text{set } M \models_m N \rangle$ **and**

$P-M: \langle P (\text{set } M) \rangle$ **and**

$\text{consistent: } \langle \text{consistent-interp } (\text{set } M) \rangle$ **and**

$\text{dist-M: } \langle \text{distinct } M \rangle$

have $st: \langle \text{next-model-filtered}^{**} (None, N) (None, N') \rangle$ **and**

$ns: \langle \text{no-step next-model-filtered } (None, N') \rangle$

using $\text{no-model unfolding full-def by blast+}$

define Ms **where** $\langle Ms = N' - N \rangle$

then have $N'[simp]: \langle N' = N + Ms \rangle$

using $\text{rtranclp-next-model-filtered-mono}[OF st]$ **by** auto

have $\langle \text{unsatisfiable } (\text{set-mset } N') \rangle$

using ns **by** $(\text{rule no-step-next-model-filtered-unsat})$

then have $\langle \neg \text{set } M \models_m Ms \rangle$

using $\text{consistent } M-N$ **by** $(\text{auto simp: satisfiable-carac[symmetric]})$

then obtain M' **where**

$M'-MS: \langle M' \in \# Ms \rangle$ **and**

$M-M': \langle \neg \text{set } M \models M' \rangle$

by $(\text{auto simp: true-cls-mset-def})$

obtain M'' **where**

$[simp]: \langle M' = \text{mset } M'' \rangle$

using $\text{ex-mset}[of M']$ **by** auto

let $?M'' = \langle \text{map } \text{uminus } M'' \rangle$

have $\langle \text{next-model } ?M'' (\text{snd } (None :: 'v \text{ literal list option}, N)) \rangle$

apply $(\text{rule rtranclp-next-model-filtered-next-modelD}[OF st, of M'])$

using $M'-MS$ **by** auto

then have

$\text{cons': } \langle \text{consistent-interp } (\text{set } ?M'') \rangle$ **and**

$M''-N: \langle \text{set } ?M'' \models_{sm} N \rangle$ **and**

$\text{dist-M'': } \langle \text{distinct } ?M'' \rangle$

unfolding next-model.simps **by** auto

let $?I = \langle \text{remdups } (M @ ?M'') \rangle$

have $\text{cons-I: } \langle \text{consistent-interp } (\text{set } ?I) \rangle$

using $M-M'$ consistent cons' **by** $(\text{auto simp: consistent-interp-def true-cls-def})$

have $\langle P (\text{set } ?I) \rangle$

```

    using filter-mono[of M (I)] cons' M''-N M-N consistent dist-M'' dist-M P-M
  by auto
then have (P (uminus ' (set M''))
  using filter-mono[of (M'' ?I)] cons' M''-N M-N consistent dist-M'' dist-M P-M cons-I
  by auto
then show False
  using rtrncp-next-model-filtered-next-false[OF st, of M' ?M'] M'-MS by auto
qed

lemma full-next-model-filtered-no-model:
  assumes
    no-model: (full next-model-filtered (None, N) (None, N')) and
    filter-mono: (M M'. set M ⊨sm N ⇒ consistent-interp (set M) ⇒ set M' ⊨sm N ⇒
      distinct M ⇒ distinct M' ⇒ set M ⊆ set M' ⇒ P (set M) ⇔ P (set M'))
  shows
    (M. set M ⊨sm N ∧ P (set M) ∧ consistent-interp (set M))
    (is (M. ?P M))
proof -
  have H: (M. ?P M) ⇔ (M. set M ⊨sm N ∧ P (set M) ∧ consistent-interp (set M) ∧ distinct
M)
  by (auto intro: exI[of - (remdups -)])
show ?thesis
  apply (subst H)
  apply (rule full-next-model-filtered-no-distinct-model)
  apply (rule no-model)
  apply (rule filter-mono; assumption)
  done
qed

end

lemma no-step-next-model-filtered-next-model-iff:
  (fst S = None ⇒ no-step (next-model-filtered P) S ⇔ (M. next-model M (snd S)))
  apply (cases S; auto simp: next-model-filtered.simps)
  by metis

lemma Ex-next-model-iff-satisfiable:
  ((M. next-model M N) ⇔ satisfiable (set-mset N))
  by (metis no-step-next-model-filtered-next-model-iff
    next-model.cases no-step-next-model-filtered-unsat prod.sel(1) prod.sel(2) satisfiable-carac)

lemma unsat-no-step-next-model-filtered':
  assumes (unsatisfiable (set-mset (snd S)) ∨ fst S ≠ None)
  shows (no-step (next-model-filtered P) S)
  using assms
  apply cases
  apply (auto dest: unsat-no-step-next-model-filtered)
  apply (metis Ex-next-model-iff-satisfiable fst-conv next-model-filtered.simps
    no-step-next-model-filtered-next-model-iff)
  by (metis Pair-inject next-model-filtered.cases option.simps(3) prod.collapse)

end

theory Watched-Literals-Transition-System-Enumeration
  imports Watched-Literals.Watched-Literals-Transition-System Model-Enumeration
begin

```


Design decision: we favour shorter clauses to (potentially) better models.

More precisely, we take the clause composed of decisions, instead of taking the full trail. This creates shorter clauses. However, this makes satisfying the initial clauses *harder* since fewer literals can be left undefined or be defined with the wrong sign.

For now there is no difference, since TWL produces only full models anyway. Remark that this is the clause that is produced by the minimization of the conflict of the full trail (except that this clauses would be learned and not added to the initial set of clauses, meaning that that the set of initial clauses is not harder to satisfy).

It is not clear if that would really make a huge performance difference.

The name DECO (e.g., *DECO-clause*) comes from Armin Biere's "decision only clauses" (DECO) optimisation (see Armin Biere's "Lingeling, Plingeling and Treengeling Entering the SAT Competition 2013"). If the learned clause becomes much larger than the clause normally learned by backjump, then the clause composed of the negation of the decision is learned instead (effectively doing a backtrack instead of a backjump). Unless we get more information from the filtering function, we are in the special case where the 1st-UIP is exactly the last decision.

An important property of the transition rules is that they violate the invariant that propagations are fully done before each decision. This means that we handle the transitions as a fast restart and not as a backjump as one would expect, since we cannot reuse any theorem about backjump.

definition *DECO-clause* :: $\langle ('v, 'a) \text{ ann-lits} \Rightarrow 'v \text{ clause} \rangle$ **where**
 $\langle \text{DECO-clause } M = (\text{uminus } o \text{ lit-of}) \text{ ' \# (filter-mset is-decided (mset } M)) \rangle$

lemma *distinct-mset-DECO*:

$\langle \text{distinct-mset (DECO-clause } M) \longleftrightarrow \text{distinct-mset (lit-of ' \# filter-mset is-decided (mset } M)) \rangle$
 $\langle \text{is } \langle ?A \longleftrightarrow ?B \rangle \rangle$

proof –

have $\langle ?A \longleftrightarrow \text{distinct-mset (uminus ' \# lit-of ' \# (filter-mset is-decided (mset } M)) \rangle$
by (*auto simp: DECO-clause-def*)
also have $\langle \dots \longleftrightarrow \text{distinct-mset (lit-of ' \# (filter-mset is-decided (mset } M)) \rangle$
apply (*subst distinct-image-mset-inj*)
subgoal by (*auto simp: inj-on-def*)
subgoal by *auto*
done
finally show *?thesis*

qed

lemma [*twl-st*]:

$\langle \text{init-clss (state}_W\text{-of } T) = \text{get-all-init-clss } T \rangle$
 $\langle \text{learned-clss (state}_W\text{-of } T) = \text{get-all-learned-clss } T \rangle$
by (*cases T; auto simp: cdcl_W-restart-mset-state; fail*) $+$

lemma *atms-of-DECO-clauseD*:

$\langle x \in \text{atms-of (DECO-clause } U) \implies x \in \text{atms-of-s (lits-of-l } U) \rangle$
 $\langle x \in \text{atms-of (DECO-clause } U) \implies x \in \text{atms-of (lit-of ' \# mset } U) \rangle$
by (*auto simp: DECO-clause-def atms-of-s-def atms-of-def lits-of-def*)

definition *TWL-DECO-clause* **where**

$\langle \text{TWL-DECO-clause } M =$
 TWL-Clause
 $((\text{uminus } o \text{ lit-of}) \text{ ' \# mset (take 2 (filter is-decided } M)))$
 $((\text{uminus } o \text{ lit-of}) \text{ ' \# mset (drop 2 (filter is-decided } M))) \rangle$

lemma *clause-TWL-Deco-clause*[simp]: $\langle \text{clause } (TWL-DECO\text{-clause } M) = DECO\text{-clause } M \rangle$
by (*auto simp: TWL-DECO-clause-def DECO-clause-def*
simp del: image-mset-union mset-append
simp add: image-mset-union[symmetric] mset-append[symmetric] mset-filter)

inductive *negate-model-and-add-tw* :: $\langle 'v \text{ twl-st} \Rightarrow 'v \text{ twl-st} \Rightarrow \text{bool} \rangle$ **where**
bj-unit:
 $\langle \text{negate-model-and-add-tw } (M, N, U, \text{None}, NP, UP, WS, Q)$
 $(\text{Propagated } (-K) (DECO\text{-clause } M) \# M1, N, U, \text{None}, \text{add-mset } (DECO\text{-clause } M) NP, UP,$
 $\{\#\}, \{\#K\# \}) \rangle$
if
 $\langle (\text{Decided } K \# M1, M2) \in \text{set } (\text{get-all-ann-decomposition } M) \rangle$ **and**
 $\langle \text{get-level } M K = \text{count-decided } M \rangle$ **and**
 $\langle \text{count-decided } M = 1 \rangle$ |
bj-nonunit:
 $\langle \text{negate-model-and-add-tw } (M, N, U, \text{None}, NP, UP, WS, Q)$
 $(\text{Propagated } (-K) (DECO\text{-clause } M) \# M1, \text{add-mset } (TWL-DECO\text{-clause } M) N, U, \text{None}, NP,$
 $UP, \{\#\},$
 $\{\#K\# \}) \rangle$
if
 $\langle (\text{Decided } K \# M1, M2) \in \text{set } (\text{get-all-ann-decomposition } M) \rangle$ **and**
 $\langle \text{get-level } M K = \text{count-decided } M \rangle$ **and**
 $\langle \text{count-decided } M \geq 2 \rangle$ |
restart-nonunit:
 $\langle \text{negate-model-and-add-tw } (M, N, U, \text{None}, NP, UP, WS, Q)$
 $(M1, \text{add-mset } (TWL-DECO\text{-clause } M) N, U, \text{None}, NP, UP, \{\#\}, \{\#\# \}) \rangle$
if
 $\langle (\text{Decided } K \# M1, M2) \in \text{set } (\text{get-all-ann-decomposition } M) \rangle$ **and**
 $\langle \text{get-level } M K < \text{count-decided } M \rangle$ **and**
 $\langle \text{count-decided } M > 1 \rangle$

Some remarks:

- Because of the invariants (unit clauses have to be propagated), a rule *restart_unit* would be the same as the *bj_unit*.
- The rules cleans the components about updates and do not assume that they are empty.

lemma *after-fast-restart-replay*:

assumes

inv: $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv } (M', N, U, \text{None}) \rangle$ **and**
stgy-invs: $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant } (M', N, U, \text{None}) \rangle$ **and**
smaller-propa: $\langle \text{cdcl}_W\text{-restart-mset.no-smaller-propa } (M', N, U, \text{None}) \rangle$ **and**
kept: $\langle \forall L E. \text{Propagated } L E \in \text{set } (\text{drop } (\text{length } M' - n) M') \longrightarrow E \in \# N + U \rangle$ **and**
 $U' \cdot U$: $\langle U' \subseteq \# U \rangle$ **and**
no-conf: $\langle \forall C \in \# N'. \forall M1 K M2. M' = M2 @ \text{Decided } K \# M1 \longrightarrow \neg M1 \models_{\text{as}} C \text{Not } C \rangle$ **and**
no-propa: $\langle \forall C \in \# N'. \forall M1 K M2 L. M' = M2 @ \text{Decided } K \# M1 \longrightarrow L \in \# C \longrightarrow$
 $\neg M1 \models_{\text{as}} C \text{Not } (\text{remove1-mset } L C) \rangle$

shows

$\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy}^{**} ([], N+N', U', \text{None}) (\text{drop } (\text{length } M' - n) M', N+N', U', \text{None}) \rangle$

proof –

let $?S = \langle \lambda n. (\text{drop } (\text{length } M' - n) M', N+N', U', \text{None}) \rangle$

note *cdcl_W-restart-mset-state*[simp]

have

M-lev: $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-M-level-inv } (M', N, U, \text{None}) \rangle$ **and**

```

  alien:  $\langle \text{cdcl}_W\text{-restart-mset.no-strange-atm } (M', N, U, \text{None}) \rangle$  and
  confl:  $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-conflicting } (M', N, U, \text{None}) \rangle$  and
  learned:  $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clause } (M', N, U, \text{None}) \rangle$ 
using inv unfolding  $\text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv-def}$  by fast+

have smaller-conf:  $\langle \text{cdcl}_W\text{-restart-mset.no-smaller-conf } (M', N, U, \text{None}) \rangle$ 
  using stgy-invs unfolding  $\text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant-def}$  by blast
have n-d:  $\langle \text{no-dup } M' \rangle$ 
  using M-lev unfolding  $\text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-M-level-inv-def}$  by simp
let  $?L = \langle \lambda m. M' ! (\text{length } M' - \text{Suc } m) \rangle$ 
have undef-nth-Suc:
   $\langle \text{undefined-lit } (\text{drop } (\text{length } M' - m) M') (\text{lit-of } (?L m)) \rangle$ 
  if  $\langle m < \text{length } M' \rangle$ 
  for m
proof –
  define k where
     $\langle k = \text{length } M' - \text{Suc } m \rangle$ 
  then have Sk:  $\langle \text{length } M' - m = \text{Suc } k \rangle$ 
    using that by linarith
  have k-le-M':  $\langle k < \text{length } M' \rangle$ 
    using that unfolding k-def by linarith
  have n-d':  $\langle \text{no-dup } (\text{take } k M' @ ?L m \# \text{drop } (\text{Suc } k) M') \rangle$ 
    using n-d
    apply (subst (asm) append-take-drop-id[symmetric, of -  $\langle \text{Suc } k \rangle$ ])
    apply (subst (asm) take-Suc-conv-app-nth)
    apply (rule k-le-M')
    apply (subst k-def[symmetric])
    by simp

  show ?thesis
    using n-d'
    apply (subst (asm) no-dup-append-cons)
    apply (subst (asm) k-def[symmetric]) +
    apply (subst k-def[symmetric]) +
    apply (subst Sk) +
    by blast
qed

have atm-in:
   $\langle \text{atm-of } (\text{lit-of } (M' ! m)) \in \text{atms-of-mm } N \rangle$ 
  if  $\langle m < \text{length } M' \rangle$ 
  for m
  using alien that
  by (auto simp:  $\text{cdcl}_W\text{-restart-mset.no-strange-atm-def}$  lits-of-def)
then have atm-in':
   $\langle \text{atm-of } (\text{lit-of } (M' ! m)) \in \text{atms-of-mm } (N + N') \rangle$ 
  if  $\langle m < \text{length } M' \rangle$ 
  for m
  using alien that
  by (auto simp:  $\text{cdcl}_W\text{-restart-mset.no-strange-atm-def}$  lits-of-def)

show ?thesis
  using kept
proof (induction n)
  case 0
  then show ?case by simp

```

```

next
case (Suc m) note IH = this(1) and kept = this(2)
consider
  (le)  $\langle m < \text{length } M' \rangle$  |
  (ge)  $\langle m \geq \text{length } M' \rangle$ 
  by linarith
then show ?case
proof (cases)
  case ge
  then show ?thesis
  using Suc by auto
next
case le
define k where
   $\langle k = \text{length } M' - \text{Suc } m \rangle$ 
then have Sk:  $\langle \text{length } M' - m = \text{Suc } k \rangle$ 
  using le by linarith
have k-le-M':  $\langle k < \text{length } M' \rangle$ 
  using le unfolding k-def by linarith
have kept':  $\langle \forall L E. \text{Propagated } L E \in \text{set } (\text{drop } (\text{length } M' - m) M') \longrightarrow E \in \# N + U \rangle$ 
  using kept k-le-M' unfolding k-def[symmetric] Sk
  by (subst (asm) Cons-nth-drop-Suc[symmetric]) auto
have M':  $\langle M' = \text{take } (\text{length } M' - \text{Suc } m) M' @ ?L m \# \text{trail } (?S m) \rangle$ 
  apply (subst append-take-drop-id[symmetric, of -  $\langle \text{Suc } k \rangle$ ])
  apply (subst take-Suc-conv-app-nth)
  apply (rule k-le-M')
  apply (subst k-def[symmetric])
  unfolding k-def[symmetric] Sk
  by auto

have  $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy } (?S m) (?S (\text{Suc } m)) \rangle$ 
proof (cases  $\langle ?L m \rangle$ )
  case (Decided K) note K = this
  have dec:  $\langle \text{cdcl}_W\text{-restart-mset.decide } (?S m) (?S (\text{Suc } m)) \rangle$ 
    apply (rule  $\text{cdcl}_W\text{-restart-mset.decide-rule[of -  $\langle \text{lit-of } (?L m) \rangle$ ]$ )
    subgoal by simp
    subgoal using undef-nth-Suc[of m] le by simp
    subgoal using le by (auto simp: atm-in)
    subgoal using le k-le-M' K unfolding k-def[symmetric] Sk
      by (auto simp: state-eq-def state-def Cons-nth-drop-Suc[symmetric])
    done
  have Dec:  $\langle M' ! k = \text{Decided } K \rangle$ 
    using K unfolding k-def[symmetric] Sk .

have H:  $\langle D + \{\#L\# \} \in \# N + U \longrightarrow \text{undefined-lit } (\text{trail } (?S m)) L \longrightarrow$ 
   $\neg (\text{trail } (?S m)) \models_{\text{as}} \text{CNot } D \rangle$  for D L
  using smaller-propa unfolding  $\text{cdcl}_W\text{-restart-mset.no-smaller-propa-def}$ 
  trail.simps clauses-def
   $\text{cdcl}_W\text{-restart-mset-state}$ 
  apply (subst (asm) M')
  unfolding Dec Sk k-def[symmetric]
  by (auto simp: clauses-def state-eq-def)
have no-new-propa:  $\langle \text{False} \rangle$ 
  if
     $\langle \text{drop } (\text{Suc } k) M' \models_{\text{as}} \text{CNot } (\text{remove1-mset } L E) \rangle$  and
     $\langle L \in \# E \rangle$  and

```

```

    ⟨undefined-lit (drop (Suc k) M') L⟩ and
    ⟨E ∈# N'⟩ for L E
  using that no-propa Sk[symmetric]
  apply (subst (asm)(3) M')
  apply (subst (asm)(2) M')
  apply (subst (asm) M')
  unfolding k-def[symmetric] Dec
  apply (auto simp: k-def dest!: multi-member-split[of - N'])
  by (metis Sk that(1))

have ⟨D ∈# N ⟶ undefined-lit (trail (?S m)) L ⟶ L ∈# D ⟶
  ¬ (trail (?S m)) ⊨as CNot (remove1-mset L D)⟩ and
⟨D ∈# U' ⟶ undefined-lit (trail (?S m)) L ⟶ L ∈# D ⟶
  ¬ (trail (?S m)) ⊨as CNot (remove1-mset L D)⟩ for D L
  using H[of ⟨remove1-mset L D⟩ L] U'-U by auto
then have nss: ⟨no-step cdclW-restart-mset.propagate (?S m)⟩
  using no-propa no-new-propa
  by (auto simp: cdclW-restart-mset.propagate.simps clauses-def
    state-eq-def k-def[symmetric] Sk)
have no-new-confl: ⟨drop (Suc k) M' ⊨as CNot D ⟹ D ∈# N' ⟹ False⟩ for D
  using no-confl
  apply (subst (asm)(2) M')
  apply (subst (asm) M')
  unfolding k-def[symmetric] Dec
  by (auto simp: k-def dest!: multi-member-split)
  (metis K M' Sk cdclW-restart-mset-state(1) drop-append
    k-def length-take true-annots-append-l)

have H: ⟨D ∈# N + U' ⟶ ¬ (trail (?S m)) ⊨as CNot D⟩ for D
  using smaller-confl U'-U unfolding cdclW-restart-mset.no-smaller-confl-def
    trail.simps clauses-def cdclW-restart-mset-state
  apply (subst (asm) M')
  unfolding Dec Sk k-def[symmetric]
  by (auto simp: clauses-def state-eq-def)
then have nsc: ⟨no-step cdclW-restart-mset.conflict (?S m)⟩
  using no-new-confl
  by (auto simp: cdclW-restart-mset.conflict.simps clauses-def state-eq-def
    k-def[symmetric] Sk)
show ?thesis
  apply (rule cdclW-restart-mset.cdclW-stgy.other')
  apply (rule nsc)
  apply (rule nss)
  apply (rule cdclW-restart-mset.cdclW-o.decide)
  apply (rule dec)
  done
next
case K: (Propagated K C)
have Propa: ⟨M' ! k = Propagated K C⟩
  using K unfolding k-def[symmetric] Sk .
have
  M-C: ⟨trail (?S m) ⊨as CNot (remove1-mset K C)⟩ and
  K-C: ⟨K ∈# C⟩
  using confl unfolding cdclW-restart-mset.cdclW-conflicting-def trail.simps
  by (subst (asm)(3) M'; auto simp: k-def[symmetric] Sk Propa)+
have [simp]: ⟨k - min (length M') k = 0⟩
  unfolding k-def by auto

```

```

have C-N-U:  $\langle C \in \# N + U \rangle$ 
  using learned kept unfolding cdclW-restart-mset.cdclW-learned-clause-alt-def Sk
    k-def[symmetric]
  apply (subst (asm)(4)M')
  apply (subst (asm)(10)M')
  unfolding K
  by (auto simp: K k-def[symmetric] Sk Propa clauses-def)
have  $\langle \text{cdcl}_W\text{-restart-mset.propagate } (?S\ m) (?S\ (Suc\ m)) \rangle$ 
  apply (rule cdclW-restart-mset.propagate-rule[of - C K])
  subgoal by simp
  subgoal using C-N-U by (auto simp add: clauses-def)
  subgoal using K-C .
  subgoal using M-C .
  subgoal using undef-nth-Suc[of m] le K by (simp add: k-def[symmetric] Sk)
  subgoal
    using le k-le-M' K unfolding k-def[symmetric] Sk
    by (auto simp: state-eq-def
      state-def Cons-nth-drop-Suc[symmetric])
  done
then show ?thesis
  by (rule cdclW-restart-mset.cdclW-stgy.propagate')
qed
then show ?thesis
  using IH[OF kept] by simp
qed
qed
qed

```

lemma *after-fast-restart-replay'*:

assumes

inv: $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv } (M', N, U, \text{None}) \rangle$ **and**
stgy-invs: $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-invariant } (M', N, U, \text{None}) \rangle$ **and**
smaller-propa: $\langle \text{cdcl}_W\text{-restart-mset.no-smaller-propa } (M', N, U, \text{None}) \rangle$ **and**
kept: $\langle \forall L\ E. \text{Propagated } L\ E \in \text{set } (\text{drop } (\text{length } M' - n)\ M') \longrightarrow E \in \# N + U \rangle$ **and**
U'-U: $\langle U' \subseteq \# U \rangle$ **and**
N-N': $\langle N \subseteq \# N' \rangle$ **and**
no-conf: $\langle \forall C \in \# N' - N. \forall M1\ K\ M2. M' = M2 @ \text{Decided } K \# M1 \longrightarrow \neg M1 \models_{as} C \text{Not } C \rangle$ **and**
no-propa: $\langle \forall C \in \# N' - N. \forall M1\ K\ M2\ L. M' = M2 @ \text{Decided } K \# M1 \longrightarrow L \in \# C \longrightarrow \neg M1 \models_{as} C \text{Not } (\text{remove1-mset } L\ C) \rangle$

shows

$\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy}^{**} ([], N', U', \text{None}) (\text{drop } (\text{length } M' - n)\ M', N', U', \text{None}) \rangle$
using *after-fast-restart-replay*[OF *inv stgy-invs smaller-propa kept U'-U*, of $\langle N' - N \rangle$]
no-conf *no-propa* *N-N'*
by *auto*

lemma *after-fast-restart-replay-no-stgy*:

assumes

inv: $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv } (M', N, U, \text{None}) \rangle$ **and**
kept: $\langle \forall L\ E. \text{Propagated } L\ E \in \text{set } (\text{drop } (\text{length } M' - n)\ M') \longrightarrow E \in \# N + N' + U \rangle$ **and**
U'-U: $\langle U' \subseteq \# U \rangle$

shows

$\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W^{**} ([], N + N', U', \text{None}) (\text{drop } (\text{length } M' - n)\ M', N + N', U', \text{None}) \rangle$

proof –

let $?S = \langle \lambda n. (\text{drop } (\text{length } M' - n)\ M', N + N', U', \text{None}) \rangle$

note *cdcl_W-restart-mset-state*[*simp*]

have

$M\text{-lev}$: $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-M-level-inv } (M', N, U, \text{None}) \rangle$ **and**
 alien : $\langle \text{cdcl}_W\text{-restart-mset.no-strange-atm } (M', N, U, \text{None}) \rangle$ **and**
 confl : $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-conflicting } (M', N, U, \text{None}) \rangle$ **and**
 learned : $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clause } (M', N, U, \text{None}) \rangle$
using inv **unfolding** $\text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv-def}$ **by** fast+

have $n\text{-d}$: $\langle \text{no-dup } M' \rangle$
using $M\text{-lev}$ **unfolding** $\text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-M-level-inv-def}$ **by** simp
let $?L = \langle \lambda m. M' ! (\text{length } M' - \text{Suc } m) \rangle$
have undef-nth-Suc :
 $\langle \text{undefined-lit } (\text{drop } (\text{length } M' - m) M') (\text{lit-of } (?L m)) \rangle$
if $\langle m < \text{length } M' \rangle$
for m
proof –
define k **where**
 $\langle k = \text{length } M' - \text{Suc } m \rangle$
then have Sk : $\langle \text{length } M' - m = \text{Suc } k \rangle$
using that **by** linarith
have $k\text{-le-}M'$: $\langle k < \text{length } M' \rangle$
using that **unfolding** $k\text{-def}$ **by** linarith
have $n\text{-d}'$: $\langle \text{no-dup } (\text{take } k M' @ ?L m \# \text{drop } (\text{Suc } k) M') \rangle$
using $n\text{-d}$
apply $(\text{subst } (\text{asm}) \text{append-take-drop-id}[\text{symmetric}, \text{of } - \langle \text{Suc } k \rangle])$
apply $(\text{subst } (\text{asm}) \text{take-Suc-conv-app-nth})$
apply $(\text{rule } k\text{-le-}M')$
apply $(\text{subst } k\text{-def}[\text{symmetric}])$
by simp

show $?thesis$
using $n\text{-d}'$
apply $(\text{subst } (\text{asm}) \text{no-dup-append-cons})$
apply $(\text{subst } (\text{asm}) k\text{-def}[\text{symmetric}]) +$
apply $(\text{subst } k\text{-def}[\text{symmetric}]) +$
apply $(\text{subst } Sk) +$
by blast

qed

have atm-in :
 $\langle \text{atm-of } (\text{lit-of } (M' ! m)) \in \text{atms-of-mm } (N + N') \rangle$
if $\langle m < \text{length } M' \rangle$
for m
using alien that
by $(\text{auto simp: cdcl}_W\text{-restart-mset.no-strange-atm-def lits-of-def})$

show $?thesis$
using kept
proof $(\text{induction } n)$
case 0
then show $?case$ **by** simp

next
case $(\text{Suc } m)$ **note** $IH = \text{this}(1)$ **and** $\text{kept} = \text{this}(2)$
consider
 $(le) \langle m < \text{length } M' \rangle \mid$
 $(ge) \langle m \geq \text{length } M' \rangle$
by linarith
then show $?case$

```

proof cases
  case ge
  then show ?thesis
    using Suc by auto
next
  case le
  define k where
     $\langle k = \text{length } M' - \text{Suc } m \rangle$ 
  then have Sk:  $\langle \text{length } M' - m = \text{Suc } k \rangle$ 
    using le by linarith
  have k-le-M':  $\langle k < \text{length } M' \rangle$ 
    using le unfolding k-def by linarith
  have kept':  $\langle \forall L E. \text{Propagated } L E \in \text{set } (\text{drop } (\text{length } M' - m) M') \longrightarrow E \in \# N + N' + U \rangle$ 
    using kept k-le-M' unfolding k-def[symmetric] Sk
    by (subst (asm) Cons-nth-drop-Suc[symmetric]) auto
  have M':  $\langle M' = \text{take } (\text{length } M' - \text{Suc } m) M' @ ?L m \# \text{trail } (?S m) \rangle$ 
    apply (subst append-take-drop-id[symmetric, of -  $\langle \text{Suc } k \rangle$ ])
    apply (subst take-Suc-conv-app-nth)
    apply (rule k-le-M')
    apply (subst k-def[symmetric])
    unfolding k-def[symmetric] Sk
    by auto

  have  $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W (?S m) (?S (\text{Suc } m)) \rangle$ 
proof (cases  $\langle ?L m \rangle$ )
  case (Decided K) note K = this
  have dec:  $\langle \text{cdcl}_W\text{-restart-mset.decide } (?S m) (?S (\text{Suc } m)) \rangle$ 
    apply (rule cdclW-restart-mset.decide-rule[of -  $\langle \text{lit-of } (?L m) \rangle$ ])
    subgoal by simp
    subgoal using undef-nth-Suc[of m] le by simp
    subgoal using le atm-in by auto
    subgoal using le k-le-M' K unfolding k-def[symmetric] Sk
      by (auto simp: state-eq-def state-def Cons-nth-drop-Suc[symmetric])
    done
  have Dec:  $\langle M' ! k = \text{Decided } K \rangle$ 
    using K unfolding k-def[symmetric] Sk .

  show ?thesis
    apply (rule cdclW-restart-mset.cdclW.intros(3))
    apply (rule cdclW-restart-mset.cdclW-o.decide)
    apply (rule dec)
    done
next
  case K: (Propagated K C)
  have Propa:  $\langle M' ! k = \text{Propagated } K C \rangle$ 
    using K unfolding k-def[symmetric] Sk .
  have
    M-C:  $\langle \text{trail } (?S m) \models_{\text{as}} \text{CNot } (\text{remove1-mset } K C) \rangle$  and
    K-C:  $\langle K \in \# C \rangle$ 
    using confl unfolding cdclW-restart-mset.cdclW-conflicting-def trail.simps
    by (subst (asm)(3) M'; auto simp: k-def[symmetric] Sk Propa)+
  have [simp]:  $\langle k - \min (\text{length } M') k = 0 \rangle$ 
    unfolding k-def by auto
  have C-N-U:  $\langle C \in \# N + N' + U \rangle$ 
    using learned kept unfolding cdclW-restart-mset.cdclW-learned-clause-alt-def Sk
    k-def[symmetric]

```



```

apply (subst (asm)(4)M')
apply (subst (asm)(10)M')
unfolding K
by (auto simp: K k-def[symmetric] Sk Propa clauses-def)
have ⟨cdclW-restart-mset.propagate (?S m) (?S (Suc m))⟩
apply (rule cdclW-restart-mset.propagate-rule[of - C K])
subgoal by simp
subgoal using C-N-U by (simp add: clauses-def)
subgoal using K-C .
subgoal using M-C .
subgoal using undef-nth-Suc[of m] le K by (simp add: k-def[symmetric] Sk)
subgoal
  using le k-le-M' K unfolding k-def[symmetric] Sk
  by (auto simp: state-eq-def
    state-def Cons-nth-drop-Suc[symmetric])
done
then show ?thesis
  by (rule cdclW-restart-mset.cdclW.intros)
qed
then show ?thesis
  using IH[OF kept] by simp
qed
qed
qed

```

lemma after-fast-restart-replay-no-stgy':

```

assumes
  inv: ⟨cdclW-restart-mset.cdclW-all-struct-inv (M', N, U, None)⟩ and
  kept: ⟨∀ L E. Propagated L E ∈ set (drop (length M' - n) M') ⟶ E ∈# N' + U'⟩ and
  U'-U: ⟨U' ⊆# U⟩ and
  N ⊆# N'
shows
  ⟨cdclW-restart-mset.cdclW** ([], N', U', None) (drop (length M' - n) M', N', U', None)⟩
using after-fast-restart-replay-no-stgy[OF inv, of n ⟨N'-N⟩ U'] assms by auto

```

lemma cdcl_W-all-struct-inv-move-to-init:

```

assumes inv: ⟨cdclW-restart-mset.cdclW-all-struct-inv (M, N, U + U', D)⟩
shows ⟨cdclW-restart-mset.cdclW-all-struct-inv (M, N + U', U, D)⟩
using assms
unfolding cdclW-restart-mset.cdclW-all-struct-inv-def
  cdclW-restart-mset.cdclW-M-level-inv-def cdclW-restart-mset.distinct-cdclW-state-def
  cdclW-restart-mset.cdclW-learned-clause-alt-def cdclW-restart-mset.cdclW-conflicting-def
  cdclW-restart-mset.no-strange-atm-def cdclW-restart-mset-state clauses-def
  assms
apply (intro conjI impI)
subgoal by auto
subgoal by auto
subgoal by auto
subgoal by auto
subgoal by auto
subgoal by auto
subgoal by auto
subgoal by auto
subgoal by auto
subgoal by auto

```

```

subgoal by auto
subgoal by auto
subgoal by (auto simp: ac-simps)
subgoal by (auto simp: ac-simps)
subgoal by auto
done

lemma twl-struct-invs-move-to-init:
  assumes  $\langle twl\text{-}struct\text{-}invs\ (M, N, U + U', D, NP, UP, WS, Q) \rangle$ 
  shows  $\langle twl\text{-}struct\text{-}invs\ (M, N + U', U, D, NP, UP, WS, Q) \rangle$ 
proof -
  have  $H: \langle N + (U + U') = N + U' + U \rangle$ 
    by simp
  have struct-invs:
     $\langle cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}all\text{-}struct\text{-}inv\ (M, clauses\ N + NP, clauses\ (U + U') + UP, D') \implies$ 
     $cdcl_W\text{-}restart\text{-}mset.cdcl_W\text{-}all\text{-}struct\text{-}inv\ (M, clauses\ (N + U') + NP, clauses\ U + UP, D') \rangle$ 
    for  $D'$ 
  using  $cdcl_W\text{-}all\text{-}struct\text{-}inv\text{-}move\text{-}to\text{-}init[of\ M\ \langle clauses\ N + NP \rangle\ \langle clauses\ U + UP \rangle$ 
     $\langle clauses\ U' \rangle\ D']$ 
  by (auto simp: ac-simps)
  have smaller:  $\langle clauses\ N + NP + (clauses\ (U + U') + UP) = clauses\ (N + U') + NP + (clauses\ U + UP) \rangle$ 
    by auto
  show ?thesis
    using assms
    apply (cases  $D$ ; clarify)
    unfolding twl-struct-invs-def twl-st-inv.simps valid-enqueued.simps
      twl-st-exception-inv.simps no-duplicate-queued.simps
      confl-cands-enqueued.simps distinct-queued.simps propa-cands-enqueued.simps
      assms entailed-clss-inv.simps past-invs.simps  $H$  stateW-of.simps
      cdclW-restart-mset.no-smaller-propa-def cdclW-restart-mset-state clauses-def
      twl-exception-inv.simps get-conflict.simps literals-to-update.simps clauses-to-update.simps
      clauses-to-update-inv.simps
    apply (intro conjI)
  subgoal by fast
  subgoal by fast
  subgoal by fast
  subgoal by fast
  subgoal by fast
  subgoal by (rule struct-invs) fast
  subgoal unfolding smaller by argo
  subgoal by argo
  subgoal by argo
  subgoal by argo
  subgoal by fast
  subgoal by fast
  subgoal by argo
  subgoal by fast
  subgoal by argo
  subgoal by blast
  subgoal by fast
  subgoal by argo
  subgoal by argo
  subgoal by argo
  subgoal by argo
  apply (intro conjI)

```

```

subgoal by fast
subgoal by fast
subgoal by fast
subgoal by fast
subgoal by fast
subgoal by (rule struct-invs) fast
subgoal unfolding smaller by argo
subgoal by argo
subgoal by argo
subgoal by argo
subgoal by fast
subgoal by fast
subgoal by argo
subgoal by fast
subgoal by argo
subgoal by argo
subgoal by fast
subgoal by argo
subgoal by argo
done
qed

lemma negate-model-and-add-twl-twl-struct-invs:
  fixes  $S\ T :: \langle 'v\ twl-st \rangle$ 
  assumes
     $\langle negate-model-and-add-twl\ S\ T \rangle$  and
     $\langle twl-struct-invs\ S \rangle$ 
  shows  $\langle twl-struct-invs\ T \rangle$ 
  using assms
proof (induction rule: negate-model-and-add-twl.induct)
  fix  $K :: \langle 'v\ literal \rangle$  and  $M1\ M2\ M\ N\ U\ NP\ UP\ WS\ Q$ 
  assume
     $decomp: \langle (Decided\ K\ \# \ M1, M2) \in set\ (get-all-ann-decomposition\ M) \rangle$  and
     $inv: \langle twl-struct-invs\ (M, N, U, None, NP, UP, WS, Q) \rangle$ 

  let  $?S = \langle (M, N, U, None, NP, UP, WS, Q) \rangle$ 
  let  $?T = \langle (Propagated\ K\ (DECO-clause\ M)\ \# \ M1, add-mset\ (TWL-DECO-clause\ M)\ N, U, None,$ 
     $NP, UP, \{\#\}, \{\# - K\#\}) \rangle$ 
  have
     $st-invs: \langle twl-st-inv\ ?S \rangle$  and
     $\langle valid-enqueued\ ?S \rangle$  and
     $struct-invs: \langle cdcl_W-restart-mset.cdcl_W-all-struct-inv\ (state_W-of\ ?S) \rangle$  and
     $no-smaller: \langle cdcl_W-restart-mset.no-smaller-propa\ (state_W-of\ ?S) \rangle$  and
     $\langle twl-st-exception-inv\ ?S \rangle$  and
     $\langle no-duplicate-queued\ ?S \rangle$  and
     $\langle distinct-queued\ ?S \rangle$  and
     $\langle confl-cands-enqueued\ ?S \rangle$  and
     $\langle propa-cands-enqueued\ ?S \rangle$  and
     $\langle get-conflict\ ?S \neq None \longrightarrow clauses-to-update\ ?S = \{\#\} \wedge literals-to-update\ ?S = \{\#\} \rangle$  and
     $entailed: \langle entailed-clss-inv\ ?S \rangle$  and
     $\langle clauses-to-update-inv\ ?S \rangle$  and
     $past: \langle past-invs\ ?S \rangle$ 
  using  $inv$  unfolding  $twl-struct-invs-def$ 
  by fast+
  obtain  $M3$  where
     $M: \langle M = M3\ @\ M2\ @\ Decided\ K\ \# \ M1 \rangle$ 

```

```

    using decomp by blast
  define M2' where
     $\langle M2' = M3 @ M2 \rangle$ 
  then have M':  $\langle M = M2' @ \text{Decided } K \# M1 \rangle$ 
    using M by auto
  then have
    st-invs-M1':  $\langle \forall C \in \#N + U. \text{twl-lazy-update } M1 \ C \wedge$ 
      watched-literals-false-of-max-level M1 C  $\wedge$ 
      twl-exception-inv (M1, N, U, None, NP, UP,  $\{\#\}$ ,  $\{\#\}$ ) C  $\rangle$  and
    confl-enqueued-M1:  $\langle \text{confl-cands-enqueued } (M1, N, U, \text{None}, NP, UP, \{\#\}, \{\#\}) \rangle$  and
    propa-enqueued-M1:  $\langle \text{propa-cands-enqueued } (M1, N, U, \text{None}, NP, UP, \{\#\}, \{\#\}) \rangle$  and
    clss-upd:  $\langle \text{clauses-to-update-inv } (M1, N, U, \text{None}, NP, UP, \{\#\}, \{\#\}) \rangle$  and
    past-M1:  $\langle \text{past-invs } (M1, N, U, \text{None}, NP, UP, \{\#\}, \{\#\}) \rangle$ 
    using past
    unfolding past-invs.simps
    by auto
  have no-dup:  $\langle \text{no-dup } M \rangle$ 
    using struct-invs unfolding cdclW-restart-mset.cdclW-all-struct-inv-def
      cdclW-restart-mset.cdclW-M-level-inv-def
    by (simp add: trail.simps)
  hence undef-K:  $\langle \text{undefined-lit } M1 \ K \rangle$  and n-d1:  $\langle \text{no-dup } M1 \rangle$ 
    unfolding M' by (auto dest: no-dup-appendD)
  have dist:  $\langle \text{distinct } (\text{map atm-of } (\text{map lit-of } M)) \rangle$ 
    using no-dup by (auto simp: no-dup-def comp-def)

  have dist-filtered:  $\langle \text{distinct-mset } (\text{lit-of } \# \text{ mset } (\text{filter is-decided } M)) \rangle$ 
    apply (rule distinct-mset-mono[of -  $\langle \text{lit-of } \# \text{ mset } M \rangle$ ])
    subgoal by (auto intro!: image-mset-subseteq-mono simp: mset-filter)
    subgoal using dist by (auto simp: mset-map[symmetric] simp del: mset-map
      intro: distinct-mapI)
    done
  then have dist-filtered':  $\langle \text{distinct-mset } (\text{uminus } \# \text{ lit-of } \# \text{ mset } (\text{filter is-decided } M)) \rangle$ 
    apply (subst distinct-image-mset-inj)
    subgoal by (auto simp: inj-on-def)
    subgoal .
    done
  have cdcl-W:  $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W^{**} ([], \text{clauses } (\text{add-mset } (\text{TWL-DECO-clause } M) \ N) + NP,$ 
    clauses U + UP, None)
    (drop (length M - length M1) M, clauses (add-mset (TWL-DECO-clause M) N) + NP, clauses
    U + UP,
    None)  $\rangle$ 
    apply (rule after-fast-restart-replay-no-stgy'[OF struct-invs[unfolded stateW-of.simps]])
  subgoal
    apply (intro allI impI conjI)
    subgoal for L E
      by (use M' struct-invs cdclW-restart-mset.in-get-all-mark-of-propagated-in-trail[of E M]
        in  $\langle \text{auto } \text{simp add: } \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clause-alt-def}$ 
          cdclW-restart-mset.cdclW-all-struct-inv-def cdclW-restart-mset-state clauses-def  $\rangle$ )
      done
    subgoal by simp
    subgoal by simp
    done

  have  $\langle \text{distinct-mset } (\text{DECO-clause } M) \rangle$ 
    using dist-filtered' unfolding DECO-clause-def
    by (simp add: mset-filter)

```

```

then have struct-invs-S':
  ⟨cdclW-restart-mset.cdclW-all-struct-inv ([], clauses (add-mset (TWL-DECO-clause M) N) + NP,
    clauses U + UP, None)⟩
using struct-invs
by (auto simp: cdclW-restart-mset.cdclW-all-struct-inv-def
  cdclW-restart-mset.cdclW-M-level-inv-def cdclW-restart-mset.distinct-cdclW-state-def
  cdclW-restart-mset.cdclW-learned-clause-alt-def cdclW-restart-mset.cdclW-conflicting-def
  cdclW-restart-mset.no-strange-atm-def cdclW-restart-mset-state)
with cdclW have struct-invs-add: ⟨cdclW-restart-mset.cdclW-all-struct-inv
  (M1, clauses (add-mset (TWL-DECO-clause M) N) + NP, clauses U + UP, None)⟩
by (auto intro: cdclW-restart-mset.rtranclp-cdclW-all-struct-inv-inv simp: M'
  dest!: cdclW-restart-mset.rtranclp-cdclW-cdclW-restart)
have no-smaller-M1:
  ⟨cdclW-restart-mset.no-smaller-propa (stateW-of (M1, N, U, None, NP, UP, WS, Q))⟩
using no-smaller by (auto simp: cdclW-restart-mset.no-smaller-propa-def
  cdclW-restart-mset-state clauses-def M')
have no-smaller-add:
  ⟨cdclW-restart-mset.no-smaller-propa
    (M1, clauses (add-mset (TWL-DECO-clause M) N) + NP, clauses U + UP, None)⟩
unfolding stateW-of.simps cdclW-restart-mset.no-smaller-propa-def
  cdclW-restart-mset-state clauses-def
proof (intro conjI impI allI)
  fix M1a M2 K' D L
  assume
    M1a: ⟨M1 = M2 @ Decided K' # M1a⟩ and
    DL: ⟨D + {#L#} ∈ # clauses (add-mset (TWL-DECO-clause M) N) + NP + (clauses U +
UP)⟩ and
    undef: ⟨undefined-lit M1a L⟩
consider
  ⟨D + {#L#} ∈ # clauses N + NP + (clauses U + UP)⟩ |
  ⟨D + {#L#} = clause (TWL-DECO-clause M)⟩
using DL by auto
then show ⟨¬ M1a as CNot D⟩
proof cases
  case 1
  then show ?thesis
    using DL M1a undef no-smaller-M1
    by (auto 5 5 simp: cdclW-restart-mset.no-smaller-propa-def
      cdclW-restart-mset-state clauses-def
      add-mset-eq-add-mset)
  next
  case 2
  moreover have ⟨K' ∉ lits-of-l M1a⟩ ⟨¬K ∉ lits-of-l M1a⟩ ⟨K ∉ lits-of-l M1a⟩
    using no-dup unfolding M' M1a
    by (auto simp: add-mset-eq-add-mset
      dest: in-lits-of-l-defined-litD
      elim!: list-match-lel-lel)
  ultimately show ?thesis
    using undef by (auto simp: add-mset-eq-add-mset DECO-clause-def M' M1a
      dest!: multi-member-split)
qed
qed
have wf-N-U: ⟨C ∈ # N + U ⇒ struct-wf-twl-cls C⟩ for C
  using st-invs unfolding twl-st-inv.simps by auto
{
  assume

```

$lev: \langle \text{get-level } M \ K = \text{count-decided } M \rangle$ **and**
 $\text{count-dec}: \langle \text{count-decided } M \geq 2 \rangle$
have $[simp]: \langle \text{filter is-decided } M2' = [] \rangle$
using $\text{count-dec } lev \text{ no-dup unfolding } M'$
by $(\text{auto simp: TWL-DECO-clause-def count-decided-def add-mset-eq-add-mset } M')$
obtain $L' \ C$ **where**
 $\text{filter-M}: \langle \text{filter is-decided } M = \text{Decided } K \ \# \ \text{Decided } L' \ \# \ C \rangle$
using $\text{count-dec } lev \text{ unfolding } M'$
by $(\text{cases } \langle \text{filter is-decided } M \rangle; \text{cases } \langle \text{tl } (\text{filter is-decided } M) \rangle;$
 $\text{cases } \langle \text{hd } (\text{filter is-decided } M) \rangle; \text{cases } \langle \text{hd } (\text{tl } (\text{filter is-decided } M)) \rangle)$
 $(\text{auto simp: TWL-DECO-clause-def count-decided-def add-mset-eq-add-mset } M'$
 $\text{filter-eq-Cons-iff tl-append})$
then have $\text{deco-M}: \langle \text{TWL-DECO-clause } M = \text{TWL-Clause } \{\#-K, -L'\# \} \ (\text{uminus } \# \text{ lit-of } \#$
 $\text{mset } C) \rangle$
by $(\text{auto simp: TWL-DECO-clause-def})$
have $C-M1: \langle C = \text{tl } (\text{filter is-decided } M1) \rangle$
using $\text{filter-M unfolding } M'$
by auto
then obtain $M1'' \ M1'$ **where**
 $M1: \langle M1 = M1'' \ @ \ \text{Decided } L' \ \# \ M1' \rangle$
by $(\text{metis } (\text{no-types, lifting}) \ M' \ \langle \text{filter is-decided } M2' = [] \rangle \ \text{append-self-conv2}$
 $\text{filter.simps}(2) \ \text{filter-M filter-append filter-eq-Cons-iff list.sel}(3))$
then have $[simp]: \langle \text{count-decided } M1'' = 0 \rangle$ **and** $\text{filter-M1}'': \langle \text{filter is-decided } M1'' = [] \rangle$
using $\text{filter-M no-dup unfolding } C-M1 \ M1 \ M'$
by $(\text{auto simp: tl-append count-decided-def dest: filter-eq-ConsD split: list.splits})$
have $C\text{-in-}M1: \langle \text{lits-of-l } C \subseteq \text{lits-of-l } M1 \rangle$
unfolding $C-M1$ **by** $(\text{auto simp: lits-of-def dest: in-set-tlD})$

let $?S' = \langle (M1, \text{add-mset } (\text{TWL-DECO-clause } M) \ N, \ U, \ \text{None}, \ NP, \ UP,$
 $\text{add-mset } (-L', (\text{TWL-DECO-clause } M)) \ \{\#\}, \ \{\#\}) \rangle$
let $?T' = \langle (\text{Propagated } (-K) \ (\text{DECO-clause } M) \ \# \ M1, \text{add-mset } (\text{TWL-DECO-clause } M) \ N, \ U,$
 $\text{None},$
 $NP, \ UP, \ \{\#\}, \ \{\#-(-K)\# \}) \rangle$
have $\text{propa}: \langle \text{cdcl-tw-clp } ?S' \ ?T' \rangle$
unfolding $\text{clause-TWL-Deco-clause[symmetric]}$
apply $(\text{rule cdcl-tw-clp.propagate})$
subgoal by $(\text{auto simp: deco-M})$
subgoal using $\text{no-dup unfolding } M \text{ by auto}$
subgoal using $C\text{-in-}M1 \text{ unfolding deco-M by } (\text{auto simp: lits-of-def})$
done

have $\text{struct-invs-}S': \langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv } (\text{state}_W\text{-of } ?S') \rangle$
using $\text{struct-invs-add by auto}$

have $\text{no-smaller-}S': \langle \text{cdcl}_W\text{-restart-mset.no-smaller-propa } (\text{state}_W\text{-of } ?S') \rangle$
using $\text{no-smaller-add by simp}$
have $[simp]: \langle \text{get-level } M1 \ L' = \text{count-decided } M1 \rangle$
using $\text{no-dup unfolding } M' \ M1 \text{ by auto}$
have $\langle \text{watched-literals-false-of-max-level } M1 \ (\text{TWL-DECO-clause } M) \rangle$
using $\text{no-dup apply } (\text{subst } (\text{asm}) \ M') \text{ by}$
 $(\text{auto simp: deco-M add-mset-eq-add-mset dest: in-lits-of-l-defined-litD})$
moreover have $\langle \text{struct-wf-tw-clc } (\text{TWL-DECO-clause } M) \rangle$
using $\text{dist-filtered' unfolding deco-M filter-M}$
by $(\text{auto simp: simp del: clause-TWL-Deco-clause})$
ultimately have $\langle \text{tw-st-inv } ?S' \rangle$
using $\text{wf-N-U st-invs-}M1' \text{ unfolding tw-st-inv.simps}$

```

by (auto simp: twl-is-an-exception-def)

moreover have ⟨valid-enqueued ?S'⟩
  by (auto simp: deco-M) (auto simp: M1)
moreover have ⟨cdclW-restart-mset.cdclW-all-struct-inv (stateW-of ?S')⟩
  using struct-invs-S' .
moreover have ⟨cdclW-restart-mset.no-smaller-propa (stateW-of ?S')⟩
  using no-smaller-S' .
moreover have ⟨twl-st-exception-inv ?S'⟩
  using st-invs-M1' C-in-M1
  by (auto simp: twl-exception-inv.simps deco-M add-mset-eq-add-mset)
  (auto simp: lits-of-def)
moreover have ⟨no-duplicate-queued ?S'⟩
  by (auto simp: M1)
moreover have ⟨distinct-queued ?S'⟩
  by auto
moreover have ⟨confl-cands-enqueued ?S'⟩
  using confl-enqueued-M1 by auto
moreover have ⟨propa-cands-enqueued ?S'⟩
  using propa-enqueued-M1 by auto
moreover {
  have ⟨get-level M L = 0 ⟹ get-level M1 L = 0⟩ for L
    using no-dup defined-lit-no-dupD(1)[of M1 L M2]
    by (cases ⟨defined-lit M L⟩)
      (auto simp: M' defined-lit-append defined-lit-cons atm-of-eq-atm-of
        get-level-cons-if split: if-splits)
  moreover have ⟨get-level M L = 0 ⟹ L ∈ lits-of-l M ⟹ L ∈ lits-of-l M1⟩ for L
    using no-dup defined-lit-no-dupD(1)[of M1 L M2]
    by (cases ⟨defined-lit M L⟩)
      (auto simp: M' defined-lit-append defined-lit-cons atm-of-eq-atm-of
        get-level-cons-if split: if-splits dest: in-lits-of-l-defined-litD)
  ultimately have ⟨entailed-clss-inv ?S'⟩
    using entailed unfolding entailed-clss-inv.simps by meson
}
moreover have ⟨clauses-to-update-inv ?S'⟩
  using clss-upd no-dup unfolding deco-M by (auto simp: deco-M add-mset-eq-add-mset M'
    dest: in-lits-of-l-defined-litD)
moreover have ⟨past-invs ?S'⟩
  unfolding past-invs.simps
proof (intro conjI impI allI)
  fix M1a M2 K'
  assume M1a: ⟨M1 = M2 @ Decided K' # M1a⟩
  let ?SM1a = ⟨(M1a, add-mset (TWL-DECO-clause M) N, U, None, NP, UP, {#}, {#})⟩
  have
    struct:
    ⟨C ∈ #N + U ⟹ twl-lazy-update M1a C ∧
      watched-literals-false-of-max-level M1a C ∧
      twl-exception-inv (M1a, N, U, None, NP, UP, {#}, {#}) C⟩
  for C
  using past-M1 unfolding past-invs.simps unfolding M1a
  by fast+
have
  confl: ⟨confl-cands-enqueued (M1a, N, U, None, NP, UP, {#}, {#})⟩ and
  propa: ⟨propa-cands-enqueued (M1a, N, U, None, NP, UP, {#}, {#})⟩ and
  clss-to-upd: ⟨clauses-to-update-inv (M1a, N, U, None, NP, UP, {#}, {#})⟩
  using past-M1 unfolding past-invs.simps unfolding M1a

```

```

  by fast+
have [iff]:  $\langle L' \notin \text{ lits-of-l } M1a \rangle \langle K \notin \text{ lits-of-l } M1a \rangle$ 
  using no-dup M1 filter-M1'' unfolding deco-M unfolding M' M1a
  by (auto simp: deco-M add-mset-eq-add-mset
      dest: in-lits-of-l-defined-litD
      simp del:  $\langle \text{ filter is-decided } M2' = [] \rangle$ 
      elim!: list-match-lcl-lcl)
have  $\langle \text{ twl-lazy-update } M1a \text{ (TWL-DECO-clause } M) \rangle$ 
  using no-dup M1 unfolding deco-M unfolding M' M1a
  by (auto simp: deco-M add-mset-eq-add-mset
      dest: in-lits-of-l-defined-litD)
moreover have  $\langle \text{ watched-literals-false-of-max-level } M1a \text{ (TWL-DECO-clause } M) \rangle$ 
  unfolding deco-M by (auto simp: add-mset-eq-add-mset)
moreover have  $\langle \text{ twl-exception-inv } ?SM1a \text{ (TWL-DECO-clause } M) \rangle$ 
  unfolding deco-M by (auto simp: add-mset-eq-add-mset twl-exception-inv.simps)
ultimately have  $\langle C \in \# \text{ add-mset (TWL-DECO-clause } M) \ N + U \implies \text{ twl-lazy-update } M1a \ C \wedge$ 
   $\text{ watched-literals-false-of-max-level } M1a \ C \wedge$ 
   $\text{ twl-exception-inv } ?SM1a \ C \rangle$  for C
  using struct[of C]
  by (auto simp: twl-exception-inv.simps)
then show  $\forall C \in \# \text{ add-mset (TWL-DECO-clause } M) \ N + U. \text{ twl-lazy-update } M1a \ C \wedge$ 
   $\text{ watched-literals-false-of-max-level } M1a \ C \wedge$ 
   $\text{ twl-exception-inv } ?SM1a \ C$ 
  by blast
show  $\langle \text{ confl-cands-enqueued } ?SM1a \rangle$ 
  using confl by (auto simp: deco-M)
show  $\langle \text{ propa-cands-enqueued } ?SM1a \rangle$ 
  using propa by (auto simp: deco-M)
show  $\langle \text{ clauses-to-update-inv } ?SM1a \rangle$ 
  using clss-to-upd
  by (auto simp: deco-M clauses-to-update-prop.simps add-mset-eq-add-mset)
qed
moreover have  $\langle \text{ get-conflict } ?S' = \text{ None} \rangle$ 
  by simp
ultimately have  $\langle \text{ twl-struct-invs } ?S' \rangle$ 
  unfolding twl-struct-invs-def
  by meson
then have  $\langle \text{ twl-struct-invs } ?T' \rangle$ 
  by (rule cdcl-tw-cl-tw-struct-invs[OF propa])
then show  $\langle \text{ twl-struct-invs (Propagated } (-K) \text{ (DECO-clause } M) \# M1, \text{ add-mset (TWL-DECO-clause } M) \ N,$ 
   $U, \text{ None, NP, UP, } \{\#\}, \{\#K\#\} \rangle$ 
  by simp
}

{
  let ?S =  $\langle \text{ (Propagated } (-K) \text{ (DECO-clause } M) \# M1, N, U, \text{ None, add-mset (DECO-clause } M) \text{ NP, UP, } \{\#\}, \{\#K\#\} \rangle$ 
  assume  $\langle \text{ count-decided } M = 1 \rangle$ 
  then have [simp]:  $\langle \text{ DECO-clause } M = \{\#-K\#\} \rangle$ 
    using decomp by (auto simp: DECO-clause-def filter-mset-empty-conv count-decided-0-iff
        dest!: get-all-ann-decomposition-exists-prepend)
  have [simp]:  $\langle \text{ get-level } M1 \ L = 0 \rangle \langle \text{ count-decided } M1 = 0 \rangle$  for L
    using count-decided-ge-get-level[of M1 L]  $\langle \text{ count-decided } M = 1 \rangle$ 
    unfolding M by auto

```



```

have K-M:  $\langle K \in \text{ lits-of-}l \ M \rangle$ 
  using M' by simp

have propa:  $\langle \text{cdcl}_W\text{-restart-mset.propagate } (M1, \text{ clauses } (\text{add-mset } (\text{TWL-DECO-clause } M) \ N) +$ 
NP, clauses U + UP, None)
   $(\text{state}_W\text{-of } ?S) \rangle$ 
  unfolding  $\text{state}_W\text{-of.simps}$ 
  apply (rule  $\text{cdcl}_W\text{-restart-mset.propagate-rule[of - } \langle \text{DECO-clause } M \rangle \langle -K \rangle$ )
  subgoal by (simp add:  $\text{cdcl}_W\text{-restart-mset-state}$ )
  subgoal by (simp add: clauses-def)
  subgoal by simp
  subgoal by (simp add:  $\text{cdcl}_W\text{-restart-mset-state}$ )
  subgoal using no-dup by (simp add:  $\text{cdcl}_W\text{-restart-mset-state } M'$ )
  subgoal by (simp add:  $\text{cdcl}_W\text{-restart-mset-state}$ )
  done
have lazy:  $\langle \text{twl-lazy-update } M1 \ C \rangle$  if  $\langle C \in \#N + U \rangle$  for C
  using that st-invs-M1' by blast
have excep:  $\langle \text{twl-exception-inv } (M1, N, U, \text{None}, NP, UP, \{\#\}, \{\#\}) \ C \rangle$  if  $\langle C \in \#N + U \rangle$  for C
  using that st-invs-M1' by blast

have  $\langle \neg \text{twl-is-an-exception } C \ \{\#K\} \ \{\#\} \implies \text{twl-lazy-update } (\text{Propagated } (-K) \ \{\#-K\} \ \#$ 
M1) C  $\rangle$  if  $\langle C \in \#N + U \rangle$  for C
  using lazy[OF that] no-dup undef-K n-d1 excep[OF that]
  by (cases C)
  (auto simp: get-level-cons-if all-conj-distrib twl-exception-inv.simps
    twl-is-an-exception-def
    dest!: no-has-blit-propagate multi-member-split)
moreover have  $\langle \text{watched-literals-false-of-max-level } (\text{Propagated } (-K) \ \{\#-K\} \ \# \ M1) \ C \rangle$  for C
  by (cases C) (auto simp: get-level-cons-if)
ultimately have  $\langle \text{twl-st-inv } ?S \rangle$ 
  using st-invs-M1' wf-N-U by (auto simp: twl-st-inv.simps
    simp del: set-mset-union)
moreover have  $\langle \text{valid-enqueued } ?S \rangle$ 
  by auto
moreover have  $\text{struct-invs-S: } \langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv } (\text{state}_W\text{-of } ?S) \rangle$ 
  using struct-invs-add propa
  by (auto dest!:  $\text{cdcl}_W\text{-restart-mset.propagate } \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-cdcl}_W\text{-restart}$ 
    simp: intro:  $\text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv-inv}$ )
moreover have  $\langle \text{cdcl}_W\text{-restart-mset.no-smaller-propa } (\text{state}_W\text{-of } ?S) \rangle$ 
  using no-smaller-add propa struct-invs-add
  by (auto 5 5 simp: dest!:  $\text{cdcl}_W\text{-restart-mset.propagate } \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy.propagate'}$ 
    intro:  $\text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-no-smaller-propa}$ )
moreover have  $\langle \text{twl-st-exception-inv } ?S \rangle$ 
  using st-invs-M1' no-dup undef-K n-d1
  by (auto simp add: twl-exception-inv.simps
    dest!: no-has-blit-propagate')
moreover have  $\langle \text{no-duplicate-queued } ?S \rangle$ 
  by auto
moreover have  $\langle \text{distinct-queued } ?S \rangle$ 
  by auto
moreover have  $\langle \text{confl-cands-enqueued } ?S \rangle$ 
  unfolding confl-cands-enqueued.simps Ball-def
proof (intro impI allI)
  fix C
  assume
    C:  $\langle C \in \#N + U \rangle$  and

```

$H: \langle \text{Propagated } (- K) (\text{DECO-clause } M) \# M1 \models_{as} \text{CNot } (\text{clause } C) \rangle$
obtain $L1\ L2\ UW$ **where**
 $C': \langle C = \text{TWL-Clause } \{\#L1, L2\# \} UW \rangle$ **and** $\text{dist-}C: \langle \text{distinct-mset } (\text{clause } C) \rangle$
using $\text{wf-}N\text{-}U[OF\ C]$
apply $(\text{cases } C)$
by $(\text{auto simp: twl-exception-inv.simps size-2-iff cdcl}_W\text{-restart-mset-state})$
have $M1\text{-}C: \langle \neg M1 \models_{as} \text{CNot } (\text{clause } C) \rangle$
using $\text{confl-queued-}M1\ C$ **by** auto
define C' **where** $\langle C' = \text{remove1-mset } K (\text{clause } C) \rangle$
then have $C\text{-}K\text{-}C': \langle \text{clause } C = \text{add-mset } K\ C' \rangle$ **and** $\langle K \notin \# C' \rangle$ **and**
 $M1\text{-}C': \langle M1 \models_{as} \text{CNot } C' \rangle$ **and** $K\text{-}C'\text{-}C: \langle \text{add-mset } K\ C' = \text{clause } C \rangle$
using $\text{dist-}C\ M1\text{-}C\ H$ **by** $(\text{auto simp: true-annots-true-cl-def-iff-negation-in-model}$
 $\text{dest: in-diffD dest!: multi-member-split})$
have $\langle C' + \{\#K\# \} \in \# \text{ clauses } (N+U) \rangle$
using $C\ M1\text{-}C'$
by $(\text{auto simp: } K\text{-}C'\text{-}C\ M')$
then have $\langle \text{undefined-lit } M1\ K \implies \neg M1 \models_{as} \text{CNot } C' \rangle$
using no-smaller
unfolding $\text{cdcl}_W\text{-restart-mset.no-smaller-propa-def state}_W\text{-of.simps cdcl}_W\text{-restart-mset-state}$
 $\text{clauses-def image-mset-union } M'\ \text{union-iff}$
by blast
then have False
using $\text{no-dup } M1\text{-}C'$ **unfolding** M'
by $(\text{auto simp: cdcl}_W\text{-restart-mset-state clauses-def } M')$
then show $\langle (\exists L'. L' \in \# \text{ watched } C \wedge L' \in \# \{\#K\# \}) \vee (\exists L. (L, C) \in \# \{\# \}) \rangle$
by fast
qed
moreover have $\langle \text{propa-cands-queued } ?S \rangle$
unfolding $\text{propa-cands-queued.simps Ball-def}$
proof (intro impI allI)
fix $C\ L$
assume
 $C: \langle C \in \# N + U \rangle$ **and**
 $L: \langle L \in \# \text{ clause } C \rangle$ **and**
 $H: \langle \text{Propagated } (- K) (\text{DECO-clause } M) \# M1 \models_{as} \text{CNot } (\text{remove1-mset } L (\text{clause } C)) \rangle$ **and**
 $\text{undef: } \langle \text{undefined-lit } (\text{Propagated } (- K) (\text{DECO-clause } M) \# M1) L \rangle$
obtain $L1\ L2\ UW$ **where**
 $C': \langle C = \text{TWL-Clause } \{\#L1, L2\# \} UW \rangle$ **and** $\text{dist-}C: \langle \text{distinct-mset } (\text{clause } C) \rangle$
using $\text{wf-}N\text{-}U[OF\ C]$
apply $(\text{cases } C)$
by $(\text{auto simp: twl-exception-inv.simps size-2-iff cdcl}_W\text{-restart-mset-state})$
have $M1\text{-}C: \langle \neg M1 \models_{as} \text{CNot } (\text{remove1-mset } L (\text{clause } C)) \rangle$
using $\text{propa-queued-}M1\ C\ \text{undef } L$ **by** auto
define C' **where** $\langle C' = \text{remove1-mset } K (\text{remove1-mset } L (\text{clause } C)) \rangle$
then have $C\text{-}K\text{-}C': \langle \text{clause } C = \text{add-mset } K (\text{add-mset } L\ C') \rangle$ **and** $\langle K \notin \# C' \rangle$ **and**
 $M1\text{-}C': \langle M1 \models_{as} \text{CNot } C' \rangle$ **and** $K\text{-}C'\text{-}C: \langle \text{add-mset } K (\text{add-mset } L\ C') = \text{clause } C \rangle$ **and**
 $K\text{-}C'\text{-}C': \langle \text{add-mset } K\ C' = \text{remove1-mset } L (\text{clause } C) \rangle$
using $\text{dist-}C\ M1\text{-}C\ H\ L$ **by** $(\text{auto simp: true-annots-true-cl-def-iff-negation-in-model}$
 $\text{dest: in-diffD dest!: multi-member-split})$
have $\text{eq2: } \langle \{\#L1, L2\# \} = \{\#L, L'\# \} \iff L = L1 \wedge L' = L2 \vee L = L2 \wedge L' = L1 \rangle$ **for** $L\ L'$
by $(\text{auto simp: add-mset-eq-add-mset})$
have $\langle \text{twl-exception-inv } (M1, N, U, \text{None}, NP, UP, \{\# \}, \{\# \})\ C \rangle$
using $\text{past } C$ **unfolding** $\text{past-invs.simps } M'$
by fast
moreover have $\langle L2 \notin \text{lits-of-}l\ M1 \rangle$

```

using  $H$  no-dup undef dist-C
unfolding true-annots-true-cls-def-iff-negation-in-model  $M'$   $C'$  Ball-def
by (cases  $\langle L = L1 \rangle$ ; cases  $\langle L = L2 \rangle$ ;
    auto dest: in-lits-of-l-defined-litD no-dup-appendD no-dup-consistentD
    simp: all-conj-distrib) +
moreover have  $\langle L1 \notin \text{lits-of-l } M1 \rangle$ 
    using  $H$  no-dup undef dist-C
    unfolding true-annots-true-cls-def-iff-negation-in-model  $M'$   $C'$  Ball-def
    apply (cases  $\langle L = L1 \rangle$ ; cases  $\langle L = L2 \rangle$ )
    by (auto dest: in-lits-of-l-defined-litD no-dup-appendD no-dup-consistentD
        simp: all-conj-distrib)
moreover {
    have  $\langle L' \in \text{lits-of-l } M1 \implies L' \in \# UW \implies \text{False} \rangle$  for  $L'$ 
        using  $H$  no-dup undef dist-C  $\langle L1 \notin \text{lits-of-l } M1 \rangle$   $\langle L2 \notin \text{lits-of-l } M1 \rangle$  n-d1
        unfolding true-annots-true-cls-def-iff-negation-in-model  $M'$   $C'$  Ball-def
        apply (cases  $\langle L = L1 \rangle$ ; cases  $\langle L = L2 \rangle$ )
        apply (auto dest: in-lits-of-l-defined-litD no-dup-appendD no-dup-consistentD
            simp: all-conj-distrib)
        by (metis diff-single-trivial in-lits-of-l-defined-litD insert-DiffM
            insert-noteq-member n-d1 no-dup-consistentD) +
    then have  $\langle \neg \text{has-blit } M1 \text{ (clause (TWL-Clause } \{\#L1, L2\} UW)) L1 \rangle$  and
         $\langle \neg \text{has-blit } M1 \text{ (clause (TWL-Clause } \{\#L1, L2\} UW)) L2 \rangle$ 
        using  $\langle L1 \notin \text{lits-of-l } M1 \rangle$   $\langle L2 \notin \text{lits-of-l } M1 \rangle$ 
        unfolding has-blit-def
        by auto
}
ultimately have
     $\langle \neg L1 \in \text{lits-of-l } M1 \implies (\forall K \in \# UW. \neg K \in \text{lits-of-l } M1) \rangle$ 
     $\langle \neg L2 \in \text{lits-of-l } M1 \implies (\forall K \in \# UW. \neg K \in \text{lits-of-l } M1) \rangle$ 
    unfolding  $C'$  twl-exception-inv.simps twl-clause.sel eq2
    by fastforce +
moreover have  $\langle L1 \neq L2 \rangle$ 
    using dist-C by (auto simp: C')
ultimately have  $\langle K \neq L1 \implies K \neq L2 \implies \text{False} \rangle$ 
    using  $M1-C' L$  undef K-C'-C no-dup[unfolded M]
    by (cases  $\langle \neg L1 \in \text{lits-of-l } M1 \rangle$ ; cases  $\langle \neg L2 \in \text{lits-of-l } M1 \rangle$ ;
        auto simp add: C' true-annots-true-cls-def-iff-negation-in-model
        add-mset-eq-add-mset
        dest!: multi-member-split[of - UW] dest: in-lits-of-l-defined-litD)
then show  $\langle (\exists L'. L' \in \# \text{watched } C \wedge L' \in \# \{\#K\}) \vee (\exists L. (L, C) \in \# \{\#\}) \rangle$ 
    by (auto simp: C')
qed
moreover have  $\langle \text{get-conflict } ?S = \text{None} \rangle$ 
    by simp
moreover {
    have  $\langle \text{get-level } M L = 0 \implies L \in \text{lits-of-l } M \implies L \in \text{lits-of-l } M1 \rangle$  for  $L$ 
        using no-dup defined-lit-no-dupD(1)[of M1 L M2]
        by (cases  $\langle \text{defined-lit } M L \rangle$ )
        (auto simp: M' defined-lit-append defined-lit-cons atm-of-eq-atm-of
            get-level-cons-if split: if-splits dest: in-lits-of-l-defined-litD)
    then have  $\langle \text{entailed-clss-inv } ?S \rangle$ 
        using entailed unfolding entailed-clss-inv.simps by (auto 5 5 simp: get-level-cons-if)
}
moreover {
    have  $\langle \neg \text{clauses-to-update-prop } \{\#\} (M1) (L, La) \implies$ 
         $\text{clauses-to-update-prop } \{\#K\} (\text{Propagated } (\neg K) \{\# - K\} \# M1) (L, La) \implies \text{False} \rangle$  for  $L$ 

```

La

```

    using no-dup n-d1 undef-K
    by (auto simp: clauses-to-update-prop.simps M'
        dest: in-lits-of-l-defined-litD)
  then have ⟨clauses-to-update-inv ?S⟩
    using clss-upd no-dup n-d1 undef-K by (force simp: filter-mset-empty-conv
        dest: in-lits-of-l-defined-litD dest!: no-has-blit-propagate')
}
moreover have ⟨past-invs ?S⟩
  unfolding past-invs.simps
proof (intro conjI impI allI)
  fix M1a M2 K'
  assume M1a': ⟨Propagated (− K) (DECO-clause M) # M1 = M2 @ Decided K' # M1a⟩
  then have M1a: ⟨M1 = tl M2 @ Decided K' # M1a⟩
    by (cases M2) auto
  let ?SM1a = ⟨(M1a, N, U, None, add-mset (DECO-clause M) NP, UP, {#}, {#})⟩
  have
    struct:
    ⟨C ∈ #N + U ⟹ twl-lazy-update M1a C ∧
      watched-literals-false-of-max-level M1a C ∧
      twl-exception-inv (M1a, N, U, None, NP, UP, {#}, {#}) C⟩
  for C
  using past-M1 unfolding past-invs.simps M1a
  by fast+
  have
    confl: ⟨confl-cands-enqueued (M1a, N, U, None, NP, UP, {#}, {#})⟩ and
    propa: ⟨propa-cands-enqueued (M1a, N, U, None, NP, UP, {#}, {#})⟩ and
    clss-to-upd: ⟨clauses-to-update-inv (M1a, N, U, None, NP, UP, {#}, {#})⟩
  using past-M1 unfolding past-invs.simps unfolding M1a
  by fast+
  show ⟨∀ C ∈ #N + U. twl-lazy-update M1a C ∧
    watched-literals-false-of-max-level M1a C ∧
    twl-exception-inv ?SM1a C⟩
  using struct by (simp add: twl-exception-inv.simps)
  show ⟨confl-cands-enqueued ?SM1a⟩
  using confl by auto
  show ⟨propa-cands-enqueued ?SM1a⟩
  using propa by auto
  show ⟨clauses-to-update-inv ?SM1a⟩
  using clss-to-upd by auto
qed
ultimately show ⟨twl-struct-invs ?S⟩
  unfolding twl-struct-invs-def
  by meson
}
{
  assume
    lev-K: ⟨get-level M K < count-decided M⟩ and
    count-dec: ⟨count-decided M > 1⟩
  obtain K1 K2 C where
    filter-M: ⟨filter is-decided M = Decided K1 # Decided K2 # C⟩
  using count-dec
  by (cases ⟨filter is-decided M⟩; cases ⟨tl (filter is-decided M)⟩;
      cases ⟨hd (filter is-decided M)⟩; cases ⟨hd (tl (filter is-decided M))⟩)
  (auto simp: TWL-DECO-clause-def count-decided-def add-mset-eq-add-mset
      filter-eq-Cons-iff tl-append)
}

```

```

then have deco-M:  $\langle \text{TWL-DECO-clause } M = \text{TWL-Clause } \{\#-K1, -K2\} \text{ (uminus } \# \text{ lit-of } \# \text{ mset } C) \rangle$ 
  by (auto simp: TWL-DECO-clause-def)

let ?S =  $\langle (M1, \text{add-mset } (\text{TWL-DECO-clause } M) \ N, \ U, \ \text{None}, \ NP, \ UP, \ \{\#\}, \ \{\#\}) \rangle$ 

have struct-invs-S:  $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv } (\text{state}_W\text{-of } ?S) \rangle$ 
  using struct-invs-add by auto

have no-smaller-S:  $\langle \text{cdcl}_W\text{-restart-mset.no-smaller-propa } (\text{state}_W\text{-of } ?S) \rangle$ 
  using no-smaller-add by simp

obtain MM3 MM2 MM1 where MM:  $\langle M = MM3 \ @ \ \text{Decided } K1 \ \# \ MM2 \ @ \ \text{Decided } K2 \ \# \ MM1 \rangle$ 
and
  [simp]:  $\langle \text{filter is-decided } MM3 = [] \rangle$  and
  [simp]:  $\langle \text{filter is-decided } MM2 = [] \rangle$ 
  using filter-M
  by (auto simp: filter-eq-Cons-iff filter-empty-conv
    eq-commute[of -  $\langle \text{filter is-decided } - \rangle$ ])
then have [simp]:  $\langle \text{count-decided } MM3 = 0 \rangle \langle \text{count-decided } MM2 = 0 \rangle$ 
  by (auto simp: count-decided-0-iff filter-empty-conv
    simp del:  $\langle \text{filter is-decided } MM3 = [] \rangle \langle \text{filter is-decided } MM2 = [] \rangle$ )
have [simp]:  $\langle \text{get-level } M \ K = \text{Suc } (\text{count-decided } M1) \rangle$ 
  using no-dup unfolding M'
  by (auto simp: get-level-skip)
then have [iff]:  $\langle K1 \neq K \rangle$ 
  using lev-K no-dup by (auto simp: MM simp del:  $\langle \text{get-level } M \ K = \text{Suc } (\text{count-decided } M1) \rangle$ )
have  $\langle \text{set } M1 \subseteq \text{set } MM1 \rangle$ 
  using refl[of M] lev-K no-dup[unfolded MM] no-dup[unfolded M']  $\langle \text{count-decided } MM2 = 0 \rangle$ 
   $\langle \text{count-decided } MM3 = 0 \rangle$ 
  apply (subst (asm) M')
  apply (subst (asm) MM)
  by (auto simp: simp del:  $\langle \text{count-decided } MM2 = 0 \rangle \langle \text{count-decided } MM3 = 0 \rangle$ 
    elim!: list-match-lcl-lcl)
then have  $\langle \text{undefined-lit } MM1 \ L \implies \text{undefined-lit } M1 \ L \rangle$  for L
  by (auto simp: Decided-Propagated-in-iff-in-lits-of-l)
then have [iff]:  $\langle K1 \notin \text{lits-of-l } M1 \rangle \langle K2 \notin \text{lits-of-l } M1 \rangle$ 
  using no-dup unfolding MM
  by (auto dest: in-lits-of-l-defined-litD)

have  $\langle \text{struct-wf-twcl-cl } (\text{TWL-DECO-clause } M) \rangle$ 
  using dist-filtered' unfolding deco-M filter-M
  by (auto simp: simp del: clause-TWL-Deco-clause)
moreover have  $\langle \text{twl-lazy-update } M1 \ (\text{TWL-DECO-clause } M) \rangle$ 
  by (auto simp: deco-M add-mset-eq-add-mset)
moreover have  $\langle \text{watched-literals-false-of-max-level } M1 \ (\text{TWL-DECO-clause } M) \rangle$ 
  by (auto simp: deco-M add-mset-eq-add-mset)
ultimately have  $\langle \text{twl-st-inv } ?S \rangle$ 
  using wf-N-U st-invs-M1' unfolding twl-st-inv.simps
  by (auto simp: twl-is-an-exception-def)
moreover have  $\langle \text{valid-enqueued } ?S \rangle$ 
  by auto
moreover have struct-invs-S:  $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv } (\text{state}_W\text{-of } ?S) \rangle$ 
  using struct-invs-add by simp
moreover have  $\langle \text{cdcl}_W\text{-restart-mset.no-smaller-propa } (\text{state}_W\text{-of } ?S) \rangle$ 
  using no-smaller-add by simp

```

```

moreover have  $\langle twl\text{-}st\text{-}exception\text{-}inv\ ?S \rangle$ 
  using  $st\text{-}invs\text{-}M1'$  by (auto simp:  $twl\text{-}exception\text{-}inv.simps\ deco\text{-}M\ add\text{-}mset\text{-}eq\text{-}add\text{-}mset$ )
moreover have  $\langle no\text{-}duplicate\text{-}queued\ ?S \rangle$ 
  by auto
moreover have  $\langle distinct\text{-}queued\ ?S \rangle$ 
  by auto
moreover have  $\langle confl\text{-}cands\text{-}enqueued\ ?S \rangle$ 
  using  $confl\text{-}enqueued\text{-}M1$  by (auto simp:  $deco\text{-}M$ )
moreover have  $\langle propa\text{-}cands\text{-}enqueued\ ?S \rangle$ 
  using  $propa\text{-}enqueued\text{-}M1$ 
  by (auto simp:  $deco\text{-}M\ true\text{-}annots\text{-}true\text{-}cls\text{-}def\text{-}iff\text{-}negation\text{-}in\text{-}model\ Ball\text{-}def$ 
     $dest: in\text{-}lits\text{-}of\text{-}l\text{-}defined\text{-}litD\ in\text{-}diffD$ )
moreover have  $\langle get\text{-}conflict\ ?S = None \rangle$ 
  by simp
moreover {
  have  $\langle get\text{-}level\ M\ L = 0 \implies get\text{-}level\ M1\ L = 0 \rangle$  for  $L$ 
    using  $no\text{-}dup\ defined\text{-}lit\text{-}no\text{-}dupD(1)[of\ M1\ L\ M2]$ 
    by (cases  $\langle defined\text{-}lit\ M\ L \rangle$ )
      (auto simp:  $M'\ defined\text{-}lit\text{-}append\ defined\text{-}lit\text{-}cons\ atm\text{-}of\text{-}eq\text{-}atm\text{-}of$ 
         $get\text{-}level\text{-}cons\text{-}if\ split: if\text{-}splits$ )
    moreover have  $\langle get\text{-}level\ M\ L = 0 \implies L \in lits\text{-}of\text{-}l\ M \implies L \in lits\text{-}of\text{-}l\ M1 \rangle$  for  $L$ 
      using  $no\text{-}dup\ defined\text{-}lit\text{-}no\text{-}dupD(1)[of\ M1\ L\ M2]$ 
      by (cases  $\langle defined\text{-}lit\ M\ L \rangle$ )
        (auto simp:  $M'\ defined\text{-}lit\text{-}append\ defined\text{-}lit\text{-}cons\ atm\text{-}of\text{-}eq\text{-}atm\text{-}of$ 
           $get\text{-}level\text{-}cons\text{-}if\ split: if\text{-}splits\ dest: in\text{-}lits\text{-}of\text{-}l\text{-}defined\text{-}litD$ )
    ultimately have  $\langle entailed\text{-}clss\text{-}inv\ ?S \rangle$ 
      using  $entailed\ unfolding\ entailed\text{-}clss\text{-}inv.simps$  by meson
  }
moreover {
  have  $\langle \neg clauses\text{-}to\text{-}update\text{-}prop\ \{\#\}\ M1\ (L, TWL\text{-}DECO\text{-}clause\ M) \rangle$  for  $L$ 
    by (auto simp:  $deco\text{-}M\ clauses\text{-}to\text{-}update\text{-}prop.simps\ add\text{-}mset\text{-}eq\text{-}add\text{-}mset$ )
  moreover have  $\langle watched\ (TWL\text{-}DECO\text{-}clause\ M) = \{\#L, L'\#\} \implies$ 
     $- L \in lits\text{-}of\text{-}l\ M1 \implies False \rangle$  for  $L\ L'$ 
    by (auto simp:  $deco\text{-}M\ add\text{-}mset\text{-}eq\text{-}add\text{-}mset$ )
  ultimately have  $\langle clauses\text{-}to\text{-}update\text{-}inv\ ?S \rangle$ 
    using  $clss\text{-}upd\ no\text{-}dup$  by (auto simp:  $filter\text{-}mset\text{-}empty\text{-}conv\ clauses\text{-}to\text{-}update\text{-}prop.simps$ 
       $dest: in\text{-}lits\text{-}of\text{-}l\text{-}defined\text{-}litD$ )
  }
moreover have  $\langle past\text{-}invs\ ?S \rangle$ 
  unfolding  $past\text{-}invs.simps$ 
proof (intro conjI impI allI)
  fix  $M1a\ M2\ K'$ 
  assume  $M1a: \langle M1 = M2 @ Decided\ K' \# M1a \rangle$ 
  let  $?SM1a = \langle (M1a, add\text{-}mset\ (TWL\text{-}DECO\text{-}clause\ M)\ N, U, None, NP, UP, \{\#\}, \{\#\}) \rangle$ 
  have
    struct:
     $\langle C \in \#N + U \implies twl\text{-}lazy\text{-}update\ M1a\ C \wedge$ 
       $watched\text{-}literals\text{-}false\text{-}of\text{-}max\text{-}level\ M1a\ C \wedge$ 
       $twl\text{-}exception\text{-}inv\ (M1a, N, U, None, NP, UP, \{\#\}, \{\#\})\ C \rangle$ 
    for  $C$ 
    using  $past\text{-}M1\ unfolding\ past\text{-}invs.simps\ M1a$ 
    by fast+
  then have [iff]:  $\langle K1 \notin lits\text{-}of\text{-}l\ M1a \rangle \langle K2 \notin lits\text{-}of\text{-}l\ M1a \rangle$ 
    using  $\langle K1 \notin lits\text{-}of\text{-}l\ M1 \rangle \langle K2 \notin lits\text{-}of\text{-}l\ M1 \rangle$  unfolding  $M1a$ 
    by (auto dest:  $in\text{-}lits\text{-}of\text{-}l\text{-}defined\text{-}litD$ )
  have

```

```

  confl: ⟨confl-cands-enqueued (M1a, N, U, None, NP, UP, {#}, {#})⟩ and
  propa: ⟨propa-cands-enqueued (M1a, N, U, None, NP, UP, {#}, {#})⟩ and
  cls-to-upd: ⟨clauses-to-update-inv (M1a, N, U, None, NP, UP, {#}, {#})⟩
  using past-M1 unfolding past-invs.simps unfolding M1a
  by fast+
show ⟨∀ C ∈ #add-mset (TWL-DECO-clause M) N + U. twl-lazy-update M1a C ∧
  watched-literals-false-of-max-level M1a C ∧
  twl-exception-inv ?SM1a C⟩
  using struct by (auto simp add: twl-exception-inv.simps deco-M add-mset-eq-add-mset)
show ⟨confl-cands-enqueued ?SM1a⟩
  using confl by (auto simp: deco-M)
show ⟨propa-cands-enqueued ?SM1a⟩
  using propa by (auto simp: deco-M)
have [iff]: ⟨¬ clauses-to-update-prop {#} M1a
  (L, TWL-Clause {#- K1, - K2#}
    {#- lit-of x. x ∈ # mset C#})⟩ for L
  by (auto simp: clauses-to-update-prop.simps add-mset-eq-add-mset)
show ⟨clauses-to-update-inv ?SM1a⟩
  using cls-to-upd by (auto simp: deco-M add-mset-eq-add-mset)
qed
ultimately show ⟨twl-struct-invs (M1, add-mset (TWL-DECO-clause M) N, U, None, NP, UP,
{#}, {#})⟩
  unfolding twl-struct-invs-def
  by meson
}
qed

```

lemma *get-all-ann-decomposition-count-decided-1*:

```

  assumes
    decomp: ⟨(Decided K # M1, M2) ∈ set (get-all-ann-decomposition M)⟩ and
    count-dec: ⟨count-decided M = 1⟩
  shows ⟨M = M2 @ Decided K # M1⟩

```

proof –

obtain M3 **where**

```

  M: ⟨M = M3 @ M2 @ Decided K # M1⟩

```

```

  using decomp by blast

```

then have M': ⟨M = (M3 @ M2) @ Decided K # M1⟩

```

  by simp

```

have count-dec-M1: ⟨count-decided M1 = 0⟩

```

  using count-dec unfolding M'

```

```

  by (auto simp: count-decided-0-iff)

```

have [simp]: ⟨length (get-all-ann-decomposition (M3 @ M2)) = Suc 0⟩

```

  ⟨length (get-all-ann-decomposition M1) = Suc 0⟩

```

```

  using count-dec unfolding M'

```

```

  by (subst no-decision-get-all-ann-decomposition; auto simp: count-decided-0-iff; fail)+

```

have ⟨length (get-all-ann-decomposition M) = 2⟩

```

  using count-dec

```

```

  unfolding M' cdclW-restart-mset.length-get-all-ann-decomposition-append-Decided

```

```

  by auto

```

moreover have ⟨get-all-ann-decomposition M = [(a, b), (Decided K # M1, M2)] ⇒ False⟩ **for** a b

```

  using decomp get-all-ann-decomposition-hd-hd[of M ⟨fst (hd (get-all-ann-decomposition M))⟩

```

```

  ⟨snd (hd (get-all-ann-decomposition M))⟩ ⟨fst ((hd o tl) (get-all-ann-decomposition M))⟩

```

```

  ⟨snd ((hd o tl) (get-all-ann-decomposition M))⟩ Nil] count-dec

```

```

  get-all-ann-decomposition-exists-prepend[of a b M]

```

```

  by (cases ⟨get-all-ann-decomposition M⟩; cases ⟨tl (get-all-ann-decomposition M)⟩;

```

cases $\langle \text{fst } ((\text{hd } o \text{ tl}) (\text{get-all-ann-decomposition } M)) \rangle$; cases a)
 (auto simp: count-decided-0-iff)
ultimately have $\langle \text{get-all-ann-decomposition } M = [(\text{Decided } K \# M1, M2), ([], M1)] \rangle$
using *decomp* $\text{get-all-ann-decomposition-hd-hd}$ [of $M \langle \text{fst } (\text{hd } (\text{get-all-ann-decomposition } M)) \rangle$
 $\langle \text{snd } (\text{hd } (\text{get-all-ann-decomposition } M)) \rangle \langle \text{fst } ((\text{hd } o \text{ tl}) (\text{get-all-ann-decomposition } M)) \rangle$
 $\langle \text{snd } ((\text{hd } o \text{ tl}) (\text{get-all-ann-decomposition } M)) \rangle \text{Nil}$]
in-get-all-ann-decomposition-decided-or-empty [of $\langle \text{fst } ((\text{hd } o \text{ tl}) (\text{get-all-ann-decomposition } M)) \rangle$
 $\langle \text{snd } ((\text{hd } o \text{ tl}) (\text{get-all-ann-decomposition } M)) \rangle M$] *count-dec-M1*
by (cases $\langle \text{get-all-ann-decomposition } M \rangle$; cases $\langle \text{tl } (\text{get-all-ann-decomposition } M) \rangle$;
 cases $\langle \text{fst } ((\text{hd } o \text{ tl}) (\text{get-all-ann-decomposition } M)) \rangle$)
 (auto simp: count-decided-0-iff)

show $\langle ?thesis \rangle$
by (*simp add*: $\langle \text{get-all-ann-decomposition } M = [(\text{Decided } K \# M1, M2), ([], M1)] \rangle$
 $\text{get-all-ann-decomposition-decomp}$)
qed

lemma *negate-model-and-add-twl-twl-stgy-invs*:
assumes
 $\langle \text{negate-model-and-add-twl } S \text{ } T \rangle$ **and**
 $\langle \text{twl-struct-invs } S \rangle$ **and**
 $\langle \text{twl-stgy-invs } S \rangle$
shows $\langle \text{twl-stgy-invs } T \rangle$
using *assms*
proof (*induction rule: negate-model-and-add-twl.induct*)
case (*bj-unit* $K \ M1 \ M2 \ M \ N \ U \ NP \ UP \ WS \ Q$) **note** *decomp* = *this*(1) **and** *lev-K* = *this*(2) **and**
count-dec = *this*(3) **and** *struct* = *this*(4) **and** *stgy* = *this*(5)
let $?S = \langle (M, N, U, \text{None}, NP, UP, WS, Q) \rangle$
let $?T = \langle (\text{Propagated } (- \ K) \ (\text{DECO-clause } M) \# M1, N, U, \text{None}, \text{add-mset } (\text{DECO-clause } M) \ NP, UP,$
 $\{\#\}, \{\#K\# \}) \rangle$
have
false-with-lev: $\langle \text{cdcl}_W\text{-restart-mset.conflict-is-false-with-level } (\text{state}_W\text{-of } ?S) \rangle$ **and**
no-smaller-confl: $\langle \text{cdcl}_W\text{-restart-mset.no-smaller-confl } (\text{state}_W\text{-of } ?S) \rangle$ **and**
confl0: $\langle \text{cdcl}_W\text{-restart-mset.conflict-non-zero-unless-level-0 } (\text{state}_W\text{-of } ?S) \rangle$
using *stgy unfolding twl-stgy-invs-def cdcl_W-restart-mset.cdcl_W-stgy-invariant-def*
by *fast+*
have $M: \langle M = M2 @ \text{Decided } K \# M1 \rangle$
using *decomp count-dec* **by** (*simp add*: *get-all-ann-decomposition-count-decided-1*)
have [*iff*]: $\langle M = M' @ \text{Decided } K' \# Ma \longleftrightarrow M' = M2 \wedge K' = K \wedge Ma = M1 \rangle$ **for** $M' \ K' \ Ma$
using *count-dec unfolding M*
by (*auto elim!*: *list-match-lel-lel*)
have [*iff*]: $\langle M1 = M' @ \text{Decided } K' \# Ma \longleftrightarrow \text{False} \rangle$ **for** $M' \ K' \ Ma$
using *count-dec unfolding M*
by (*auto elim!*: *list-match-lel-lel*)
have
false-with-lev: $\langle \text{cdcl}_W\text{-restart-mset.conflict-is-false-with-level } (\text{state}_W\text{-of } ?T) \rangle$
using *false-with-lev unfolding cdcl_W-restart-mset.no-smaller-confl-def*
by (*auto simp*: *cdcl_W-restart-mset-state clauses-def*)
moreover have $\langle \text{cdcl}_W\text{-restart-mset.no-smaller-confl } (\text{state}_W\text{-of } ?T) \rangle$
using *no-smaller-confl unfolding cdcl_W-restart-mset.no-smaller-confl-def*
by (*auto simp*: *cdcl_W-restart-mset-state clauses-def*
cdcl_W-restart-mset.propagated-cons-eq-append-decide-cons
dest!: *multi-member-split*)
moreover have $\langle \text{cdcl}_W\text{-restart-mset.conflict-non-zero-unless-level-0 } (\text{state}_W\text{-of } ?T) \rangle$
using *no-smaller-confl unfolding cdcl_W-restart-mset.conflict-non-zero-unless-level-0-def*


```

    by (auto simp: cdclW-restart-mset-state clauses-def
        cdclW-restart-mset.propagated-cons-eq-append-decide-cons
        dest!: multi-member-split)
ultimately show ?case
  unfolding twl-stgy-invs-def cdclW-restart-mset.cdclW-stgy-invariant-def
  by (auto simp: cdclW-restart-mset-state clauses-def)
next
case (bj-nonunit K M1 M2 M N U NP UP WS Q) note decomp = this(1) and lev-K = this(2) and
  count-dec = this(3) and struct = this(4) and stgy = this(5)
let ?S = ⟨(M, N, U, None, NP, UP, WS, Q)⟩
let ?T = ⟨(Propagated (− K) (DECO-clause M) # M1, add-mset (TWL-DECO-clause M) N, U,
  None, NP, UP, {#}, {#K#})⟩
have
  false-with-lev: ⟨cdclW-restart-mset.conflict-is-false-with-level (stateW-of ?S)⟩ and
  no-smaller-confl: ⟨cdclW-restart-mset.no-smaller-confl (stateW-of ?S)⟩ and
  confl0: ⟨cdclW-restart-mset.conflict-non-zero-unless-level-0 (stateW-of ?S)⟩
  using stgy unfolding twl-stgy-invs-def cdclW-restart-mset.cdclW-stgy-invariant-def
  by fast+
obtain M3 where M: ⟨M = M3 @ M2 @ Decided K # M1⟩
  using decomp by auto
have ⟨no-dup M⟩
  using struct unfolding twl-struct-invs-def cdclW-restart-mset.cdclW-all-struct-inv-def
  cdclW-restart-mset.cdclW-M-level-inv-def trail.simps stateW-of.simps
  by fast
then have H: ⟨M1 = M' @ Decided Ka # M2 ⟹ ¬M2 ⊢as CNot (DECO-clause M)⟩ for M' Ka
M2
  by (auto simp: M DECO-clause-def
      dest: in-lits-of-l-defined-litD in-diffD)
have
  false-with-lev: ⟨cdclW-restart-mset.conflict-is-false-with-level (stateW-of ?T)⟩
  using false-with-lev unfolding cdclW-restart-mset.no-smaller-confl-def
  by (auto simp: cdclW-restart-mset-state clauses-def)
moreover have ⟨cdclW-restart-mset.no-smaller-confl (stateW-of ?T)⟩
  using no-smaller-confl H unfolding cdclW-restart-mset.no-smaller-confl-def
  by (auto simp: cdclW-restart-mset-state clauses-def M
      cdclW-restart-mset.propagated-cons-eq-append-decide-cons
      dest!: multi-member-split)
moreover have ⟨cdclW-restart-mset.conflict-non-zero-unless-level-0 (stateW-of ?T)⟩
  using no-smaller-confl unfolding cdclW-restart-mset.conflict-non-zero-unless-level-0-def
  by (auto simp: cdclW-restart-mset-state clauses-def
      cdclW-restart-mset.propagated-cons-eq-append-decide-cons
      dest!: multi-member-split)
ultimately show ?case
  unfolding twl-stgy-invs-def cdclW-restart-mset.cdclW-stgy-invariant-def by fast
next
case (restart-nonunit K M1 M2 M N U NP UP WS Q) note decomp = this(1) and lev-K = this(2)
and
  count-dec = this(3) and struct = this(4) and stgy = this(5)
let ?S = ⟨(M, N, U, None, NP, UP, WS, Q)⟩
let ?T = ⟨(M1, add-mset (TWL-DECO-clause M) N, U, None, NP, UP, {#}, {#})⟩
have
  false-with-lev: ⟨cdclW-restart-mset.conflict-is-false-with-level (stateW-of ?S)⟩ and
  no-smaller-confl: ⟨cdclW-restart-mset.no-smaller-confl (stateW-of ?S)⟩ and
  confl0: ⟨cdclW-restart-mset.conflict-non-zero-unless-level-0 (stateW-of ?S)⟩
  using stgy unfolding twl-stgy-invs-def cdclW-restart-mset.cdclW-stgy-invariant-def
  by fast+

```

obtain $M3$ **where** M : $\langle M = M3 @ M2 @ \text{Decided } K \# M1 \rangle$
using *decomp* **by** *auto*
have $\langle \text{no-dup } M \rangle$
using *struct unfolding twl-struct-invs-def cdcl_W-restart-mset.cdcl_W-all-struct-inv-def*
cdcl_W-restart-mset.cdcl_W-M-level-inv-def trail.simps state_W-of.simps
by *fast*
then have H : $\langle M1 = M' @ \text{Decided } Ka \# M2 \implies \neg M2 \models_{as} \text{CNot } (\text{DECO-clause } M) \rangle$ **for** $M' Ka$
 $M2$
by (*auto simp: M DECO-clause-def*
dest: in-lits-of-l-defined-litD in-diffD)
have
false-with-lev: $\langle \text{cdcl}_W\text{-restart-mset.conflict-is-false-with-level } (\text{state}_W\text{-of } ?T) \rangle$
using *false-with-lev unfolding cdcl_W-restart-mset.no-smaller-confl-def*
by (*auto simp: cdcl_W-restart-mset-state clauses-def*)
moreover have $\langle \text{cdcl}_W\text{-restart-mset.no-smaller-confl } (\text{state}_W\text{-of } ?T) \rangle$
using *no-smaller-confl H unfolding cdcl_W-restart-mset.no-smaller-confl-def*
by (*auto simp: cdcl_W-restart-mset-state clauses-def M*
cdcl_W-restart-mset.propagated-cons-eq-append-decide-cons
dest!: multi-member-split)
moreover have $\langle \text{cdcl}_W\text{-restart-mset.conflict-non-zero-unless-level-0 } (\text{state}_W\text{-of } ?T) \rangle$
using *no-smaller-confl unfolding cdcl_W-restart-mset.conflict-non-zero-unless-level-0-def*
by (*auto simp: cdcl_W-restart-mset-state clauses-def*
cdcl_W-restart-mset.propagated-cons-eq-append-decide-cons
dest!: multi-member-split)
ultimately show *?case*
unfolding *twl-stgy-invs-def cdcl_W-restart-mset.cdcl_W-stgy-invariant-def* **by** *fast*
qed

lemma *cdcl-tw-stgy-cdcl_W-learned-clauses-entailed-by-init:*
assumes
 $\langle \text{cdcl-tw-stgy } S \ s \rangle$ **and**
 $\langle \text{twl-struct-invs } S \rangle$ **and**
 $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init } (\text{state}_W\text{-of } S) \rangle$
shows
 $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init } (\text{state}_W\text{-of } s) \rangle$
by (*meson assms cdcl_W-restart-mset.cdcl_W-all-struct-inv-def*
cdcl_W-restart-mset.rtranclp-cdcl_W-learned-clauses-entailed
cdcl_W-restart-mset.rtranclp-cdcl_W-stgy-rtranclp-cdcl_W-restart
cdcl-tw-stgy-cdcl_W-stgy twl-struct-invs-def)

lemma *rtranclp-cdcl-tw-stgy-cdcl_W-learned-clauses-entailed-by-init:*
assumes
 $\langle \text{cdcl-tw-stgy}^{**} S \ s \rangle$ **and**
 $\langle \text{twl-struct-invs } S \rangle$ **and**
 $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init } (\text{state}_W\text{-of } S) \rangle$
shows
 $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init } (\text{state}_W\text{-of } s) \rangle$
using *assms*
by (*induction rule: rtranclp-induct*)
(auto intro: cdcl-tw-stgy-cdcl_W-learned-clauses-entailed-by-init
rtranclp-cdcl-tw-stgy-tw-struct-invs)

lemma *negate-model-and-add-tw-cdcl_W-learned-clauses-entailed-by-init:*
assumes
 $\langle \text{negate-model-and-add-tw } S \ s \rangle$ **and**

```

    ⟨twl-struct-invs S⟩ and
    ⟨cdclW-restart-mset.cdclW-learned-clauses-entailed-by-init (stateW-of S)⟩
  shows
    ⟨cdclW-restart-mset.cdclW-learned-clauses-entailed-by-init (stateW-of s)⟩
  using assms
  by (induction rule: negate-model-and-add-twl.induct)
    (auto simp: cdclW-restart-mset.cdclW-learned-clauses-entailed-by-init-def
      cdclW-restart-mset-state)

end

theory Watched-Literals-Algorithm-Enumeration
  imports Watched-Literals.Watched-Literals-Algorithm Watched-Literals-Transition-System-Enumeration
begin

definition cdcl-twl-enum-inv :: ⟨'v twl-st ⇒ bool⟩ where
  ⟨cdcl-twl-enum-inv S ⟷ twl-struct-invs S ∧ twl-stgy-invs S ∧ final-twl-state S ∧
```

$$\text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init } (\text{state}_W\text{-of } S) \rangle$$

```

definition mod-restriction :: ⟨'v clauses ⇒ 'v clauses ⇒ bool⟩ where
  ⟨mod-restriction N N' ⟷
    (∀ M. M ⊨sm N ⟶ M ⊨sm N') ∧
    (∀ M. total-over-m M (set-mset N') ⟶ consistent-interp M ⟶ M ⊨sm N' ⟶ M ⊨sm N)⟩

lemma mod-restriction-satisfiable-iff:
  ⟨mod-restriction N N' ⟷ satisfiable (set-mset N) ⟷ satisfiable (set-mset N')⟩
apply (auto simp: mod-restriction-def satisfiable-carac[symmetric])
by (meson satisfiable-carac satisfiable-def true-clss-set-mset)

definition enum-mod-restriction-st-clss :: ⟨('v twl-st × ('v literal list option × 'v clauses)) set⟩ where
  ⟨enum-mod-restriction-st-clss = {(S, (M, N)). mod-restriction (get-all-init-clss S) N ∧
    twl-struct-invs S ∧ twl-stgy-invs S ∧
    cdclW-restart-mset.cdclW-learned-clauses-entailed-by-init (stateW-of S) ∧
    atms-of-mm (get-all-init-clss S) = atms-of-mm N}⟩

definition enum-model-st-direct :: ⟨('v twl-st × ('v literal list option × 'v clauses)) set⟩ where
  ⟨enum-model-st-direct = {(S, (M, N)).
```

$$\text{mod-restriction } (\text{get-all-init-clss } S) \ N \wedge$$

$$(\text{get-conflict } S = \text{None} \longrightarrow M \neq \text{None} \wedge \text{lit-of } \# \text{ mset } (\text{get-trail } S) = \text{mset } (\text{the } M)) \wedge$$

$$(\text{get-conflict } S \neq \text{None} \longrightarrow M = \text{None}) \wedge$$

$$\text{atms-of-mm } (\text{get-all-init-clss } S) = \text{atms-of-mm } N \wedge$$

$$(\text{get-conflict } S = \text{None} \longrightarrow \text{next-model } (\text{map lit-of } (\text{get-trail } S)) \ N) \wedge$$

$$\text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init } (\text{state}_W\text{-of } S) \wedge$$

$$\text{cdcl-twl-enum-inv } S \rangle$$

```

definition enum-model-st :: ⟨((bool × 'v twl-st) × ('v literal list option × 'v clauses)) set⟩ where
  ⟨enum-model-st = {(b, S), (M, N)).
```

$$\text{mod-restriction } (\text{get-all-init-clss } S) \ N \wedge$$

$$(b \longrightarrow \text{get-conflict } S = \text{None} \wedge M \neq \text{None} \wedge \text{lits-of-l } (\text{get-trail } S) = \text{set } (\text{the } M)) \wedge$$

$$(\text{get-conflict } S \neq \text{None} \longrightarrow \neg b \wedge M = \text{None}) \rangle$$

```

fun add-to-init-cls :: ⟨'v twl-cl ⇒ 'v twl-st ⇒ 'v twl-st⟩ where
  ⟨add-to-init-cls C (M, N, U, D, NE, UE, WS, Q) = (M, add-mset C N, U, D, NE, UE, WS, Q)⟩

lemma cdcl-twl-stgy-final-twl-stateE:

```

assumes
 $\langle \text{cdcl_twl_stgy}^{**} S T \rangle$ **and**
 $\langle \text{final_twl_state } T \rangle$ **and**
 $\langle \text{twl_struct_invs } S \rangle$ **and**
 $\langle \text{twl_stgy_invs } S \rangle$ **and**
 $\text{ent}: \langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init (state}_W\text{-of } S) \rangle$ **and**
 $\text{Hunsat}: \langle \text{get-conflict } T \neq \text{None} \implies \text{unsatisfiable (set-mset (get-all-init-clss } S)) \implies P \rangle$ **and**
 $\text{Hsat}: \langle \text{get-conflict } T = \text{None} \implies \text{consistent-interp (lits-of-l (get-trail } T)) \implies$
 $\text{no-dup (get-trail } T) \implies \text{atm-of ' (lits-of-l (get-trail } T)) \subseteq \text{atms-of-mm (get-all-init-clss } T) \implies$
 $\text{get-trail } T \models_{\text{asm}} \text{get-all-init-clss } S \implies \text{satisfiable (set-mset (get-all-init-clss } S)) \implies P \rangle$
shows P
proof –
have $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy}^{**} (\text{state}_W\text{-of } S) (\text{state}_W\text{-of } T) \rangle$
by (*simp add: assms(1) assms(3) rtranclp-cdcl-tw-stgy-cdcl_W-stgy*)
have $\text{all-struct-T}: \langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv (state}_W\text{-of } T) \rangle$
using *assms(1) assms(3) rtranclp-cdcl-tw-stgy-tw-struct-invs tw-struct-invs-def* **by** *blast*
then have
 $\text{M-leve}: \langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-M-level-inv (state}_W\text{-of } T) \rangle$ **and**
 $\text{alien}: \langle \text{cdcl}_W\text{-restart-mset.no-strange-atm (state}_W\text{-of } T) \rangle$
unfolding *cdcl_W-restart-mset.cdcl_W-all-struct-inv-def* **by** *fast+*

have $\text{ent}' : \langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init (state}_W\text{-of } T) \rangle$
by (*meson* $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy}^{**} (\text{state}_W\text{-of } S) (\text{state}_W\text{-of } T) \rangle$ *assms(3)*
 $\text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv-def}$
 $\text{cdcl}_W\text{-restart-mset.rtranclp-cdcl}_W\text{-learned-clauses-entailed}$
 $\text{cdcl}_W\text{-restart-mset.rtranclp-cdcl}_W\text{-stgy-rtranclp-cdcl}_W\text{-restart ent tw-struct-invs-def}$)
have [*simp*]: $\langle \text{get-all-init-clss } T = \text{get-all-init-clss } S \rangle$
by (*metis* *assms(1) rtranclp-cdcl-tw-stgy-all-learned-diff-learned*)
have $\text{stgy-T}: \langle \text{twl-stgy-invs } T \rangle$
using *assms(1) assms(3) assms(4) rtranclp-cdcl-tw-stgy-tw-stgy-invs* **by** *blast*
consider
 $\langle \text{conf} \rangle \langle \text{count-decided (get-trail } T) = 0 \rangle$ **and** $\langle \text{get-conflict } T \neq \text{None} \rangle$ |
 $\langle \text{sat} \rangle \langle \text{no-step cdcl-tw-stgy } T \rangle$ **and** $\langle \text{get-conflict } T = \text{None} \rangle$ |
 $\langle \text{unsat} \rangle \langle \text{no-step cdcl-tw-stgy } T \rangle$ **and** $\langle \text{get-conflict } T \neq \text{None} \rangle$
using *final unfolding final-tw-state-def*
by *fast*
then show *?thesis*
proof *cases*
case *conf*
then show *?thesis*
using *conflict-of-level-unsatisfiable[OF all-struct-T] ent'*
by (*auto simp: twl-st intro!: Hunsat*)
next
case *sat*
have $\langle \text{no-step cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy (state}_W\text{-of } T) \rangle$
using *assms(1) assms(3) no-step-cdcl-tw-stgy-no-step-cdcl_W-stgy*
 $\text{rtranclp-cdcl-tw-stgy-tw-struct-invs sat(1)}$ **by** *blast*
from $\text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-stgy-final-state-conclusive2[OF this]}$
have $\langle \text{get-trail } T \models_{\text{asm}} \text{cdcl}_W\text{-restart-mset.clauses (state}_W\text{-of } T) \rangle$
using *sat all-struct-T*
unfolding *cdcl_W-restart-mset.cdcl_W-all-struct-inv-def*
by (*auto simp: twl-st*)
then have $\text{tr-T}: \langle \text{get-trail } T \models_{\text{asm}} \text{get-all-init-clss } T \rangle$
by (*cases T*) (*auto simp: clauses-def*)
show *?thesis*
apply (*rule Hsat*)

```

subgoal using sat by auto
subgoal using M-lev unfolding cdclW-restart-mset.cdclW-M-level-inv-def
  by (auto simp: twl-st)
subgoal
  using tr-T M-lev unfolding cdclW-restart-mset.cdclW-M-level-inv-def by (auto simp: twl-st)
subgoal using alien unfolding cdclW-restart-mset.no-strange-atm-def by (auto simp: twl-st)
subgoal using tr-T by auto
subgoal using tr-T M-lev unfolding cdclW-restart-mset.cdclW-M-level-inv-def
  by (auto simp: satisfiable-carac[symmetric] twl-st true-annots-true-cls)
done
next
case unsat
have ⟨no-step cdclW-restart-mset.cdclW-stgy (stateW-of T)⟩
  using assms(1) assms(3) no-step-cdcl-tw-stgy-no-step-cdclW-stgy
  rtranclp-cdcl-tw-stgy-tw-struct-invs unsat(1) by blast
from cdclW-restart-mset.cdclW-stgy-final-state-conclusive2[OF this]
have unsat': ⟨unsatisfiable (set-mset (cdclW-restart-mset.clauses (stateW-of T)))⟩
  using unsat all-struct-T stgy-T
  unfolding cdclW-restart-mset.cdclW-all-struct-inv-def twl-stgy-invs-def
  cdclW-restart-mset.cdclW-stgy-invariant-def
  by (auto simp: twl-st)
have unsat': ⟨unsatisfiable (set-mset (get-all-init-clss T))⟩
proof (rule ccontr)
  assume ¬ ?thesis
  then obtain I where
    cons: ⟨consistent-interp I⟩ and
    I: ⟨I ⊨sm get-all-init-clss T⟩ and
    tot: ⟨total-over-m I (set-mset (get-all-init-clss T))⟩
    unfolding satisfiable-def by blast
  have [simp]: ⟨cdclW-restart-mset.clauses (stateW-of T) = get-all-init-clss T + get-all-learned-clss
T)
    by (cases T) (auto simp: clauses-def)
  moreover have ⟨total-over-m I (set-mset (cdclW-restart-mset.clauses (stateW-of T)))⟩
    using alien tot unfolding cdclW-restart-mset.no-strange-atm-def
    by (auto simp: cdclW-restart-mset-state total-over-m-alt-def twl-st)
  ultimately have ⟨I ⊨sm cdclW-restart-mset.clauses (stateW-of T)⟩
    using ent' I cons unfolding cdclW-restart-mset.cdclW-learned-clauses-entailed-by-init-def
    true-clss-clss-def total-over-m-def
    by (auto simp: clauses-def cdclW-restart-mset-state satisfiable-carac[symmetric] twl-st)
  then show False
    using unsat' cons I by auto
qed
show ?thesis
  apply (rule Hunsat)
  subgoal using unsat by auto
  subgoal using unsat' by auto
  done
qed
qed

```

context

fixes $P :: \langle 'v \text{ literal set} \Rightarrow \text{bool} \rangle$

begin

fun negate-model-and-add :: $\langle 'v \text{ literal list option} \times 'v \text{ clauses} \Rightarrow - \times 'v \text{ clauses} \rangle$ where

```

⟨negate-model-and-add (Some M, N) =
  (if P (set M) then (Some M, N)
   else (None, add-mset (uminus '# mset M) N))⟩ |
⟨negate-model-and-add (None, N) = (None, N)⟩

```

The code below is a little tricky to get right (in a way that can be easily refined later).

There are three cases:

1. the considered clauses are not satisfiable. Then we can conclude that there is no model.
2. the considered clauses are satisfiable and there is at least one decision. Then, we can simply apply *negate-model-and-add-twl*.
3. the considered clauses are satisfiable and there are no decisions. Then we cannot apply *negate-model-and-add-twl*, because that would produce the empty clause that cannot be part of our state (because of our invariants). Therefore, as we know that the model is the last possible model, we break out of the loop and handle test if the model is acceptable outside of the loop.

definition *cdcl-twl-enum* :: ⟨'v twl-st ⇒ bool nres⟩ **where**

```

⟨cdcl-twl-enum S = do {
  S ← conclusive-TWL-run S;
  S ← WHILET cdcl-twl-enum-inv
    (λS. get-conflict S = None ∧ count-decided(get-trail S) > 0 ∧ ¬P (lits-of-l (get-trail S)))
    (λS. do {
      S ← SPEC (negate-model-and-add-twl S);
      conclusive-TWL-run S
    })
  S;
  if get-conflict S = None
  then RETURN (if count-decided(get-trail S) = 0 then P (lits-of-l (get-trail S)) else True)
  else RETURN (False)
}⟩

```

definition *next-model-filtered-nres* **where**

```

⟨next-model-filtered-nres N =
  SPEC (λb. ∃ M. full (next-model-filtered P) N M ∧ b = (fst M ≠ None))⟩

```

lemma *mod-restriction-next-modelD*:

```

⟨mod-restriction N N' ⇒ atms-of-mm N ⊆ atms-of-mm N' ⇒ next-model M N ⇒ next-model M N'⟩

```

by (auto simp: mod-restriction-def next-model.simps)

definition *enum-mod-restriction-st-clss-after* :: ⟨('v twl-st × ('v literal list option × 'v clauses)) set⟩ **where**

```

⟨enum-mod-restriction-st-clss-after = {(S, (M, N)).
  (get-conflict S = None → count-decided (get-trail S) = 0 →
    mod-restriction (add-mset {#} (get-all-init-clss S))
      (add-mset (uminus '# lit-of '# mset (get-trail S)) N)) ∧
  (mod-restriction (get-all-init-clss S) N) ∧
  twl-struct-invs S ∧ twl-stgy-invs S ∧
  (get-conflict S = None → M ≠ None → P (set(the M)) ∧ lit-of '# mset (get-trail S) = mset
    (the M)) ∧
  (get-conflict S ≠ None → M = None) ∧
  cdclW-restart-mset.cdclW-learned-clauses-entailed-by-init (stateW-of S) ∧

```

$atms-of-mm \ (get-all-init-clss \ S) = atms-of-mm \ N \rangle$

lemma $atms-of-uminus-lit-of[simp]$: $\langle atms-of \ \{\#- \text{lit-of } x. x \in \# \ A \# \} = atms-of \ (\text{lit-of } \# \ A) \rangle$
by $(auto \ simp: \ atms-of-def \ image-image)$

lemma $lit-of-mset-eq-mset-setD[dest]$:
 $\langle \text{lit-of } \# \ mset \ M = mset \ aa \implies set \ aa = \text{lit-of } \# \ set \ M \rangle$
by $(metis \ set-image-mset \ set-mset-mset)$

lemma $mod-restriction-add-twice[simp]$:
 $\langle mod-restriction \ A \ (add-mset \ C \ (add-mset \ C \ N)) \longleftrightarrow mod-restriction \ A \ (add-mset \ C \ N) \rangle$
by $(auto \ simp: \ mod-restriction-def)$

lemma

assumes

$conf$: $\langle get-conflict \ W = None \rangle$ **and**
 $count-dec$: $\langle count-decided \ (get-trail \ W) = 0 \rangle$ **and**
 $enum-inv$: $\langle cdcl-tw-l-enun-inv \ W \rangle$ **and**
 $mod-rest-U$: $\langle mod-restriction \ (get-all-init-clss \ W) \ N \rangle$ **and**
 $atms-U-U'$: $\langle atms-of-mm \ (get-all-init-clss \ W) = atms-of-mm \ N \rangle$

shows

$final-level0-add-empty-clause$:
 $\langle mod-restriction \ (add-mset \ \{\#\} \ (get-all-init-clss \ W)) \ (add-mset \ \{\#- \text{lit-of } x. x \in \# \ mset \ (get-trail \ W) \# \} \ N) \rangle \ (\text{is } ?A) \ \text{and}$
 $final-level0-add-empty-clause-unsat$:
 $\langle unsatisfiable \ (set-mset \ (add-mset \ \{\#- \text{lit-of } x. x \in \# \ mset \ (get-trail \ W) \# \} \ N)) \rangle \ (\text{is } ?B)$

proof –

have $[simp]$: $\langle DECO-clause \ (get-trail \ W) = \{\#\} \rangle$ **and**
 $[simp]$: $\langle \{unmark \ L \mid L. \text{is-decided } L \wedge L \in set \ (trail \ (state_W-of \ W))\} = \{\} \rangle$
using $count-dec$ **by** $(auto \ simp: \ count-decided-0-iff \ DECO-clause-def \ filter-mset-empty-conv \ twl-st)$
have $struct-W$: $\langle twl-struct-invs \ W \rangle$ **and**
 $ent-W$: $\langle cdcl_W-restart-mset.cdcl_W-learned-clauses-entailed-by-init \ (state_W-of \ W) \rangle$
using $enum-inv$
unfolding $cdcl-tw-l-enun-inv-def$ **by** $blast+$
have $\langle cdcl_W-restart-mset.no-strange-atm \ (state_W-of \ W) \rangle$ **and**
 $decomp$: $\langle all-decomposition-implies-m \ (cdcl_W-restart-mset.clauses \ (state_W-of \ W)) \ (get-all-ann-decomposition \ (trail \ (state_W-of \ W))) \rangle$
using $struct-W$ **unfolding** $twl-struct-invs-def \ cdcl_W-restart-mset.cdcl_W-all-struct-inv-def$
by $fast+$
have $alien-W$: $\langle cdcl_W-restart-mset.no-strange-atm \ (state_W-of \ W) \rangle$
using $struct-W$
unfolding $twl-struct-invs-def \ cdcl_W-restart-mset.cdcl_W-all-struct-inv-def$
by $fast$
have 1 : $\langle set-mset \ (cdcl_W-restart-mset.clauses \ (state_W-of \ W)) \models ps \ unmark-l \ (trail \ (state_W-of \ W)) \rangle$
using $all-decomposition-implies-propagated-lits-are-implied[OF \ decomp]$
by $simp$
then have 2 : $\langle set-mset \ (get-all-init-clss \ W) \models ps \ unmark-l \ (trail \ (state_W-of \ W)) \rangle$
using $ent-W$ **unfolding** $cdcl_W-restart-mset.cdcl_W-learned-clauses-entailed-by-init-def \ cdcl_W-restart-mset.clauses-def$
by $(fastforce \ simp: \ clauses-def \ twl-st \ dest: \ true-clss-clss-generalise-true-clss-clss)$

have H : $False$

if $M-tr-W$: $\langle M \models \{\#- \text{lit-of } x. x \in \# \ mset \ (get-trail \ W) \# \} \rangle$ **and**

```

  M-U':  $\langle M \models_m N \rangle$  and
  tot:  $\langle \text{total-over-m } M \text{ (set-mset } N) \rangle$  and
  cons:  $\langle \text{consistent-interp } M \rangle$ 
for M
proof -
  have  $\langle M \models_{sm} \text{get-all-init-clss } W \rangle$ 
  using mod-rest-U M-U' cons
  unfolding mod-restriction-def
  apply auto
  using tot apply blast+
  done
moreover have  $\langle \text{total-over-m } M \text{ (set-mset (get-all-init-clss } W) \cup$ 
   $\text{unmark-l (trail (state}_W\text{-of } W))) \rangle$ 
  using alien-W atms-U-U' tot
  unfolding total-over-m-alt-def total-over-set-alt-def
  cdclW-restart-mset.no-strange-atm-def
  by (auto 5 5 dest: atms-of-DECO-clauseD simp: lits-of-def twl-st)
ultimately have  $\langle M \models_s \text{unmark-l (trail (state}_W\text{-of } W)) \rangle$ 
  using 2 cons unfolding true-clss-clss-def
  by auto
then show False
  using cons M-tr-W
  by (auto simp: true-clss-def twl-st true-clss-def consistent-interp-def)
qed
then show ?A
  unfolding mod-restriction-def
  by auto
from mod-restriction-satisfiable-iff[OF this]
show ?B
  by (auto simp: satisfiable-def)
qed

```

lemma *cdcl-twl-enum-next-model-filtered-nres:*

$\langle (\text{cdcl-twl-enum}, \text{next-model-filtered-nres}) \in$
 $[\lambda(M, N). M = \text{None}]_f \text{ enum-mod-restriction-st-clss} \rightarrow \langle \text{bool-rel} \rangle \text{nres-rel} \rangle$

proof -

define *model-if-exists* **where**

$\langle \text{model-if-exists } S \equiv \lambda M.$
 $(\text{if } \exists M. \text{next-model } M \text{ (snd } S)$
 $\text{then (fst } M \neq \text{None} \wedge \text{next-model (the (fst } M)) \text{ (snd } S) \wedge \text{snd } M = \text{snd } S)$
 $\text{else (fst } M = \text{None} \wedge M = S)) \rangle$

for $S :: \langle - \times 'v \text{ clauses} \rangle$

have $\langle \text{full (next-model-filtered } P) S U \longleftrightarrow$

$(\exists T. \text{model-if-exists } S T \wedge \text{full (next-model-filtered } P) (\text{None}, \text{snd } T) U) \rangle$

(is $\langle ?A \longleftrightarrow ?B \rangle$)

if $\langle \text{fst } S = \text{None} \rangle$

for $S U$

proof

assume $?A$

then consider

$(\text{nss}) \langle \text{no-step (next-model-filtered } P) S \rangle \mid$

$(\text{s1}) T \text{ where } \langle \text{next-model-filtered } P) S T \rangle \text{ and } \langle \text{full (next-model-filtered } P) T U \rangle$

unfolding *full-def*

by $(\text{metis converse-rtranclpE})$


```

then show ?B
proof cases
  case nss
  then have SU:  $\langle S = U \rangle$ 
    using ?A
    apply (subst (asm) no-step-full-iff-eq)
    apply assumption by simp
  have  $\langle \text{model-if-exists } S \ S \rangle$  and  $\langle \text{fst } S = \text{None} \rangle$ 
    using nss no-step-next-model-filtered-next-model-iff[of  $\langle -, \text{snd } S \rangle$ ] that
    unfolding model-if-exists-def
    by (cases S; auto; fail)+
  moreover {
    have  $\langle \text{no-step } (\text{next-model-filtered } P) \ (\text{None}, \text{snd } S) \rangle$ 
      using nss
      apply (subst no-step-next-model-filtered-next-model-iff)
      subgoal using that by (cases S) auto
      apply (subst (asm) no-step-next-model-filtered-next-model-iff)
      subgoal using that by (cases S) auto
      unfolding Ex-next-model-iff-satisfiable
      apply (rule unsatisfiable-mono)
      defer
      apply assumption
      by (cases S; cases  $\langle \text{fst } S \rangle$ ) (auto intro: unsatisfiable-mono)
    then have  $\langle \text{full } (\text{next-model-filtered } P) \ (\text{None}, \text{snd } S) \ U \rangle$ 
      apply (subst no-step-full-iff-eq)
      apply assumption
      using SU  $\langle \text{fst } S = \text{None} \rangle$ 
      by (cases S) auto
  }
  ultimately show ?B
    by fast
next
case (s1 T)
obtain M where
  M:  $\langle \text{next-model } M \ (\text{snd } S) \rangle$  and
  T:  $\langle T = (\text{if } P \ (\text{set } M) \ \text{then } (\text{Some } M, \text{snd } S) \ \text{else } (\text{None}, \text{add-mset } (\text{image-mset } \text{uminus } (\text{mset } M)) \ (\text{snd } S))) \rangle$ 
  using s1
  unfolding model-if-exists-def
  apply (cases T)
  apply (auto simp: next-model-filtered.simps)
  done
let ?T =  $\langle ((\text{Some } M, \text{snd } S)) \rangle$ 
have nm:  $\langle \text{model-if-exists } S \ ?T \rangle$ 
  using M T that unfolding model-if-exists-def
  by (cases S) auto
moreover have  $\langle \text{full } (\text{next-model-filtered } P) \ (\text{negate-model-and-add } ?T) \ U \rangle$ 
  using s1(2) T
  by (auto split: if-splits)
moreover have  $\langle \text{next-model-filtered } P \ (\text{None}, \text{snd } ?T) \ (\text{negate-model-and-add } (\text{Some } M, \text{snd } S)) \rangle$ 
  using nm that by (cases S) (auto simp: next-model-filtered.simps model-if-exists-def
    split: if-splits)
ultimately show ?B
proof -
  have  $(\text{None}, \text{snd } (\text{Some } M, \text{snd } S)) = S$ 
    by (metis (no-types) sndI surjective-pairing that)

```

```

    then have full (next-model-filtered P) (None, snd (Some M, snd S)) U
      by (metis (full (next-model-filtered P) S U))
    then show ?thesis
      using (model-if-exists S (Some M, snd S)) by blast
  qed
qed
next
assume ?B
then show ?A
  apply (auto simp: model-if-exists-def full1-is-full full-fullI split: if-splits)
  by (metis prod.exhaust-sel that)
qed
note H = this
have next-model-filtered-nres-alt-def: (next-model-filtered-nres S = do {
  S ← SPEC (model-if-exists S);
  T ← SPEC (λT. full (next-model-filtered P) (None, snd S) T);
  RETURN (fst T ≠ None)
}) if (fst S = None) for S
using that
unfolding next-model-filtered-nres-def RES-RES-RETURN-RES RES-RETURN-RES
H[OF that]
by blast+
have conclusive-run: (conclusive-TWL-run S
  ≤ ↓ {(S, T). (S, T) ∈ enum-model-st-direct ∧ final-twl-state S ∧
    (get-conflict S = None → next-model (map lit-of (get-trail S)) (snd T)) ∧
    (get-conflict S ≠ None → unsatisfiable (set-mset (snd T)))}
    (SPEC (model-if-exists MN)))
  (is (· ≤ ↓ ?spec-twl ·))
if
  S-MN: (S, MN) ∈ enum-mod-restriction-st-clss and
  M: (case MN of (M, N) ⇒ M = None)
for S MN
proof -
  have H: (∃ s' ∈ Collect (model-if-exists MN). (s, s') ∈ enum-model-st-direct ∧ final-twl-state s ∧
    (get-conflict s = None → next-model (map lit-of (get-trail s)) (snd s')) ∧
    (get-conflict s ≠ None → unsatisfiable (set-mset (snd s'))))
  if
    star: (cdcl-twl-stgy** S s) and
    final: (final-twl-state s)
  for s :: ('v twl-st)
proof -
  obtain N where
    [simp]: (MN = (None, N))
  using M by auto
  have [simp]: (get-all-init-clss s = get-all-init-clss S)
  by (metis rtranclp-cdcl-twl-stgy-cdclW-learned-clauses-entailed-by-init that(1))

  have struct-S: (twl-struct-invs S)
  using S-MN unfolding enum-mod-restriction-st-clss-def by blast
  moreover have stgy-S: (twl-stgy-invs S)
  using S-MN unfolding enum-mod-restriction-st-clss-def by blast
  moreover have ent: (cdclW-restart-mset.cdclW-learned-clauses-entailed-by-init (stateW-of S))
  using S-MN unfolding enum-mod-restriction-st-clss-def by blast
  then have ent-s: (cdclW-restart-mset.cdclW-learned-clauses-entailed-by-init (stateW-of s))
  using rtranclp-cdcl-twl-stgy-cdclW-learned-clauses-entailed-by-init star struct-S by blast
  then have enum-inv: (cdcl-twl-enum-inv s)

```

```

using star S-MN final unfolding enum-mod-restriction-st-clss-def cdcl-tw-enum-inv-def
by (auto intro: rtranclp-cdcl-tw-stgy-tw-struct-invs
    rtranclp-cdcl-tw-stgy-tw-stgy-invs)
show ?thesis
  using struct-S stgy-S ent
proof (rule cdcl-tw-stgy-final-tw-stateE[OF star final])
  assume
    confl: ⟨get-conflict s ≠ None⟩ and
    unsat: ⟨unsatisfiable (set-mset (get-all-init-clss S))⟩
  let ?s = ⟨(None, snd MN)⟩
  have s: ⟨(s, ?s) ∈ enum-model-st-direct⟩
    using S-MN confl unsat enum-inv ent star unfolding enum-model-st-def
    by (auto simp: enum-model-st-direct-def enum-mod-restriction-st-clss-def
        intro: rtranclp-cdcl-tw-stgy-cdclW-learned-clauses-entailed-by-init)
  moreover have ⟨model-if-exists MN ?s⟩
    using unsat S-MN unsat-no-step-next-model-filtered[of N P] Ex-next-model-iff-satisfiable[of N]
    unfolding model-if-exists-def
    by (auto simp: enum-mod-restriction-st-clss-def
        mod-restriction-satisfiable-iff)
  moreover have ⟨unsatisfiable (set-mset N)⟩
    using unsat
    using s unfolding enum-model-st-direct-def
    by (auto simp: mod-restriction-satisfiable-iff)
  ultimately show ?thesis
    apply –
    by (rule bezI[of - ⟨?s⟩]) (use confl final in auto)
next
  let ?s = ⟨(Some (map lit-of (get-trail s)), N)⟩
  assume
    confl: ⟨get-conflict s = None⟩ and
    cons: ⟨consistent-interp (lits-of-l (get-trail s))⟩ and
    ent: ⟨get-trail s ⊨asm get-all-init-clss S⟩ and
    sat: ⟨satisfiable (set-mset (get-all-init-clss S))⟩ and
    n-d: ⟨no-dup (get-trail s)⟩ and
    alien: ⟨atm-of ‘ (lits-of-l (get-trail s)) ⊆ atms-of-mm (get-all-init-clss s)⟩
  moreover have nm: ⟨next-model (map lit-of (get-trail s)) N⟩
    ⟨next-model (map lit-of (get-trail s)) (get-all-init-clss s)⟩
    using ent cons n-d S-MN alien
    by (auto simp: next-model.simps true-annots-true-cls lits-of-def
        no-dup-map-lit-of enum-mod-restriction-st-clss-def mod-restriction-def)
  ultimately have s: ⟨(s, ?s) ∈ enum-model-st-direct⟩
    using S-MN enum-inv star ent unfolding enum-model-st-direct-def
    by (auto simp: mod-restriction-satisfiable-iff next-model.simps
        enum-mod-restriction-st-clss-def lits-of-def
        rtranclp-cdcl-tw-stgy-cdclW-learned-clauses-entailed-by-init)
  moreover have ⟨model-if-exists (None, N) (Some (map lit-of (get-trail s)), N)⟩
    using nm by (auto simp: model-if-exists-def
        enum-mod-restriction-st-clss-def
        mod-restriction-satisfiable-iff)
  moreover have ⟨satisfiable (set-mset N)⟩
    using sat
    using s unfolding enum-model-st-direct-def
    by (auto simp: Ex-next-model-iff-satisfiable[symmetric])
  ultimately show ?thesis
    using nm
    apply –

```

```

    by (rule bxI[of - ⟨(Some (map lit-of (get-trail s)), snd MN)⟩])
      (use final confl in auto)
  qed
qed
show ?thesis
  unfolding conclusive-TWL-run-def
  apply (rule RES-refine)
  unfolding mem-Collect-eq prod.simps
  apply (rule H)
  apply fast+
  done
qed
have loop: ⟨WHILET cdcl-tw-l-enun-inv
  (λS. get-conflict S = None ∧ count-decided (get-trail S) > 0 ∧
    ¬P (lits-of-l (get-trail S)))
  (λS. SPEC (negate-model-and-add-tw-l S) ≫≡
    conclusive-TWL-run) T
  ≤ SPEC
  (λy. ∃ x. (y, x) ∈ {(y, x).
    ((get-conflict y ≠ None ∧ fst x = None) ∨
      (fst x ≠ None ∧ P (lits-of-l (get-trail y))) ∧
      (y, x) ∈ enum-mod-restriction-st-clss-after) ∨
      (get-conflict y = None ∧ count-decided (get-trail y) = 0 ∧
        ¬P (lits-of-l (get-trail y)) ∧ fst x = None ∧
        (y, (None, remove1-mset (uminus ‘# lit-of ‘# mset (get-trail y)) (snd x)))
        ∈ enum-mod-restriction-st-clss-after))
    } ∧
    full (next-model-filtered P) (None, snd M) x)
  (is ⟨WHILET⁻ ?Cond - - ≤ SPEC ?Spec)
  is ⟨- ≤ SPEC (λy. ∃ x. (y, x) ∈ ?Res ∧ ?Full x)⟩)
if
  MN: ⟨case MN of (M, N) ⇒ M = None⟩ and
  S: ⟨(S, MN) ∈ enum-mod-restriction-st-clss⟩ and
  T: ⟨(T, M) ∈ ?spec-tw-l⟩ and
  M: ⟨M ∈ Collect (model-if-exists MN)⟩
for S T :: ‘v tw-l-st⟩ and MN M
proof -
  define R where
    R = {(T :: ‘v tw-l-st⟩, S :: ‘v tw-l-st⟩.
      get-conflict S = None ∧ ¬P (lits-of-l (get-trail S)) ∧ get-conflict T = None ∧
        ¬P (lits-of-l (get-trail T)) ∧
        (get-all-init-clss T, get-all-init-clss S) ∈ measure (λN. card (all-models N)))} ∪
    {(T :: ‘v tw-l-st⟩, S :: ‘v tw-l-st⟩.
      get-conflict S = None ∧ ¬P (lits-of-l (get-trail S)) ∧
        (get-conflict T ≠ None ∨ P (lits-of-l (get-trail T))))}
  have wf: ⟨wf R⟩
  unfolding R-def
  apply (subst Un-commute)
  apply (rule wf-Un)
  subgoal
    by (rule wf-no-loop)
    auto
  subgoal
    by (rule wf-if-measure-in-wf[of ⟨measure (λN. card (all-models N))⟩ - ⟨get-all-init-clss⟩])
    auto

```

```

subgoal
  by auto
done
define I where  $\langle I \ s = (\exists x. (s, x) \in \text{enum-mod-restriction-st-clss-after} \wedge$ 
   $(\text{next-model-filtered } P)^{**} (\text{None}, \text{snd } M) (\text{negate-model-and-add } x) \wedge$ 
   $(\text{next-model-filtered } P)^{**} (\text{None}, \text{snd } M) (\text{None}, \text{snd } (\text{negate-model-and-add } x)) \wedge$ 
   $(\text{get-conflict } s = \text{None} \longrightarrow \text{next-model } (\text{map lit-of } (\text{get-trail } s)) (\text{snd } x)) \wedge$ 
   $(\text{get-conflict } s \neq \text{None} \longrightarrow \text{unsatisfiable } (\text{set-mset } (\text{snd } x))) \wedge$ 
   $\text{final-twl-state } s \rangle$  for s
let ?Q =  $\langle \lambda U \ V \ s'. \text{cdcl-twl-enum-inv } s' \wedge \text{final-twl-state } s' \wedge \text{cdcl-twl-stgy}^{**} \ V \ s' \wedge (s', U) \in R \rangle$ 
have
  conc-run:  $\langle \text{conclusive-TWL-run } V \leq \text{SPEC } (?Q \ U \ V) \rangle$ 
  (is ?conc-run is  $\langle - \leq \text{SPEC } ?Q \rangle$ ) and
  inv-I:  $\langle ?Q \ U \ V \ W \Longrightarrow I \ W \rangle$  (is  $\langle - \Longrightarrow ?I \rangle$ )
if
  U:  $\langle \text{cdcl-twl-enum-inv } U \rangle$  and
  confl:  $\langle ?\text{Cond } U \rangle$  and
  neg:  $\langle \text{negate-model-and-add-twl } U \ V \rangle$  and
  I-U:  $\langle I \ U \rangle$ 
for U V W
proof -
{
  have  $\langle \text{clauses-to-update } V = \{\#\} \rangle$ 
    using neg by (auto simp: negate-model-and-add-twl.simps)
  have
    ent-V:  $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init } (\text{state}_W\text{-of } V) \rangle$  and
    struct-U:  $\langle \text{twl-struct-invs } U \rangle$  and
    ent-U:  $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init } (\text{state}_W\text{-of } U) \rangle$ 
    using U unfolding cdcl-twl-enum-inv-def
    using neg negate-model-and-add-twl-cdcl_W-learned-clauses-entailed-by-init by blast+
  have invs-V:  $\langle \text{twl-struct-invs } V \rangle \langle \text{twl-stgy-invs } V \rangle$ 
    using U neg unfolding cdcl-twl-enum-inv-def
    using negate-model-and-add-twl-twl-struct-invs negate-model-and-add-twl-twl-stgy-invs
    by blast+
  have [simp]:  $\langle \text{get-all-init-clss } V = \text{add-mset } (\text{DECO-clause } (\text{get-trail } U)) (\text{get-all-init-clss } U) \rangle$ 
    using neg by (auto simp: negate-model-and-add-twl.simps)

  have next-mod-U:  $\langle \text{next-model } (\text{map lit-of } (\text{get-trail } U)) (\text{get-all-init-clss } U) \rangle$ 
    if None:  $\langle \text{get-conflict } U = \text{None} \rangle$ 
  proof (rule cdcl-twl-stgy-final-twl-stateE[of U U])
    show  $\langle \text{cdcl-twl-stgy}^{**} \ U \ U \rangle$ 
      by simp
    show  $\langle \text{final-twl-state } U \rangle \langle \text{twl-struct-invs } U \rangle \langle \text{twl-stgy-invs } U \rangle$ 
       $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-learned-clauses-entailed-by-init } (\text{state}_W\text{-of } U) \rangle$ 
      using U unfolding cdcl-twl-enum-inv-def by blast+
    show ?thesis
      if  $\langle \text{get-conflict } U \neq \text{None} \rangle$ 
      using that None by blast
    show ?thesis
      if
         $\langle \text{get-conflict } U = \text{None} \rangle$  and
         $\langle \text{consistent-interp } (\text{lits-of-l } (\text{get-trail } U)) \rangle$  and
         $\langle \text{no-dup } (\text{get-trail } U) \rangle$  and
         $\text{incl: } \langle \text{atm-of } ' \text{lits-of-l } (\text{get-trail } U) \subseteq \text{atms-of-mm } (\text{get-all-init-clss } U) \rangle$  and
         $\langle \text{get-trail } U \models_{\text{asm}} \text{get-all-init-clss } U \rangle$  and
         $\langle \text{satisfiable } (\text{set-mset } (\text{get-all-init-clss } U)) \rangle$ 

```

```

    using that that(5) unfolding next-model.simps
    by (auto simp: lits-of-def true-annots-true-cls no-dup-map-lit-of)
qed
have ⟨cdclW-restart-mset.no-strange-atm (stateW-of U)⟩ and
  decomp: ⟨all-decomposition-implies-m (cdclW-restart-mset.clauses (stateW-of U))
    (get-all-ann-decomposition (trail (stateW-of U)))⟩
  using struct-U unfolding twl-struct-invs-def cdclW-restart-mset.cdclW-all-struct-inv-def
  by fast+

have ⟨all-models (add-mset ((uminus o lit-of) ‘# mset (get-trail U)) (get-all-init-clss U)) ⊇
  all-models (add-mset (DECO-clause (get-trail U)) (get-all-init-clss U))⟩
  if None: ⟨get-conflict U = None⟩
proof (rule cdcl-tw-stgy-final-tw-stateE[of U U])
  show ⟨cdcl-tw-stgy** U U⟩
    by simp
  show ⟨final-tw-state U⟩ ⟨twl-struct-invs U⟩ ⟨twl-stgy-invs U⟩
    ⟨cdclW-restart-mset.cdclW-learned-clauses-entailed-by-init (stateW-of U)⟩
    using U unfolding cdcl-tw-enum-inv-def by blast+
  show ?thesis
    if ⟨get-conflict U ≠ None⟩
    using that None by blast
  show ?thesis
    if
      ⟨get-conflict U = None⟩ and
      ⟨consistent-interp (lits-of-l (get-trail U))⟩ and
      ⟨no-dup (get-trail U)⟩ and
      incl: ⟨atm-of ‘lits-of-l (get-trail U) ⊆ atms-of-mm (get-all-init-clss U)⟩ and
      ⟨get-trail U ⊨asm get-all-init-clss U⟩ and
      ⟨satisfiable (set-mset (get-all-init-clss U))⟩
  proof -
    have 1: ⟨I ⊨ {#- lit-of x. x ∈# mset (get-trail U)#}⟩
      if
        I-U: ⟨I ⊨ DECO-clause (get-trail U)⟩
      for I
      by (rule true-cls-mono-set-mset[OF - I-U]) (auto simp: DECO-clause-def)
    have ⟨atms-of (DECO-clause (get-trail U)) ∪ atms-of-mm (get-all-init-clss U) =
      atms-of-mm (get-all-init-clss U)⟩
      using incl by (auto simp: DECO-clause-def lits-of-def atms-of-def)
    then show ?thesis
      by (auto simp: all-models-def 1)
  qed
qed
qed
from card-mono[OF - this]
have card-decr: ⟨card (all-models (add-mset (DECO-clause (get-trail U)) (get-all-init-clss U))) <
  card (all-models (get-all-init-clss U))⟩
  if ⟨get-conflict U = None⟩
  using next-model-decreasing[OF next-mod-U] that by (auto simp: finite-all-models)

{
  fix WW
  assume star: ⟨cdcl-tw-stgy** V WW⟩ and final: ⟨final-tw-state WW⟩
  have ent-W: ⟨cdclW-restart-mset.cdclW-learned-clauses-entailed-by-init (stateW-of WW)⟩
    using U ent-V neg invs-V rtranclp-cdcl-tw-stgy-cdclW-learned-clauses-entailed-by-init
    star
    unfolding cdcl-tw-enum-inv-def by blast
  then have H1: ⟨cdcl-tw-enum-inv WW⟩

```

```

using star final invs-V unfolding cdcl-tw-enum-inv-def
using rtrancp-cdcl-tw-stgy-tw-stgy-invs rtrancp-cdcl-tw-stgy-tw-struct-invs by blast
have init-clss-WW-V[simp]: ⟨get-all-init-clss WW = get-all-init-clss V⟩
by (metis rtrancp-cdcl-tw-stgy-all-learned-diff-learned star)

have next-mod: ⟨next-model (map lit-of (get-trail WW)) (get-all-init-clss WW)⟩
if None: ⟨get-conflict WW = None⟩
using invs-V ent-V
proof (rule cdcl-tw-stgy-final-tw-stateE[OF star final])
show ?thesis
if ⟨get-conflict WW ≠ None⟩
using that None by blast
show ?thesis
if
  ⟨get-conflict WW = None⟩ and
  ⟨consistent-interp (lits-of-l (get-trail WW))⟩ and
  ⟨no-dup (get-trail WW)⟩ and
  ⟨atm-of ‘lits-of-l (get-trail WW) ⊆ atms-of-mm (get-all-init-clss WW)⟩ and
  ⟨get-trail WW ⊨asm get-all-init-clss V⟩ and
  ⟨satisfiable (set-mset (get-all-init-clss V))⟩
using that that(5) unfolding next-model.simps
by (auto simp: lits-of-def true-annots-true-clss no-dup-map-lit-of)
qed
have not-none-unsat: ⟨unsatisfiable (set-mset (get-all-init-clss V))⟩
if None: ⟨get-conflict WW ≠ None⟩
using invs-V ent-V
proof (rule cdcl-tw-stgy-final-tw-stateE[OF star final])
show ?thesis
if ⟨unsatisfiable (set-mset (get-all-init-clss V))⟩
using that None by blast
show ?thesis
if
  ⟨get-conflict WW = None⟩
using that None unfolding next-model.simps
by (auto simp: lits-of-def true-annots-true-clss no-dup-map-lit-of)
qed
have H2: ⟨(WW, U) ∈ R⟩
using confl card-decr unfolding R-def by (auto)
note H1 H2 next-mod init-clss-WW-V not-none-unsat
} note H = this(1,2) and next-mod = this(3) and init-clss-WW-V = this(4) and
not-none-unsat = this(5)

```

```

{
assume ⟨?Q W⟩
then have
  tw-enum: ⟨cdcl-tw-enum-inv W⟩ and
  final: ⟨final-tw-state W⟩ and
  st: ⟨cdcl-tw-stgy** V W⟩ and
  W-U: ⟨(W, U) ∈ R⟩
by blast+
obtain U' where
  U-U': ⟨(U, U') ∈ enum-mod-restriction-st-clss-after⟩ and
  st-M-U': ⟨(next-model-filtered P)** (None, snd M) (negate-model-and-add U')⟩
using I-U unfolding I-def by blast

```

have *1: {⟨unmark L | L. is-decided L ∧ L ∈ set (trail (state_W-of U))⟩ =*

$CNot (DECO\text{-}clause (get\text{-}trail U))$
 by (force simp: DECO-clause-def twl-st CNot-def)
 have ent3-gnerealise: $\langle A \cup B \cup C \models_{ps} D \implies A \models_{ps} B \implies A \cup C \models_{ps} D \rangle$ for $A B C D$
 by (metis Un-absorb inf-sup-aci(5) true-clss-clss-def
 true-clss-clss-generalise-true-clss-clss)

 have $\langle set\text{-}mset (cdcl_W\text{-}restart\text{-}mset.clauses (state_W\text{-}of U)) \cup$
 $CNot (DECO\text{-}clause (get\text{-}trail U)) \models_{ps} unmark\text{-}l (trail (state_W\text{-}of U)) \rangle$
 using all-decomposition-implies-propagated-lits-are-implied[OF decomp]
 unfolding 1 .
 then have 2: $\langle set\text{-}mset (get\text{-}all\text{-}init\text{-}clss U) \cup CNot (DECO\text{-}clause (get\text{-}trail U)) \models_{ps}$
 $unmark\text{-}l (trail (state_W\text{-}of U)) \rangle$
 using ent-U unfolding cdcl_W-restart-mset.cdcl_W-learned-clauses-entailed-by-init-def
 cdcl_W-restart-mset.clauses-def
 by (auto simp: clauses-def twl-st intro: ent3-gnerealise)
 have [simp]: $\langle unmark\text{-}l (get\text{-}trail U) = CNot \{ \# - \text{lit-of } x. x \in \# mset (get\text{-}trail U) \# \} \rangle$
 by (force simp: CNot-def)
 have mod-U: $\langle mod\text{-}restriction (get\text{-}all\text{-}init\text{-}clss U) (snd U') \rangle$ and
 $atms\text{-}U\text{-}U'$: $\langle atms\text{-}of\text{-}mm (get\text{-}all\text{-}init\text{-}clss U) = atms\text{-}of\text{-}mm (snd U') \rangle$
 using U-U' confl unfolding enum-mod-restriction-st-clss-after-def by (cases U'; auto; fail)+
 have alien-U: $\langle cdcl_W\text{-}restart\text{-}mset.no\text{-}strange\text{-}atm (state_W\text{-}of U) \rangle$
 using twl-struct-invs U
 unfolding twl-struct-invs-def cdcl_W-restart-mset.cdcl_W-all-struct-inv-def
 by fast
 have mod-restriction-H: $\langle M \models DECO\text{-}clause (get\text{-}trail U) \rangle$
 if
 $total$: $\langle total\text{-}over\text{-}m M (set\text{-}mset (snd U')) \rangle$ and
 $consistent$: $\langle consistent\text{-}interp M \rangle$ and
 $M\text{-}tr$: $\langle M \models \{ \# - \text{lit-of } x. x \in \# mset (get\text{-}trail U) \# \} \rangle$ and
 $M\text{-}U'$: $\langle M \models_m snd U' \rangle$
 for M
 proof (rule ccontr)
 assume $\neg ?thesis$
 moreover have tot-tr: $\langle total\text{-}over\text{-}m M \{ DECO\text{-}clause (get\text{-}trail U) \} \rangle$
 using alien-U total atms-U-U' unfolding cdcl_W-restart-mset.no-strange-atm-def
 apply (auto simp: twl-st image-iff total-over-m-alt-def lits-of-def
 dest!: atms-of-DECO-clauseD(1))
 apply (metis atms-of-s-def contra-subsetD image-iff in-atms-of-s-decomp)+
 done
 ultimately have $\langle M \models_s CNot (DECO\text{-}clause (get\text{-}trail U)) \rangle$
 by (simp add: total-not-true-clss-true-clss-CNot)
 moreover have $\langle M \models_{sm} get\text{-}all\text{-}init\text{-}clss U \rangle$
 using mod-U total consistent M-U' unfolding mod-restriction-def
 by blast
 moreover have $\langle total\text{-}over\text{-}m M (set\text{-}mset (get\text{-}all\text{-}init\text{-}clss U)) \rangle$
 using total atms-U-U' by (simp add: total-over-m-def)
 moreover have $\langle total\text{-}over\text{-}m M (unmark\text{-}l (trail (state_W\text{-}of U))) \rangle$
 using alien-U tot-tr total atms-U-U' unfolding cdcl_W-restart-mset.no-strange-atm-def
 apply (auto simp: total-over-m-alt-def
 twl-st dest: atms-of-DECO-clauseD)
 by (metis atms-of-uminus-lit-atm-of-lit-of atms-of-uminus-lit-of lits-of-def
 set-mset-mset subsetCE total total-over-m-def total-over-set-def)
 ultimately have $\langle M \models_s unmark\text{-}l (trail (state_W\text{-}of U)) \rangle$
 using 2 total consistent tot-tr unfolding true-clss-clss-def
 by auto
 then show False


```

    using M-tr tot-tr consistent
    by (auto simp: true-clss-def twl-st true-cls-def consistent-interp-def)
qed
have ⟨mod-restriction (get-all-init-clss U) (snd U')⟩
  using U-U' confl unfolding enum-mod-restriction-st-clss-after-def
  by auto
moreover have ⟨M ⊨ {#- lit-of x. x ∈# mset (get-trail U)#}⟩
  if ⟨M ⊨ DECO-clause (get-trail U)⟩ for M
  by (rule true-cls-mono-set-mset[OF - that]) (auto simp: DECO-clause-def)
ultimately have mod-rest-U:
  ⟨mod-restriction (add-mset (DECO-clause (get-trail U)) (get-all-init-clss U))
    (add-mset {#- lit-of x. x ∈# mset (get-trail U)#} (snd U'))⟩
  using 2
  by (auto simp: mod-restriction-def twl-st mod-restriction-H)
have ⟨(next-model-filtered P) (negate-model-and-add U')
  ((negate-model-and-add (Some (map lit-of (get-trail U))), snd U'))⟩
  using confl U-U'
  apply (cases U'; cases ⟨fst U'⟩)
  apply (auto simp: enum-mod-restriction-st-clss-after-def lits-of-def
    eq-commute[of - ⟨mset -⟩] next-model-filtered.simps
    intro!: exI[of - ⟨map lit-of (get-trail U)⟩]
    dest: mset-eq-setD)
  defer
  apply (metis list.set-map mset-eq-setD mset-map)
  using next-mod-U by (auto dest: mod-restriction-next-modelD)
then have ⟨(next-model-filtered P)** (None, snd M)
  ((negate-model-and-add (Some (map lit-of (get-trail U))), snd U'))⟩
  using st-M-U' by simp
moreover {
  have ⟨mod-restriction (add-mset {#} (get-all-init-clss W))
    (add-mset {#- lit-of x. x ∈# mset (get-trail W)#}
      (add-mset {#- lit-of x. x ∈# mset (get-trail U)#} (snd U'))⟩
  if
    confl: ⟨get-conflict W = None⟩ and
    count-dec: ⟨count-decided (get-trail W) = 0⟩
  apply (rule final-level0-add-empty-clause[OF that])
  using ⟨cdcl-tw-enum-inv W ∧ final-tw-state W ∧ cdcl-tw-stgy** V W ∧
    (W, U) ∈ R⟩ mod-rest-U init-clss-WW-V[OF st final] U-U' atms-U-U' alien-U
  unfolding cdcl_W-restart-mset.no-strange-atm-def
  by (auto dest: atms-of-DECO-clauseD(2) simp: twl-st lits-of-def)
  (auto simp: image-image atms-of-def)
then have W: ⟨(W, (negate-model-and-add (Some (map lit-of (get-trail U))), snd U'))
  ∈ enum-mod-restriction-st-clss-after⟩
  using confl init-clss-WW-V[OF st final] twl-enum alien-U atms-U-U' confl
  apply (auto simp: enum-mod-restriction-st-clss-after-def lits-of-def
    cdcl-tw-enum-inv-def mod-rest-U
    dest: atms-of-DECO-clauseD)
  defer
  apply (smt U atms-of-def cdcl-tw-enum-inv-def cdcl-tw-stgy-final-tw-stateE contra-subsetD
    lits-of-def rtranclp.intros(1) set-image-mset set-mset-mset)
  done
} note W = this
moreover have ⟨get-conflict W = None ⟹ 0 < count-decided (get-trail W) ⟹
  next-model (map lit-of (get-trail W))
  (add-mset {#- lit-of x. x ∈# mset (get-trail U)#} (snd U'))⟩
  using W next-mod[OF st] final confl unfolding enum-mod-restriction-st-clss-after-def

```

```

    by (auto simp: mod-restriction-def next-model.simps lits-of-def)
  moreover have ⟨get-conflict W = None ⟹ count-decided (get-trail W) = 0 ⟹
    next-model (map lit-of (get-trail W))
    (add-mset {#- lit-of x. x ∈ # mset (get-trail U)#} (snd U'))⟩
  using W next-mod[OF st] final confl unfolding enum-mod-restriction-st-clss-after-def
  apply (subst (asm)(2) mod-restriction-def)
  by (auto simp: mod-restriction-def next-model.simps lits-of-def)
  moreover have ⟨get-conflict W ≠ None ⟹
    unsatisfiable (set-mset (add-mset {#- lit-of x. x ∈ # mset (get-trail U)#} (snd U'))))⟩
  using W not-none-unsat[OF st] final confl mod-rest-U unfolding enum-mod-restriction-st-clss-after-def
  by (auto simp: lits-of-def dest: mod-restriction-satisfiable-iff
    split: if-splits)
  ultimately have ?I
  using final next-mod[OF st]
  unfolding I-def
  apply -
  apply (rule exI[of - ⟨(negate-model-and-add (Some (map lit-of (get-trail U)), snd U'))⟩])
  using confl
  by (auto simp: lits-of-def)
} note I = this
note H and I
} note H = this(1,2) and I = this(3)
then show ?conc-run
  by (auto simp add: conclusive-TWL-run-def)

show ?I if ⟨?Q W⟩
  using I that
  by (auto simp: I-def)
qed
have neg-neg[simp]: ⟨negate-model-and-add (negate-model-and-add M) = negate-model-and-add M⟩
  by (cases M; cases ⟨fst M⟩; auto)
have [simp]: ⟨(T, a, b) ∈ enum-model-st-direct ⟹ (T, None, b) ∈ enum-mod-restriction-st-clss-after⟩
  for a b
  unfolding enum-model-st-direct-def enum-mod-restriction-st-clss-after-def
  cdcl-twL-enum-inv-def
  by (auto intro!: final-level0-add-empty-clause simp: cdcl-twL-enum-inv-def)
have I-T: ⟨I T⟩
  unfolding I-def
  apply (rule exI[of - ⟨(None, snd M)⟩])
  unfolding neg-neg
  apply (intro conjI)
  subgoal
    using T by (cases M) auto
  subgoal by (auto simp: enum-mod-restriction-st-clss-after-def cdcl-twL-enum-inv-def
    enum-model-st-def enum-model-st-direct-def)
  subgoal by (auto simp: enum-mod-restriction-st-clss-after-def cdcl-twL-enum-inv-def
    enum-model-st-def enum-model-st-direct-def)
  subgoal using T by (auto simp: enum-mod-restriction-st-clss-after-def cdcl-twL-enum-inv-def
    enum-model-st-def enum-model-st-direct-def)
  subgoal using T by (auto simp: enum-mod-restriction-st-clss-after-def cdcl-twL-enum-inv-def
    enum-model-st-def enum-model-st-direct-def)
  done
have final: ⟨?Spec s⟩

```

```

if
  I:  $\langle I \ s \rangle$  and
  cond:  $\langle \neg (\text{?Cond } s) \rangle$  and
  enum:  $\langle \text{cdcl-tw1-enum-inv } s \rangle$ 
for s
proof –
  obtain x where
    sx:  $\langle (s, x) \in \text{enum-mod-restriction-st-clss-after} \rangle$  and
    st':  $\langle (\text{next-model-filtered } P)^{**} (\text{None}, \text{snd } M) (\text{None}, \text{snd } (\text{negate-model-and-add } x)) \rangle$  and
    st:  $\langle (\text{next-model-filtered } P)^{**} (\text{None}, \text{snd } M) (\text{negate-model-and-add } x) \rangle$  and
    final:  $\langle \text{final-tw1-state } s \rangle$  and
    nm:  $\langle \text{get-conflict } s = \text{None} \implies \text{next-model } (\text{map lit-of } (\text{get-trail } s)) (\text{snd } x) \rangle$  and
    unsat:  $\langle \text{get-conflict } s \neq \text{None} \implies \text{unsatisfiable } (\text{set-mset } (\text{snd } x)) \rangle$ 
  using I unfolding I-def by meson
  let ?x =  $\langle \text{if } \text{get-conflict } s = \text{None}$ 
    then  $(\text{Some } (\text{map lit-of } (\text{get-trail } s)), \text{snd } x)$ 
    else  $(\text{None}, \text{snd } x) \rangle$ 
  let ?y =  $\langle \text{negate-model-and-add } ?x \rangle$ 
  have step:  $\langle (\text{next-model-filtered } P) (\text{None}, \text{snd } (\text{negate-model-and-add } x)) ?y \rangle$ 
  if  $\langle \text{get-conflict } s = \text{None} \rangle$  and  $\langle P (\text{lits-of-l } (\text{get-trail } s)) \rangle$ 
  using cond that sx final nm unfolding enum-mod-restriction-st-clss-after-def
    enum-model-st-def
  by (cases x; cases  $\langle \text{fst } x \rangle$ )
    (auto simp: next-model-filtered.simps lits-of-def
      conclusive-TWL-run-def conc-fun-RES
      intro!: exI[of -  $\langle \text{map lit-of } (\text{get-trail } s) \rangle$ ])
  moreover have step:  $\langle (\text{next-model-filtered } P)^{**} (\text{negate-model-and-add } x) ?y \rangle$ 
  if  $\langle \text{get-conflict } s \neq \text{None} \rangle$ 
  using cond that sx unfolding enum-mod-restriction-st-clss-after-def
    enum-model-st-def
  by (cases x; cases  $\langle \text{fst } x \rangle$ )
    (auto simp: next-model-filtered.simps lits-of-def)
  moreover have step:  $\langle (\text{next-model-filtered } P) (\text{negate-model-and-add } x) ?y \vee$ 
     $(\text{negate-model-and-add } x) = ?y \rangle$ 
  if  $\langle \text{get-conflict } s = \text{None} \rangle$  and  $\langle \neg P (\text{lits-of-l } (\text{get-trail } s)) \rangle$ 
  using cond that sx nm unfolding enum-mod-restriction-st-clss-after-def
    enum-model-st-def
  apply (cases x; cases  $\langle \text{fst } x \rangle$ )
  by (auto simp: next-model-filtered.simps lits-of-def
    conclusive-TWL-run-def conc-fun-RES
    intro!: exI[of -  $\langle \text{map lit-of } (\text{get-trail } s) \rangle$ ])
  ultimately have st:  $\langle (\text{next-model-filtered } P)^{**} (\text{None}, \text{snd } M) ?y \rangle$ 
  using st st' by force
  have 1:  $\langle (s, ?x) \in \text{enum-mod-restriction-st-clss-after} \rangle$ 
  if  $\langle \text{count-decided } (\text{get-trail } s) \neq 0 \vee \text{get-conflict } s \neq \text{None} \vee P (\text{lit-of 'set } (\text{get-trail } s)) \rangle$ 
  using sx cond nm that unfolding enum-mod-restriction-st-clss-after-def
    enum-model-st-def
  by (cases x; cases  $\langle \text{fst } x \rangle$ ) (auto simp: lits-of-def)
  have unsat':  $\langle \text{unsatisfiable } (\text{set-mset } (\text{add-mset } \{\# - \text{lit-of } x. x \in \# \text{ mset } (\text{get-trail } s) \# \} (\text{snd } x))) \rangle$ 
  if  $\langle \text{get-conflict } s = \text{None} \rangle$  and  $\langle \text{count-decided } (\text{get-trail } s) = 0 \rangle$  and
     $\langle \neg P (\text{lit-of 'set } (\text{get-trail } s)) \rangle$ 
  apply (rule final-level0-add-empty-clause-unsat)
  using cond that sx nm enum unfolding enum-mod-restriction-st-clss-after-def
    enum-model-st-def apply –
  by (cases x; cases  $\langle \text{fst } x \rangle$ )
    (force simp: next-model-filtered.simps lits-of-def)+

```

```

have ⟨no-step (next-model-filtered P) ?y⟩
  apply (rule unsat-no-step-next-model-filtered')
  apply (cases x; cases ⟨fst x⟩)
  using cond unsat nm unsat' that
  by (auto simp: lits-of-def)
then have 2: ⟨full (next-model-filtered P) (None, snd M) ?y⟩
  using st that unfolding full-def by blast
have 1b: ⟨count-decided (get-trail s) = 0 ⟹
  ¬ P (lit-of ' set (get-trail s)) ⟹
  get-conflict s = None ⟹
  (s, None, snd x) ∈ enum-mod-restriction-st-clss-after⟩
  using that cond unsat nm unsat' sx
  unfolding enum-mod-restriction-st-clss-after-def
  by (cases x; cases ⟨fst x⟩) auto
show ?thesis
  apply (rule exI[of - ⟨?y⟩])
  using 1 1b 2 cond by (auto simp: lits-of-def)
qed
show ?thesis
  apply (refine-vcg WHILEIT-rule-stronger-inv[where R=⟨R⟩ and I' = I] conc-run)
  subgoal by (rule wf)
  subgoal
    using T S unfolding enum-model-st-direct-def enum-mod-restriction-st-clss-def
      cdcl-tw1-enum-inv-def
    by auto
  subgoal by (rule I-T)
  apply assumption
  subgoal by fast
  subgoal by fast
  subgoal by fast
  subgoal by fast
  subgoal by fast
  subgoal by fast
  subgoal for U V W by (rule inv-I)
  subgoal by fast
  subgoal by (rule final)
  done
qed
have H1: ⟨(if get-conflict Sb = None
  then RETURN
    (if count-decided (get-trail Sb) = 0
      then P (lits-of-l (get-trail Sb)) else True)
    else RETURN False)
  ≤ ⇓ bool-rel (RETURN (fst x' ≠ None))⟩
if
  ⟨case y of (M, N) ⇒ M = None⟩ and
  ⟨(Sb, x') ∈ ?Res⟩ and
  ⟨x' ∈ Collect (full (next-model-filtered P) (None, snd Sa))⟩
for x x' Sa Sb S y
using that
by (auto simp: enum-mod-restriction-st-clss-after-def enum-model-st-def
  enum-mod-restriction-st-clss-def lits-of-def split: if-splits)
show ?thesis
  supply if-splits[split]
  unfolding cdcl-tw1-enum-def
  apply (intro freI nres-relI)

```

```

    apply (subst next-model-filtered-nres-alt-def)
    subgoal by auto
    apply (refine-vcg conclusive-run)
    unfolding conc-fun-SPEC
    apply (rule loop; assumption)
    apply (rule H1; assumption)
    done
qed

end

end

theory Watched-Literals-List-Enumeration
  imports Watched-Literals-Algorithm-Enumeration Watched-Literals.Watched-Literals-List
begin

lemma convert-lits-l-filter-decided-uminus:  $\langle (S, S') \in \text{convert-lits-l } M \ N \implies$ 
  map  $(\lambda x. \text{--lit-of } x) (\text{filter is-decided } S') = \text{map } (\lambda x. \text{--lit-of } x) (\text{filter is-decided } S) \rangle$ 
  apply (induction S arbitrary: S')
  subgoal by auto
  subgoal for L S S'
    by (cases S') auto
  done

lemma convert-lits-l-DECO-clause[simp]:
   $\langle (S, S') \in \text{convert-lits-l } M \ N \implies \text{DECO-clause } S' = \text{DECO-clause } S \rangle$ 
  by (auto simp: DECO-clause-def uminus-lit-of-image-mset
    convert-lits-l-filter-decided-uminus simp flip: mset-filter mset-map)

lemma convert-lits-l-TWL-DECO-clause[simp]:
   $\langle (S, S') \in \text{convert-lits-l } M \ N \implies \text{TWL-DECO-clause } S' = \text{TWL-DECO-clause } S \rangle$ 
  by (auto simp: TWL-DECO-clause-def uminus-lit-of-image-mset
    (auto simp: take-map[symmetric] drop-map[symmetric]
      mset-filter[symmetric] convert-lits-l-filter-decided mset-map[symmetric]
      simp del: mset-map))

lemma [twl-st-l]:
   $\langle (S, S') \in \text{twl-st-l } b \implies \text{DECO-clause } (\text{get-trail } S') = \text{DECO-clause } (\text{get-trail-l } S) \rangle$ 
  by (auto simp: twl-st-l-def convert-lits-l-DECO-clause)

lemma [twl-st-l]:
   $\langle (S, S') \in \text{twl-st-l } b \implies \text{TWL-DECO-clause } (\text{get-trail } S') = \text{TWL-DECO-clause } (\text{get-trail-l } S) \rangle$ 
  by (auto simp: twl-st-l-def convert-lits-l-DECO-clause)

lemma DECO-clause-simp[simp]:
   $\langle \text{DECO-clause } (A @ B) = \text{DECO-clause } A + \text{DECO-clause } B \rangle$ 
   $\langle \text{DECO-clause } (\text{Decided } K \ \# \ A) = \text{add-mset } (-K) (\text{DECO-clause } A) \rangle$ 
   $\langle \text{DECO-clause } (\text{Propagated } K \ C \ \# \ A) = \text{DECO-clause } A \rangle$ 
   $\langle (\bigwedge K. K \in \text{set } A \implies \neg \text{is-decided } K) \implies \text{DECO-clause } A = \{\#\} \rangle$ 
  by (auto simp: DECO-clause-def filter-mset-empty-conv)

definition find-decomp-target ::  $\langle \text{nat} \Rightarrow 'v \ \text{twl-st-l} \Rightarrow ('v \ \text{twl-st-l} \times 'v \ \text{literal}) \ \text{nres} \rangle$  where
  find-decomp-target =  $(\lambda i \ S.$ 
    SPEC  $(\lambda (T, K). \exists M2 \ M1. \text{equality-except-trail } S \ T \wedge \text{get-trail-l } T = M1 \wedge$ 
       $(\text{Decided } K \ \# \ M1, M2) \in \text{set } (\text{get-all-ann-decomposition } (\text{get-trail-l } S)) \wedge$ 
       $\text{get-level } (\text{get-trail-l } S) \ K = i) \rangle)$ 

```

fun *propagate-unit-and-add* :: $\langle 'v \text{ literal} \Rightarrow 'v \text{ twl-st} \Rightarrow 'v \text{ twl-st} \rangle$ **where**
 $\langle \text{propagate-unit-and-add } K \ (M, N, U, D, NE, UE, WS, Q) =$
 $(\text{Propagated } (-K) \ \{\#-K\# \} \ \# \ M, N, U, \text{None}, \text{add-mset } \{\#-K\# \} \ NE, UE, \{\#\}, \{\#K\#\}) \rangle$

fun *propagate-unit-and-add-l* :: $\langle 'v \text{ literal} \Rightarrow 'v \text{ twl-st-l} \Rightarrow 'v \text{ twl-st-l} \rangle$ **where**
 $\langle \text{propagate-unit-and-add-l } K \ (M, N, D, NE, UE, WS, Q) =$
 $(\text{Propagated } (-K) \ 0 \ \# \ M, N, \text{None}, \text{add-mset } \{\#-K\# \} \ NE, UE, \{\#\}, \{\#K\#\}) \rangle$

definition *negate-mode-bj-unit-l-inv* :: $\langle 'v \text{ twl-st-l} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{negate-mode-bj-unit-l-inv } S \longleftrightarrow$
 $(\exists (S' :: 'v \text{ twl-st}) \ b. (S, S') \in \text{twl-st-l } b \wedge \text{twl-list-invs } S \wedge \text{twl-stgy-invs } S' \wedge$
 $\text{twl-struct-invs } S' \wedge \text{get-conflict-l } S = \text{None}) \rangle$

definition *negate-mode-bj-unit-l* :: $\langle 'v \text{ twl-st-l} \Rightarrow 'v \text{ twl-st-l nres} \rangle$ **where**
 $\langle \text{negate-mode-bj-unit-l} = (\lambda S. \text{do } \{$
 $\text{ASSERT}(\text{negate-mode-bj-unit-l-inv } S);$
 $(S, K) \leftarrow \text{find-decomp-target } 1 \ S;$
 $\text{RETURN } (\text{propagate-unit-and-add-l } K \ S)$
 $\}) \rangle$

lemma *negate-mode-bj-unit-l*:

fixes $S :: \langle 'v \text{ twl-st-l} \rangle$ **and** $S' :: \langle 'v \text{ twl-st} \rangle$
assumes $\langle \text{count-decided } (\text{get-trail-l } S) = 1 \rangle$ **and**
 $SS': \langle (S, S') \in \text{twl-st-l } b \rangle$ **and**
 $\text{struct-invs}: \langle \text{twl-struct-invs } S' \rangle$ **and**
 $\text{add-inv}: \langle \text{twl-list-invs } S \rangle$ **and**
 $\text{stgy-inv}: \langle \text{twl-stgy-invs } S' \rangle$ **and**
 $\text{confl}: \langle \text{get-conflict-l } S = \text{None} \rangle$

shows

$\langle \text{negate-mode-bj-unit-l } S \leq \Downarrow \{ (S, S''). (S, S'') \in \text{twl-st-l } \text{None} \wedge \text{twl-list-invs } S \wedge$
 $\text{clauses-to-update-l } S = \{\#\} \}$
 $(\text{SPEC } (\text{negate-model-and-add-tw-l } S')) \rangle$

proof –

have $H: \langle \exists y \in \text{Collect } (\text{negate-model-and-add-tw-l } S').$
 $(\text{propagate-unit-and-add-l } x2 \ x1, y)$
 $\in \{ (S, S''). (S, S'') \in \text{twl-st-l } \text{None} \wedge \text{twl-list-invs } S \wedge \text{clauses-to-update-l } S = \{\#\} \} \rangle$

if

$\text{count-dec}: \langle \text{count-decided } (\text{get-trail-l } S) = 1 \rangle$ **and**

$S-S': \langle (S, S') \in \text{twl-st-l } b \rangle$ **and**

$\langle \text{twl-struct-invs } S' \rangle$ **and**

$\langle \text{twl-list-invs } S \rangle$ **and**

$\langle \text{twl-stgy-invs } S' \rangle$ **and**

$x-S: \langle x \in \{ (T, K).$

$\exists M2 \ M1.$

$\text{equality-except-trail } S \ T \wedge$

$\text{get-trail-l } T = M1 \wedge$

$(\text{Decided } K \ \# \ M1, M2)$

$\in \text{set } (\text{get-all-ann-decomposition } (\text{get-trail-l } S)) \wedge$

$\text{get-level } (\text{get-trail-l } S) \ K = 1 \} \rangle$ **and**

$x: \langle x = (x1, x2) \rangle$

for $x :: \langle 'v \text{ twl-st-l} \times 'v \text{ literal} \rangle$ **and** $x1 :: \langle 'v \text{ twl-st-l} \rangle$ **and** $x2 :: \langle 'v \text{ literal} \rangle$

proof –

let $?y0 = \langle (\lambda(M, Oth). (\text{drop } (\text{length } M - \text{length } (\text{get-trail-l } x1)) (\text{get-trail } S'), Oth)) \ S' \rangle$

let $?y1 = \langle \text{propagate-unit-and-add } x2 \ ?y0 \rangle$

```

obtain  $M1\ M2$  where
   $S\text{-}x1$ :  $\langle \text{equality-except-trail } S\ x1 \rangle$  and
   $tr\text{-}M1$ :  $\langle \text{get-trail-l } x1 = M1 \rangle$  and
   $decomp$ :  $\langle (\text{Decided } x2 \# M1, M2) \in \text{set } (\text{get-all-ann-decomposition } (\text{get-trail-l } S)) \rangle$  and
   $lev\text{-}x2$ :  $\langle \text{get-level } (\text{get-trail-l } S)\ x2 = 1 \rangle$ 
  using  $x\text{-}S$  unfolding  $x$  by  $blast$ 
obtain  $M2'$  where
   $decomp'$ :  $\langle (\text{Decided } x2 \# \text{drop } (\text{length } (\text{get-trail } S') - \text{length } M1) (\text{get-trail } S'), M2') \in \text{set } (\text{get-all-ann-decomposition } (\text{get-trail } S')) \rangle$  and
   $conv$ :  $\langle (\text{get-trail-l } S, \text{get-trail } S') \in \text{convert-lits-l } (\text{get-clauses-l } S) (\text{get-unit-clauses-l } S) \rangle$  and
   $conv\text{-}M1$ :  $\langle (M1, \text{drop } (\text{length } (\text{get-trail } S') - \text{length } M1) (\text{get-trail } S')) \in \text{convert-lits-l } (\text{get-clauses-l } S) (\text{get-unit-clauses-l } S) \rangle$ 
  using  $\text{convert-lits-l-decomp-ex}[OF\ decomp, \text{ of } \langle \text{get-trail } S' \rangle \langle \text{get-clauses-l } S \rangle \langle \text{get-unit-clauses-l } S \rangle]$   $S\text{-}S'$ 
  by  $(\text{auto simp: twl-st-l-def})$ 
have  $x2\text{-DECO}$ :  $\langle \{ \# - x2 \# \} = \text{DECO-clause } (\text{get-trail } S') \rangle$ 
  using  $decomp\ \text{count-dec } S\text{-}S'$ 
  by  $(\text{auto simp: twl-st-l filter-mset-empty-conv count-decided-0-iff dest!: get-all-ann-decomposition-exists-prepend})$ 
have  $M1\text{-drop}$ :  $\langle \text{drop } (\text{length } (\text{get-trail-l } S) - \text{length } M1) (\text{get-trail-l } S) = M1 \rangle$ 
  using  $decomp$  by  $auto$ 
have  $\langle \text{propagate-unit-and-add-l } x2\ x1, ?y1 \rangle \in \{ (S, S''). (S, S'') \in \text{twl-st-l None} \wedge \text{twl-list-invs } S \wedge \text{clauses-to-update-l } S = \{ \# \} \}$ 
  using  $S\text{-}S'\ S\text{-}x1\ tr\text{-}M1\ decomp\ decomp'\ lev\text{-}x2\ add\text{-}inv\ conv\text{-}M1$  unfolding  $x$ 
  apply  $(\text{cases } x1; \text{cases } S')$ 
  by  $(\text{auto simp: twl-st-l-def twl-list-invs-def convert-lit.simps split: option.splits intro: convert-lits-l-extend-mono})$ 
moreover have  $\langle \text{negate-model-and-add-tw } S'\ ?y1 \rangle$ 
  using  $S\text{-}S'\ \text{confl } lev\text{-}x2\ \text{count-dec } tr\text{-}M1\ S\text{-}x1\ decomp\ decomp'\ M1\text{-drop}$ 
  unfolding  $x$ 
  by  $(\text{cases } x1)$ 
   $(\text{auto simp: twl-st-l-def } x2\text{-DECO}\ \text{simp del: convert-lits-l-DECO-clause intro!: negate-model-and-add-tw.bj-unit}[of - -] \text{split: option.splits})$ 
ultimately show  $?thesis$ 
  apply  $-$ 
  by  $(\text{rule } \text{bexI}[of - ?y1])\ \text{fast+}$ 
qed

show  $?thesis$ 
using  $assms$ 
unfolding  $\text{negate-mode-bj-unit-l-def find-decomp-target-def}$ 
apply  $(\text{refine-rcg})$ 
subgoal unfolding  $\text{negate-mode-bj-unit-l-inv-def}$  by  $fast$ 
subgoal
  by  $(\text{subst } \text{RETURN-RES-refine-iff})\ (\text{rule } H; \text{assumption})$ 
done
qed

```

definition $\text{DECO-clause-l} :: \langle ('v, 'a)\ \text{ann-lits} \Rightarrow 'v\ \text{clause-l} \rangle$ **where**
 $\langle \text{DECO-clause-l } M = \text{map } (\text{uminus } o\ \text{lit-of})\ (\text{filter is-decided } M) \rangle$

fun *propagate-nonunit-and-add* :: $\langle 'v \text{ literal} \Rightarrow 'v \text{ literal multiset twl-clause} \Rightarrow 'v \text{ twl-st} \Rightarrow 'v \text{ twl-st} \rangle$
where

$\langle \text{propagate-nonunit-and-add } K \ C \ (M, N, U, D, NE, UE, WS, Q) = \text{do} \{$
 $\quad (\text{Propagated } (-K) \ (\text{clause } C) \ \# \ M, \text{add-mset } C \ N, U, \text{None},$
 $\quad NE, UE, \{\#\}, \{\#K\# \})$
 $\} \rangle$

fun *propagate-nonunit-and-add-l* :: $\langle 'v \text{ literal} \Rightarrow 'v \text{ clause-l} \Rightarrow \text{nat} \Rightarrow 'v \text{ twl-st-l} \Rightarrow 'v \text{ twl-st-l} \rangle$ **where**

$\langle \text{propagate-nonunit-and-add-l } K \ C \ i \ (M, N, D, NE, UE, WS, Q) = \text{do} \{$
 $\quad (\text{Propagated } (-K) \ i \ \# \ M, \text{fmupd } i \ (C, \text{True}) \ N, \text{None},$
 $\quad NE, UE, \{\#\}, \{\#K\# \})$
 $\} \rangle$

definition *negate-mode-bj-nonunit-l-inv* **where**

$\langle \text{negate-mode-bj-nonunit-l-inv } S \longleftrightarrow$
 $\quad (\exists S'' \ b. (S, S'') \in \text{twl-st-l } b \wedge \text{twl-list-invs } S \wedge \text{count-decided } (\text{get-trail-l } S) > 1 \wedge$
 $\quad \text{twl-struct-invs } S'' \wedge \text{twl-stgy-invs } S'' \wedge \text{get-conflict-l } S = \text{None}) \rangle$

definition *negate-mode-bj-nonunit-l* :: $\langle 'v \text{ twl-st-l} \Rightarrow 'v \text{ twl-st-l nres} \rangle$ **where**

$\langle \text{negate-mode-bj-nonunit-l} = (\lambda S. \text{do} \{$
 $\quad \text{ASSERT}(\text{negate-mode-bj-nonunit-l-inv } S);$
 $\quad \text{let } C = \text{DECO-clause-l } (\text{get-trail-l } S);$
 $\quad (S, K) \leftarrow \text{find-decomp-target } (\text{count-decided } (\text{get-trail-l } S)) \ S;$
 $\quad i \leftarrow \text{get-fresh-index } (\text{get-clauses-l } S);$
 $\quad \text{RETURN } (\text{propagate-nonunit-and-add-l } K \ C \ i \ S)$
 $\} \rangle$

lemma *DECO-clause-l-DECO-clause[simp]*:

$\langle \text{mset } (\text{DECO-clause-l } M1) = \text{DECO-clause } M1 \rangle$
by (*induction* *M1*) (*auto simp: DECO-clause-l-def DECO-clause-def convert-lits-l-def*)

lemma *TWL-DECO-clause-alt-def*:

$\langle \text{TWL-DECO-clause } M1 =$
 $\quad \text{TWL-Clause } (\text{mset } (\text{watched-l } (\text{DECO-clause-l } M1)))$
 $\quad (\text{mset } (\text{unwatched-l } (\text{DECO-clause-l } M1))) \rangle$
unfolding *TWL-DECO-clause-def convert-lits-l-def*
by (*auto simp: TWL-DECO-clause-def convert-lits-l-def filter-map take-map drop-map*
 DECO-clause-l-def)

lemma *length-DECO-clause-l[simp]*:

$\langle \text{length } (\text{DECO-clause-l } M) = \text{count-decided } M \rangle$
unfolding *DECO-clause-l-def count-decided-def* **by** *auto*

lemma *negate-mode-bj-nonunit-l*:

fixes *S* :: $\langle 'v \text{ twl-st-l} \rangle$ **and** *S'* :: $\langle 'v \text{ twl-st} \rangle$
assumes
 $\text{count-dec: } \langle \text{count-decided } (\text{get-trail-l } S) > 1 \rangle$ **and**
 $\text{SS': } \langle (S, S') \in \text{twl-st-l } b \rangle$ **and**
 $\text{struct-invs: } \langle \text{twl-struct-invs } S' \rangle$ **and**
 $\text{add-inv: } \langle \text{twl-list-invs } S \rangle$ **and**
 $\text{stgy-inv: } \langle \text{twl-stgy-invs } S' \rangle$ **and**
 $\text{confl: } \langle \text{get-conflict-l } S = \text{None} \rangle$

shows

$\langle \text{negate-mode-bj-nonunit-l } S \leq \Downarrow \{ (S, S''). (S, S'') \in \text{twl-st-l } \text{None} \wedge \text{twl-list-invs } S \wedge$
 $\quad \text{clauses-to-update-l } S = \{\#\} \}$
 $\quad (\text{SPEC } (\text{negate-model-and-add-tw } S')) \rangle$

proof –

have H : $\langle \text{RETURN } (\text{propagate-nonunit-and-add-l } x2 \text{ (DECO-clause-l (get-trail-l } S)) \text{ } i \text{ } x1) \leq \Downarrow \{(S, S''). (S, S'') \in \text{twl-st-l None} \wedge \text{twl-list-invs } S \wedge \text{clauses-to-update-l } S = \{\#\}\} \text{ (SPEC (negate-model-and-add-tw-l } S')) \rangle$

if

x - S : $\langle x \in \{(T, K). \exists M2 \text{ } M1. \text{equality-except-trail } S \text{ } T \wedge \text{get-trail-l } T = M1 \wedge (\text{Decided } K \# M1, M2) \in \text{set (get-all-ann-decomposition (get-trail-l } S)) \wedge \text{get-level (get-trail-l } S) \text{ } K = \text{count-decided (get-trail-l } S)\} \rangle$ **and**

x : $\langle x = (x1, x2) \rangle$ **and**

i : $\langle i \in \{i. 0 < i \wedge i \notin \# \text{ dom-}m \text{ (get-clauses-l } x1)\} \rangle$

for $x :: \langle 'v \text{ twl-st-l} \times 'v \text{ literal} \rangle$ **and**

$x1 :: \langle 'v \text{ twl-st-l} \rangle$ **and** $x2 :: \langle 'v \text{ literal} \rangle$ **and** $i :: \langle \text{nat} \rangle$

proof –

obtain $M \text{ } N \text{ } U \text{ } D \text{ } NE \text{ } UE \text{ } Q$ **where**

$x1$: $\langle x1 = (M, N, U, D, NE, UE, Q) \rangle$

by $(\text{cases } x1)$

obtain $M1 \text{ } M2$ **where**

S - $x1$: $\langle \text{equality-except-trail } S \text{ } x1 \rangle$ **and**

tr - $M1$: $\langle \text{get-trail-l } x1 = M1 \rangle$ **and**

$decomp$: $\langle (\text{Decided } x2 \# M1, M2) \in \text{set (get-all-ann-decomposition (get-trail-l } S)) \rangle$ **and**

lev - K : $\langle \text{get-level (get-trail-l } S) \text{ } x2 = \text{count-decided (get-trail-l } S) \rangle$

using x - S **unfolding** x **by** blast

let $?y0 = \langle (\lambda(M, Oth). (\text{drop (length } M - \text{length (get-trail-l } x1)) \text{ (get-trail } S'), Oth)) \text{ } S' \rangle$

let $?y1 = \langle \text{propagate-nonunit-and-add } x2 \text{ (TWL-DECO-clause (get-trail } S')) \text{ } ?y0 \rangle$

obtain $M3$ **where**

$M3$: $\langle \text{get-trail-l } S = M3 @ M2 @ \text{Decided } x2 \# M1 \rangle$

using $decomp$ **by** blast

have confl' : $\langle \text{get-conflict } S' = \text{None} \rangle$ **and**

trail-S' : $\langle (\text{get-trail-l } S, \text{get-trail } S') \in \text{convert-lits-l (get-clauses-l } S) \text{ (get-unit-clauses-l } S) \rangle$

using $\text{confl } SS'$ **by** $(\text{auto simp: twl-st-l-def})$

have $\langle \text{no-dup (trail (state}_W\text{-of } S')) \rangle$

using struct-invs **unfolding** $\text{twl-struct-invs-def cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv-def cdcl}_W\text{-restart-mset.cdcl}_W\text{-M-level-inv-def}$

by fast

then have $\langle \text{no-dup (get-trail-l } S) \rangle$

using SS' **by** $(\text{auto simp: twl-st twl-st-l})$

then have $[simp]$: $\langle \text{count-decided } M3 = 0 \rangle \langle \text{count-decided } M2 = 0 \rangle$

$\langle \text{filter is-decided } M3 = [] \rangle$

$\langle \text{filter is-decided } M2 = [] \rangle$

using lev-K

by $(\text{auto simp: } M3 \text{ count-decided-0-iff})$

obtain $M2'$ **where**

$decomp'$: $\langle (\text{Decided } x2 \# \text{drop (length (get-trail } S') - \text{length } M1) \text{ (get-trail } S'), M2') \in \text{set (get-all-ann-decomposition (get-trail } S')) \rangle$ **and**

conv : $\langle (\text{get-trail-l } S, \text{get-trail } S') \in \text{convert-lits-l (get-clauses-l } S) \text{ (get-unit-clauses-l } S) \rangle$ **and**

conv-M1 : $\langle (M1, \text{drop (length (get-trail } S') - \text{length } M1) \text{ (get-trail } S')) \in \text{convert-lits-l (get-clauses-l } S) \text{ (get-unit-clauses-l } S) \rangle$

using $\text{convert-lits-l-decomp-ex}[OF \text{ } decomp, \text{ of } \langle \text{get-trail } S' \rangle \langle \text{get-clauses-l } S \rangle \langle \text{get-unit-clauses-l } S \rangle]$ SS'

by $(\text{auto simp: twl-st-l-def})$

```

have M1-drop: ⟨drop (length (get-trail-l S) - length M1) (get-trail-l S) = M1⟩
  using decomp by auto
moreover have ⟨- x2 ∈ set (watched-l (DECO-clause-l (get-trail-l S)))⟩
  using S-x1 tr-M1 SS' i decomp add-inv lev-K M3
  by (auto simp: DECO-clause-l-def)
moreover have ⟨DECO-clause-l (get-trail-l S) ! 0 = -x2⟩
  by (auto simp: M3 DECO-clause-l-def)
moreover have ⟨Propagated L i ∉ set M1⟩ for L
  using add-inv i S-x1 M3 unfolding twl-list-invs-def
  by (cases S; cases x1) auto
ultimately have ⟨(propagate-nonunit-and-add-l x2 (DECO-clause-l (get-trail-l S)) i x1, ?y1) ∈
  {(S, S''). (S, S'') ∈ twl-st-l None ∧ twl-list-invs S ∧ clauses-to-update-l S = {#}}⟩
  using S-x1 tr-M1 SS' i add-inv decomp conv-M1 M1-drop
  by (cases S; cases S')
  (auto simp add: x1 twl-st-l-def twl-list-invs-def init-clss-l-mapsto-upd-notin
    TWL-DECO-clause-alt-def[symmetric] learned-clss-l-mapsto-upd-notin-irrelev
    convert-lit.simps
    intro!: convert-lits-l-extend-mono[of - - N ⟨D + NE⟩])
moreover have ⟨?y1 ∈ Collect (negate-model-and-add-tw1 S')⟩
  using S-x1 tr-M1 i add-inv decomp confl confl' count-dec lev-K decomp' S-x1 SS'
  by (cases S; cases S')
  (auto simp: x1 twl-st-l-def
    intro!: negate-model-and-add-tw1.bj-nonunit[of - - M2'])
ultimately have ⟨∃ y ∈ Collect (negate-model-and-add-tw1 S').
  (propagate-nonunit-and-add-l x2 (DECO-clause-l (get-trail-l S)) i x1, y)
  ∈ {(S, S''). (S, S'') ∈ twl-st-l None ∧ twl-list-invs S ∧
  clauses-to-update-l S = {#}}⟩
  apply -
  apply (rule bexI[of - ?y1])
  apply fast+
  done
then show ?thesis
  unfolding x1
  apply (subst RETURN-RES-refine-iff)
  by fast
qed
have ⟨negate-mode-bj-nonunit-l-inv S⟩
  using assms unfolding negate-mode-bj-nonunit-l-inv-def by blast
then show ?thesis
  unfolding negate-mode-bj-nonunit-l-def find-decomp-target-def get-fresh-index-def
  apply refine-vcg
  apply (rule H; assumption)
  done
qed

```

```

fun restart-nonunit-and-add :: ⟨'v literal multiset twl-clause ⇒ 'v twl-st ⇒ 'v twl-st⟩ where
  ⟨restart-nonunit-and-add C (M, N, U, D, NE, UE, WS, Q) = do {
    (M, add-mset C N, U, None, NE, UE, {#}, {#})
  }⟩

```

```

fun restart-nonunit-and-add-l :: ⟨'v clause-l ⇒ nat ⇒ 'v twl-st-l ⇒ 'v twl-st-l⟩ where
  ⟨restart-nonunit-and-add-l C i (M, N, D, NE, UE, WS, Q) = do {
    (M, fmupd i (C, True) N, None, NE, UE, {#}, {#})
  }⟩

```

definition *negate-mode-restart-nonunit-l-inv* :: $\langle 'v \text{ twl-st-l} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{negate-mode-restart-nonunit-l-inv } S \longleftrightarrow$
 $(\exists S' b. (S, S') \in \text{twl-st-l } b \wedge \text{twl-struct-invs } S' \wedge \text{twl-list-invs } S \wedge \text{twl-stgy-invs } S' \wedge$
 $\text{count-decided } (\text{get-trail-l } S) > 1 \wedge \text{get-conflict-l } S = \text{None}) \rangle$

definition *negate-mode-restart-nonunit-l* :: $\langle 'v \text{ twl-st-l} \Rightarrow 'v \text{ twl-st-l nres} \rangle$ **where**
 $\langle \text{negate-mode-restart-nonunit-l} = (\lambda S. \text{do } \{$
 $\text{ASSERT}(\text{negate-mode-restart-nonunit-l-inv } S);$
 $\text{let } C = \text{DECO-clause-l } (\text{get-trail-l } S);$
 $i \leftarrow \text{SPEC}(\lambda i. i < \text{count-decided } (\text{get-trail-l } S));$
 $(S, K) \leftarrow \text{find-decomp-target } i \text{ } S;$
 $i \leftarrow \text{get-fresh-index } (\text{get-clauses-l } S);$
 $\text{RETURN } (\text{restart-nonunit-and-add-l } C \text{ } i \text{ } S)$
 $\}) \rangle$

lemma *negate-mode-restart-nonunit-l*:

fixes $S :: \langle 'v \text{ twl-st-l} \rangle$ **and** $S' :: \langle 'v \text{ twl-st-l} \rangle$

assumes

count-dec: $\langle \text{count-decided } (\text{get-trail-l } S) > 1 \rangle$ **and**
SS': $\langle (S, S') \in \text{twl-st-l } b \rangle$ **and**
struct-invs: $\langle \text{twl-struct-invs } S' \rangle$ **and**
add-inv: $\langle \text{twl-list-invs } S \rangle$ **and**
stgy-inv: $\langle \text{twl-stgy-invs } S' \rangle$ **and**
confl: $\langle \text{get-conflict-l } S = \text{None} \rangle$

shows

$\langle \text{negate-mode-restart-nonunit-l } S \leq \Downarrow \{ (S, S''). (S, S'') \in \text{twl-st-l } \text{None} \wedge \text{twl-list-invs } S \wedge$
 $\text{clauses-to-update-l } S = \{ \# \} \}$
 $(\text{SPEC } (\text{negate-model-and-add-tw-l } S')) \rangle$

proof –

have H : $\langle \text{RETURN } (\text{restart-nonunit-and-add-l } (\text{DECO-clause-l } (\text{get-trail-l } S)) \text{ } i \text{ } x1)$
 $\leq \Downarrow \{ (S, S''). (S, S'') \in \text{twl-st-l } \text{None} \wedge \text{twl-list-invs } S \wedge$
 $\text{clauses-to-update-l } S = \{ \# \} \}$
 $(\text{SPEC } (\text{negate-model-and-add-tw-l } S')) \rangle$

if

j : $\langle j \in \{ i. i < \text{count-decided } (\text{get-trail-l } S) \} \rangle$ **and**

x : $\langle x \in \{ (T, K). \}$

$\exists M2 \text{ } M1.$

$\text{equality-except-trail } S \text{ } T \wedge$

$\text{get-trail-l } T = M1 \wedge$

$(\text{Decided } K \# M1, M2) \in \text{set } (\text{get-all-ann-decomposition } (\text{get-trail-l } S)) \wedge$

$\text{get-level } (\text{get-trail-l } S) \text{ } K = j \rangle$ **and**

x : $\langle x = (x1, x2) \rangle$ **and**

i : $\langle i \in \{ i. 0 < i \wedge i \notin \# \text{ dom-}m \text{ } (\text{get-clauses-l } x1) \} \rangle$

for $x :: \langle 'v \text{ twl-st-l} \times 'v \text{ literal} \rangle$ **and**

$x1 :: \langle 'v \text{ twl-st-l} \rangle$ **and** $x2 :: \langle 'v \text{ literal} \rangle$ **and** $i \text{ } j :: \langle \text{nat} \rangle$

proof –

obtain $M \text{ } N \text{ } U \text{ } D \text{ } NE \text{ } UE \text{ } Q$ **where**

$x1$: $\langle x1 = (M, N, U, D, NE, UE, Q) \rangle$

by $(\text{cases } x1)$

obtain $M1 \text{ } M2$ **where**

S - $x1$: $\langle \text{equality-except-trail } S \text{ } x1 \rangle$ **and**

tr - $M1$: $\langle \text{get-trail-l } x1 = M1 \rangle$ **and**

$decomp$: $\langle (\text{Decided } x2 \# M1, M2) \in \text{set } (\text{get-all-ann-decomposition } (\text{get-trail-l } S)) \rangle$ **and**

lev - K : $\langle \text{get-level } (\text{get-trail-l } S) \text{ } x2 = j \rangle$

```

using  $x$ - $S$  unfolding  $x$  by blast
let  $?y0 = \langle \lambda(M, Oth). (drop (length (get-trail S') - length M1) (get-trail S'), Oth)) S' \rangle$ 
let  $?y1 = \langle restart-nonunit-and-add (TWL-DECO-clause (get-trail S')) ?y0 \rangle$ 

obtain  $M3$  where
   $M3: \langle get-trail-l S = M3 @ M2 @ Decided x2 \# M1 \rangle$ 
  using decomp by blast
have  $\langle M = M1 \rangle$ 
  using  $S$ - $x1$   $SS'$  decomp  $tr$ - $M1$  unfolding  $x1$ 
  by (cases  $S$ ; cases  $S'$ ) auto
have confl':  $\langle get-conflict S' = None \rangle$  and
   $trail$ - $S'$ :  $\langle (get-trail-l S, get-trail S') \in convert-lits-l (get-clauses-l S) (get-unit-clauses-l S) \rangle$ 
  using confl  $SS'$  by (auto simp: twl-st-l)
have  $\langle no-dup (trail (state_W-of S')) \rangle$ 
  using struct-invs unfolding twl-struct-invs-def cdcl_W-restart-mset.cdcl_W-all-struct-inv-def
  cdcl_W-restart-mset.cdcl_W-M-level-inv-def
  by fast
then have  $\langle no-dup (get-trail-l S) \rangle$ 
  using  $SS'$  by (auto simp: twl-st twl-st-l)
obtain  $M2'$  where
  decomp':  $\langle (Decided x2 \# drop (length (get-trail S') - length M1) (get-trail S'), M2') \in set (get-all-ann-decomposition (get-trail S')) \rangle$  and
  conv:  $\langle (get-trail-l S, get-trail S') \in convert-lits-l (get-clauses-l S) (get-unit-clauses-l S) \rangle$  and
  conv- $M1$ :  $\langle (M1, drop (length (get-trail S') - length M1) (get-trail S')) \in convert-lits-l (get-clauses-l S) (get-unit-clauses-l S) \rangle$ 
  using convert-lits-l-decomp-ex[OF decomp, of  $\langle get-trail S' \rangle \langle get-clauses-l S \rangle \langle get-unit-clauses-l S \rangle$ ]  $SS'$ 
  by (auto simp: twl-st-l-def)
have  $M1$ -drop:  $\langle drop (length (get-trail-l S) - length M1) (get-trail-l S) = M1 \rangle$ 
  using decomp by auto

moreover have  $\langle Propagated L i \notin set M1 \rangle$  for  $L$ 
  using add-inv  $i$   $S$ - $x1$   $M3$  unfolding twl-list-invs-def
  by (cases  $S$ ; cases  $x1$ ) auto
ultimately have  $\langle (restart-nonunit-and-add-l (DECO-clause-l (get-trail-l S)) i x1, ?y1) \in \{(S, S''). (S, S'') \in twl-st-l None \wedge twl-list-invs S \wedge clauses-to-update-l S = \{\#\}\} \rangle$ 
  using  $S$ - $x1$   $tr$ - $M1$   $SS'$   $i$  add-inv decomp conv- $M1$  decomp'
  by (cases  $S$ ; cases  $S'$ )
  (auto simp:  $x1$  twl-st-l-def twl-list-invs-def init-clss-l-mapsto-upd-notin
  TWL-DECO-clause-alt-def[symmetric] learned-clss-l-mapsto-upd-notin-irrelev
  dest: get-all-ann-decomposition-exists-prepend
  intro!: convert-lits-l-extend-mono[of - -  $N \langle D+NE \rangle$ ])
moreover {
  have  $\langle get-level (get-trail-l S) x2 < count-decided (get-trail-l S) \rangle$ 
  using lev-K  $j$  by auto
  then have  $\langle ?y1 \in Collect (negate-model-and-add-tw1 S') \rangle$ 
  using  $S$ - $x1$   $tr$ - $M1$   $i$  add-inv decomp' confl confl' count-dec lev-K  $SS'$ 
  by (cases  $S$ ; cases  $S'$ )
  (auto simp:  $x1$  twl-st-l-def
  intro!: negate-model-and-add-tw1.restart-nonunit[of  $x2 - \langle M2' \rangle$ ])
}
ultimately have  $\langle \exists y \in Collect (negate-model-and-add-tw1 S'). (restart-nonunit-and-add-l (DECO-clause-l (get-trail-l S)) i x1, y) \in \{(S, S''). (S, S'') \in twl-st-l None \wedge twl-list-invs S \wedge$ 

```

```

    clauses-to-update-l S = {#} }
  apply -
  apply (rule bexI[of - ?y1])
  apply fast+
  done
then show ?thesis
  unfolding x1
  apply (subst RETURN-RES-refine-iff)
  by fast
qed
show ?thesis
  unfolding negate-mode-restart-nonunit-l-def find-decomp-target-def get-fresh-index-def
  apply refine-vcg
  subgoal
    using assms unfolding negate-mode-restart-nonunit-l-inv-def by fast
  subgoal
    supply [[unify-trace-failure]]
    apply (rule H; assumption)
    done
  done
qed

```

definition *negate-mode-l-inv* **where**

```

⟨negate-mode-l-inv S ⟷
  (∃ S' b. (S, S') ∈ twl-st-l b ∧ twl-struct-invs S' ∧ twl-list-invs S ∧ twl-stgy-invs S' ∧
    get-conflict-l S = None ∧ count-decided (get-trail-l S) ≠ 0)⟩

```

definition *negate-mode-l* :: ⟨'v twl-st-l ⇒ 'v twl-st-l nres⟩ **where**

```

⟨negate-mode-l S = do {
  ASSERT(negate-mode-l-inv S);
  if count-decided (get-trail-l S) = 1
  then negate-mode-bj-unit-l S
  else do {
    b ← SPEC(λ-. True);
    if b then negate-mode-bj-nonunit-l S else negate-mode-restart-nonunit-l S
  }
}⟩

```

lemma *negate-mode-l*:

fixes S :: ⟨'v twl-st-l⟩ **and** S' :: ⟨'v twl-st-l⟩

assumes

SS': ⟨(S, S') ∈ twl-st-l b⟩ **and**
 struct-invs: ⟨twl-struct-invs S'⟩ **and**
 add-inv: ⟨twl-list-invs S⟩ **and**
 stgy-inv: ⟨twl-stgy-invs S'⟩ **and**
 confl: ⟨get-conflict-l S = None⟩ **and**
 count-decided ⟨get-trail-l S⟩ ≠ 0

shows

⟨negate-mode-l S ≤ ↓{(S, S''). (S, S'') ∈ twl-st-l None ∧ twl-list-invs S ∧
 clauses-to-update-l S = {#}}
 (SPEC (negate-model-and-add-twl S'))⟩

unfolding *negate-mode-l-def*

apply (refine-vcg negate-mode-restart-nonunit-l[OF - SS'] negate-mode-bj-unit-l[OF - SS']
 negate-mode-bj-nonunit-l[OF - SS'] lhs-step-If)

subgoal using *assms* **unfolding** *negate-mode-l-inv-def* **by** *fast*

subgoal using *assms* **by** *fast*

```

subgoal using assms by fast
subgoal using assms by fast
subgoal using assms by fast
subgoal using assms by simp
subgoal using assms by fast
subgoal using assms by fast
subgoal using assms by fast
subgoal using assms by fast
subgoal using assms by simp
subgoal using assms by fast
subgoal using assms by fast
subgoal using assms by fast
subgoal using assms by fast
done

```

context

fixes $P :: \langle 'v \text{ literal set} \Rightarrow \text{bool} \rangle$

begin

definition $\text{cdcl-twl-enum-inv-l} :: \langle 'v \text{ twl-st-l} \Rightarrow \text{bool} \rangle$ **where**

$\langle \text{cdcl-twl-enum-inv-l } S \longleftrightarrow$
 $(\exists S'. (S, S') \in \text{twl-st-l None} \wedge \text{cdcl-twl-enum-inv } S') \wedge$
 $\text{twl-list-invs } S \rangle$

definition $\text{cdcl-twl-enum-l} :: \langle 'v \text{ twl-st-l} \Rightarrow \text{bool nres} \rangle$ **where**

$\langle \text{cdcl-twl-enum-l } S = \text{do} \{$
 $S \leftarrow \text{cdcl-twl-stgy-prog-l } S;$
 $S \leftarrow \text{WHILE}_T^{\text{cdcl-twl-enum-inv-l}}$
 $(\lambda S. \text{get-conflict-l } S = \text{None} \wedge \text{count-decided}(\text{get-trail-l } S) > 0 \wedge$
 $\neg P (\text{lits-of-l } (\text{get-trail-l } S)))$
 $(\lambda S. \text{do} \{$
 $S \leftarrow \text{negate-mode-l } S;$
 $\text{cdcl-twl-stgy-prog-l } S$
 $\})$
 $S;$
 $\text{if } \text{get-conflict-l } S = \text{None}$
 $\text{then RETURN } (\text{if } \text{count-decided}(\text{get-trail-l } S) = 0 \text{ then } P (\text{lits-of-l } (\text{get-trail-l } S)) \text{ else True})$
 $\text{else RETURN } (\text{False})$
 $\} \rangle$

lemma $\text{negate-model-and-add-twl-resultD}:$

$\langle \text{negate-model-and-add-twl } S \ T \Longrightarrow$
 $\text{clauses-to-update } T = \{\#\} \wedge \text{get-conflict } T = \text{None} \rangle$
by $(\text{auto simp: negate-model-and-add-twl.simps})$

lemma $\text{cdcl-twl-enum-l}:$

fixes $S :: \langle 'v \text{ twl-st-l} \rangle$ **and** $S' :: \langle 'v \text{ twl-st-l} \rangle$
assumes
 $SS': \langle (S, S') \in \text{twl-st-l None} \rangle$ **and**
 $\text{struct-invs: } \langle \text{twl-struct-invs } S' \rangle$ **and**
 $\text{add-inv: } \langle \text{twl-list-invs } S \rangle$ **and**
 $\text{stgy-inv: } \langle \text{twl-stgy-invs } S' \rangle$ **and**
 $\text{confl: } \langle \text{get-conflict-l } S = \text{None} \rangle$ **and**
 $\langle \text{count-decided } (\text{get-trail-l } S) \neq 0 \rangle$ **and**
 $\langle \text{clauses-to-update-l } S = \{\#\} \rangle$

```

shows
  ⟨cdcl-tw-l-enum-l S ≤ ↓ bool-rel
    (cdcl-tw-l-enum P S')⟩
unfolding cdcl-tw-l-enum-l-def cdcl-tw-l-enum-def
apply (refine-vcg cdcl-tw-l-stgy-prog-l-spec-final' negate-mode-l)
subgoal
  using assms unfolding cdcl-tw-l-stgy-prog-l-pre-def
  by fast
apply assumption
subgoal for S S' U U'
  using assms unfolding cdcl-tw-l-enum-inv-l-def
  apply -
  apply (intro conjI)
  apply (rule exI[of - U'])
  by auto
subgoal by (auto simp: tw-l-st-l)
apply auto[]
subgoal unfolding cdcl-tw-l-enum-inv-def by auto
subgoal by fast
subgoal by (auto simp: tw-l-st-l cdcl-tw-l-enum-inv-def)
subgoal by (auto simp: tw-l-st-l)
subgoal by (auto simp: tw-l-st-l)
subgoal for S S' T T' U U'
  by (rule cdcl-tw-l-stgy-prog-l-spec-final'[THEN order.trans])
  (auto simp: tw-l-st tw-l-st-l cdcl-tw-l-stgy-prog-l-pre-def cdcl-tw-l-enum-inv-def
    intro: negate-model-and-add-tw-l-tw-l-struct-invs
      negate-model-and-add-tw-l-tw-l-stgy-invs conc-fun-R-mono
      dest: negate-model-and-add-tw-l-resultD)
subgoal by (auto simp: tw-l-st-l)
subgoal by (auto simp: tw-l-st-l)
done

end

end
theory Watched-Literals-Watch-List-Enumeration
imports Watched-Literals-List-Enumeration Watched-Literals.Watched-Literals-Watch-List
begin

definition find-decomp-target-wl :: ⟨nat ⇒ 'v tw-l-st-wl ⇒ ('v tw-l-st-wl × 'v literal) nres⟩ where
  ⟨find-decomp-target-wl = (λi S.
    SPEC(λ(T, K). ∃ M2 M1. equality-except-trail-wl S T ∧ get-trail-wl T = M1 ∧
      (Decided K # M1, M2) ∈ set (get-all-ann-decomposition (get-trail-wl S)) ∧
      get-level (get-trail-wl S) K = i))⟩

fun propagate-unit-and-add-wl :: ⟨'v literal ⇒ 'v tw-l-st-wl ⇒ 'v tw-l-st-wl⟩ where
  ⟨propagate-unit-and-add-wl K (M, N, D, NE, UE, Q, W) =
    (Propagated (-K) 0 # M, N, None, add-mset {#-K#} NE, UE, {#K#}, W)⟩

definition negate-mode-bj-unit-wl :: ⟨'v tw-l-st-wl ⇒ 'v tw-l-st-wl nres⟩ where
  ⟨negate-mode-bj-unit-wl = (λS. do {
    (S, K) ← find-decomp-target-wl 1 S;
    ASSERT(K ∈ # all-lits-of-mm (clause '# tw-l-clause-of '# ran-mf (get-clauses-wl S) +
      get-unit-clauses-wl S));
    RETURN (propagate-unit-and-add-wl K S)
  })⟩

```

abbreviation *find-decomp-target-wl-ref* **where**

$\langle \text{find-decomp-target-wl-ref } S \equiv$
 $\{((T, K), (T', K')). (T, T') \in \{(T, T'). (T, T') \in \text{state-wl-l None} \wedge \text{correct-watching } T\} \wedge$
 $(K, K') \in \text{Id} \wedge$
 $K \in \# \text{ all-lits-of-mm (clause '\# twl-clause-of '\# ran-mf (get-clauses-wl } T) +$
 $\text{get-unit-clauses-wl } T) \wedge$
 $K \in \# \text{ all-lits-of-mm (clause '\# twl-clause-of '\# ran-mf (get-clauses-wl } T) +$
 $\text{get-unit-init-clss-wl } T) \wedge \text{equality-except-trail-wl } S \text{ } T \wedge$
 $\text{atms-of (DECO-clause (get-trail-wl } S)) \subseteq \text{atms-of-mm (clause '\# twl-clause-of '\# ran-mf}$
 $(\text{get-clauses-wl } T) +$
 $\text{get-unit-init-clss-wl } T) \wedge \text{distinct-mset (DECO-clause (get-trail-wl } S)) \wedge$
 $\text{correct-watching } T\}$

lemma *DECO-clause-nil[simp]*: $\langle \text{DECO-clause } [] = \{\#\} \rangle$

by (*auto simp: DECO-clause-def*)

lemma *in-DECO-clauseD*: $\langle x \in \# \text{ DECO-clause } M \implies -x \in \text{lits-of-l } M \rangle$

by (*auto simp: DECO-clause-def lits-of-def*)

lemma *in-atms-of-DECO-clauseD*: $\langle x \in \text{atms-of (DECO-clause } M) \implies x \in \text{atm-of ' (lits-of-l } M) \rangle$

by (*auto simp: DECO-clause-def lits-of-def atms-of-def*)

lemma *no-dup-distinct-mset-DECO-clause*:

assumes $\langle \text{no-dup } M \rangle$

shows $\langle \text{distinct-mset (DECO-clause } M) \rangle$

proof –

have $\langle \text{distinct (map lit-of (filter is-decided } M)) \rangle$

using *no-dup-map-lit-of[OF assms] distinct-map-filter* **by** *blast*

moreover have $\langle ?thesis \longleftrightarrow \text{distinct (map lit-of (filter is-decided } M)) \rangle$

unfolding *DECO-clause-def image-mset.compositionality[symmetric]*

apply (*subst distinct-image-mset-inj*)

subgoal by (*auto simp: inj-on-def*)

subgoal by (*auto simp flip: mset-filter*

distinct-mset-mset-distinct simp del: mset-filter)

done

ultimately show *?thesis* **by** *blast*

qed

lemma *find-decomp-target-wl-find-decomp-target-l*:

assumes

SS': $\langle (S, S') \in \{(S, S''). (S, S'') \in \text{state-wl-l None} \wedge \text{correct-watching } S\} \rangle$ **and**

inv: $\langle \exists S'' b. (S', S'') \in \text{twl-st-l } b \wedge \text{twl-struct-invs } S'' \rangle$ **and**

[simp]: $\langle a = a' \rangle$

shows $\langle \text{find-decomp-target-wl } a \text{ } S \leq$

$\Downarrow (\text{find-decomp-target-wl-ref } S) (\text{find-decomp-target } a' \text{ } S') \rangle$

(is $\langle - \leq \Downarrow ?\text{negate } - \rangle$)

proof –

let $?y0 = \langle \lambda S S'. (\lambda (M, Oth). (\text{get-trail-wl } S, Oth)) S' \rangle$

have *K*: $\langle \bigwedge K. K \in \text{lits-of-l (get-trail-wl } S) \implies$

$K \in \# \text{ all-lits-of-mm (clause '\# twl-clause-of '\# ran-mf (get-clauses-wl } S) +$

$\text{get-unit-init-clss-wl } S) \rangle$ **(is** $\langle \bigwedge K. ?HK K \implies ?K K \rangle$) **and**

DECO:

$\langle \text{atms-of (DECO-clause (get-trail-wl } S)) \subseteq \text{atms-of-mm (clause '\# twl-clause-of '\# ran-mf}$

$(\text{get-clauses-wl } S) +$

$\text{get-unit-init-clss-wl } S) \rangle$ **(is** *?DECO*) **and**

distinct-DECO:
 $\langle \text{distinct-mset } (\text{DECO-clause } (\text{get-trail-wl } S)) \rangle \text{ (is } ?\text{dist-DECO})$
proof –
obtain $b \ S''$ **where**
 $S'-S'': \langle (S', S'') \in \text{twl-st-l } b \rangle$ **and**
 $\text{struct}: \langle \text{twl-struct-invs } S'' \rangle$
using *inv unfolding negate-mode-bj-unit-l-inv-def* **by** *blast*
then have *no-alien*: $\langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv } (\text{state}_W\text{-of } S'') \rangle$
using *struct unfolding twl-struct-invs-def* **by** *fast*
then have *no-alien*: $\langle \text{cdcl}_W\text{-restart-mset.no-strange-atm } (\text{state}_W\text{-of } S'') \rangle$ **and**
 $M\text{-lev}: \langle \text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-M-level-inv } (\text{state}_W\text{-of } S'') \rangle$
unfolding *cdcl_W-restart-mset.cdcl_W-all-struct-inv-def* **by** *fast+*
moreover have $\langle \text{atms-of-mm } (\text{get-all-init-clss } S'') =$
 $\text{atms-of-mm } (\text{mset } '\# (\text{ran-mf } (\text{get-clauses-wl } S)) + \text{get-unit-init-clss-wl } S) \rangle$
apply *(subst all-clss-lf-ran-m[symmetric])*
using *no-alien*
using $S'-S'' \ SS'$ **unfolding** *cdcl_W-restart-mset.no-strange-atm-def*
by *(cases S; cases S'; cases b)*
 $(\text{auto simp: mset-take-mset-drop-mset' cdcl}_W\text{-restart-mset-state}$
 $\text{in-all-lits-of-mm-ain-atms-of-iff twl-st-l-def state-wl-l-def})$
ultimately show $\langle \bigwedge K. ?HK \ K \implies ?K \ K \rangle$
using $S'-S'' \ SS'$ **unfolding** *cdcl_W-restart-mset.no-strange-atm-def*
by *(auto 5 5 simp: twl-st-l twl-st mset-take-mset-drop-mset'*
 $\text{in-all-lits-of-mm-ain-atms-of-iff get-unit-clauses-wl-alt-def})$
then show *?DECO*
using $S'-S'' \ SS'$ **unfolding** *cdcl_W-restart-mset.no-strange-atm-def*
by *(auto simp: twl-st-l twl-st mset-take-mset-drop-mset'*
 $\text{in-all-lits-of-mm-ain-atms-of-iff get-unit-clauses-wl-alt-def}$
 $\text{dest: in-atms-of-DECO-clauseD})$

show *?dist-DECO*
by *(rule no-dup-distinct-mset-DECO-clause)*
 $(\text{use } M\text{-lev } S'-S'' \ SS' \text{ in } (\text{auto simp: cdcl}_W\text{-restart-mset.cdcl}_W\text{-M-level-inv-def twl-st}))$
qed

show *?thesis*
using SS'
unfolding *find-decomp-target-wl-def find-decomp-target-def* **apply** –
apply *(rule RES-refine)*
apply *(rule-tac x = (⟨?y0 (fst s) S', snd s⟩) in bexI)*
subgoal
using $K \ \text{DECO} \ \text{distinct-DECO}$
by *(cases S; cases S')*
 $(\text{force simp: state-wl-l-def correct-watching.simps clause-to-update-def}$
 $\text{mset-take-mset-drop-mset' all-lits-of-mm-union}$
 $\text{dest! : get-all-ann-decomposition-exists-prepend})+$
subgoal
by *(cases S; cases S')*
 $(\text{auto simp: state-wl-l-def correct-watching.simps clause-to-update-def})$
done
qed

lemma *negate-mode-bj-unit-wl-negate-mode-bj-unit-l*:
fixes $S :: \langle 'v \ \text{twl-st-wl} \rangle$ **and** $S' :: \langle 'v \ \text{twl-st-l} \rangle$
assumes $\langle \text{count-decided } (\text{get-trail-wl } S) = 1 \rangle$ **and**
 $SS': \langle (S, S') \in \{(S, S'). (S, S') \in \text{state-wl-l None} \wedge \text{correct-watching } S\} \rangle$

```

shows
  ⟨negate-mode-bj-unit-wl  $S \leq \Downarrow \{(S, S'). (S, S') \in \text{state-wl-l None} \wedge \text{correct-watching } S\}$ 
    (negate-mode-bj-unit-l  $S'\rangle$ 
    (is  $\langle - \leq \Downarrow ?R - \rangle$ )
proof –
  have 2: ⟨(propagate-unit-and-add-wl  $x2a\ x1a$ , propagate-unit-and-add-l  $x2\ x1$ )
     $\in \{(S, S'). (S, S') \in \text{state-wl-l None} \wedge \text{correct-watching } S\}$ 
    if
      ⟨ $(x, x') \in \text{find-decomp-target-wl-ref } S\rangle$  and
      ⟨ $x' = (x1, x2)\rangle$  and
      ⟨ $x = (x1a, x2a)\rangle$ 
      for  $x2a\ x1a\ x2\ x1$  and  $x :: \langle 'v\ \text{twl-st-wl} \times 'v\ \text{literal} \rangle$  and  $x' :: \langle 'v\ \text{twl-st-l} \times 'v\ \text{literal} \rangle$ 
    proof –
      show ?thesis
      using that
      by (cases  $x1a$ ; cases  $x1$ )
        (auto, auto simp: state-wl-l-def correct-watching.simps clause-to-update-def
          all-lits-of-mm-add-mset
          all-lits-of-m-add-mset all-lits-of-mm-union mset-take-mset-drop-mset'
          dest: in-all-lits-of-mm-uminusD)
    qed

  show ?thesis
  using  $SS'$  unfolding negate-mode-bj-unit-wl-def negate-mode-bj-unit-l-def
  apply (refine-rcg find-decomp-target-wl-find-decomp-target-l 2)
  subgoal unfolding negate-mode-bj-unit-l-inv-def by blast
  subgoal unfolding negate-mode-bj-unit-l-inv-def by blast
  subgoal by blast
  apply assumption+
  done
qed

definition propagate-nonunit-and-add-wl-pre
  :: ⟨'v literal  $\Rightarrow$  'v clause-l  $\Rightarrow$  nat  $\Rightarrow$  'v twl-st-wl  $\Rightarrow$  bool⟩ where
  ⟨propagate-nonunit-and-add-wl-pre  $K\ C\ i\ S \longleftrightarrow$ 
    length  $C \geq 2 \wedge i > 0 \wedge i \notin \# \text{ dom-}m\ (\text{get-clauses-wl } S) \wedge$ 
    atms-of (mset  $C) \subseteq \text{atms-of-mm } (\text{clause } \# \text{ twl-clause-of } \# \text{ ran-mf } (\text{get-clauses-wl } S) +$ 
      get-unit-init-clss-wl  $S)\rangle$ 

fun propagate-nonunit-and-add-wl
  :: ⟨'v literal  $\Rightarrow$  'v clause-l  $\Rightarrow$  nat  $\Rightarrow$  'v twl-st-wl  $\Rightarrow$  'v twl-st-wl nres⟩
where
  ⟨propagate-nonunit-and-add-wl  $K\ C\ i\ (M, N, D, NE, UE, Q, W) = \text{do } \{$ 
    ASSERT(propagate-nonunit-and-add-wl-pre  $K\ C\ i\ (M, N, D, NE, UE, Q, W));$ 
    let  $b = (\text{length } C = 2);$ 
    let  $W = W(C!0 := W\ (C!0) @ [(i, C!1, b)]);$ 
    let  $W = W(C!1 := W\ (C!1) @ [(i, C!0, b)]);$ 
    RETURN (Propagated  $(-K)\ i\ \# M, \text{fmupd } i\ (C, \text{True})\ N, \text{None},$ 
      NE, UE,  $\{\#K\# \}, W)$ 
  ⟩

lemma twl-st-l-splitD:
  ⟨ $(\bigwedge M\ N\ D\ NE\ UE\ Q\ W. f\ (M, N, D, NE, UE, Q, W) = P\ M\ N\ D\ NE\ UE\ Q\ W) \implies$ 
     $f\ S = P\ (\text{get-trail-l } S)\ (\text{get-clauses-l } S)\ (\text{get-conflict-l } S)\ (\text{get-unit-init-clauses-l } S)$ 
     $(\text{get-unit-learned-clauses-l } S)\ (\text{clauses-to-update-l } S)\ (\text{literals-to-update-l } S)\rangle$ 
  by (cases  $S$ ) auto

```

lemma *twl-st-wl-splitD*:

$\langle (\bigwedge M N D NE UE Q W. f (M, N, D, NE, UE, Q, W) = P M N D NE UE Q W) \implies$
 $f S = P (get-trail-wl S) (get-clauses-wl S) (get-conflict-wl S) (get-unit-init-clss-wl S)$
 $(get-unit-learned-clss-wl S) (literals-to-update-wl S) (get-watched-wl S) \rangle$
by (cases S) auto

definition *negate-mode-bj-nonunit-wl-inv* **where**

$\langle negate-mode-bj-nonunit-wl-inv S \longleftrightarrow$
 $(\exists S'' b. (S, S'') \in state-wl-l b \wedge negate-mode-bj-nonunit-l-inv S'' \wedge correct-watching S) \rangle$

definition *negate-mode-bj-nonunit-wl* :: $\langle 'v twl-st-wl \implies 'v twl-st-wl nres \rangle$ **where**

$\langle negate-mode-bj-nonunit-wl = (\lambda S. do \{$
 $ASSERT(negate-mode-bj-nonunit-wl-inv S);$
 $let C = DECO-clause-l (get-trail-wl S);$
 $(S, K) \leftarrow find-decomp-target-wl (count-decided (get-trail-wl S)) S;$
 $i \leftarrow get-fresh-index-wl (get-clauses-wl S) (get-unit-clauses-wl S) (get-watched-wl S);$
 $propagate-nonunit-and-add-wl K C i S$
 $\}) \rangle$

lemmas *propagate-nonunit-and-add-wl-def* =

twl-st-wl-splitD[of $\langle propagate-nonunit-and-add-wl - - \rangle$, OF *propagate-nonunit-and-add-wl.simps*]

lemmas *propagate-nonunit-and-add-l-def* =

twl-st-l-splitD[of $\langle propagate-nonunit-and-add-l - - \rangle$, OF *propagate-nonunit-and-add-l.simps*,
rule-format]

lemma *atms-of-subset-in-atms-ofI*:

$\langle atms-of C \subseteq atms-of-ms N \implies L \in \# C \implies atm-of L \in atms-of-ms N \rangle$
by (auto dest!: multi-member-split)

lemma *in-DECO-clause-l-in-DECO-clause-iff*:

$\langle x \in set (DECO-clause-l M) \longleftrightarrow x \in \# (DECO-clause M) \rangle$
by (metis DECO-clause-l-DECO-clause set-mset-mset)

lemma *distinct-DECO-clause-l*:

$\langle no-dup M \implies distinct (DECO-clause-l M) \rangle$
by (auto simp: DECO-clause-l-def distinct-map inj-on-def
dest!: no-dup-map-lit-of)

lemma *propagate-nonunit-and-add-wl-propagate-nonunit-and-add-l*:

assumes

SS': $\langle (S, S') \in state-wl-l None \rangle$ **and**

inv: $\langle negate-mode-bj-nonunit-wl-inv S \rangle$ **and**

TK: $\langle (TK, TK') \in find-decomp-target-wl-ref S \rangle$ **and**

[*simp*]: $\langle TK' = (T, K) \rangle$ **and**

[*simp*]: $\langle TK = (T', K') \rangle$ **and**

ij: $\langle (i, j) \in \{(i, j). i = j \wedge i \notin \# dom-m (get-clauses-wl T') \wedge i > 0 \wedge$

$(\forall L \in \# all-lits-of-mm (mset \# ran-mf (get-clauses-wl T') + get-unit-clauses-wl T') .$
 $i \notin fst \text{ ' set (watched-by T' L) }) \rangle$

shows $\langle propagate-nonunit-and-add-wl K' (DECO-clause-l (get-trail-wl S)) i T'$

$\leq SPEC (\lambda c. (c, propagate-nonunit-and-add-l K$
 $(DECO-clause-l (get-trail-l S')) j T)$

$\in \{(S, S'').$

$(S, S'') \in state-wl-l None \wedge correct-watching S \rangle$

proof –

```

have [simp]:  $\langle i = j \rangle$  and  $j: \langle j \notin \text{dom-}m \text{ (get-clauses-wl } T') \rangle$ 
  using  $ij$  by auto
have [simp]:  $\langle \text{DECO-clause-l (get-trail-l } S') = \text{DECO-clause-l (get-trail-wl } S) \rangle$ 
  using  $SS'$  by auto
obtain  $T \ U \ b \ b'$  where
   $ST: \langle (S, T) \in \text{state-wl-l } b \rangle$  and
   $\text{corr}: \langle \text{correct-watching } S \rangle$  and
   $TU: \langle (T, U) \in \text{twl-st-l } b' \rangle$  and
   $\langle \text{twl-list-invs } T \rangle$  and
   $\text{ge1}: \langle 1 < \text{count-decided (get-trail-l } T) \rangle$  and
   $\text{st}: \langle \text{twl-struct-invs } U \rangle$  and
   $\langle \text{twl-stgy-invs } U \rangle$  and
   $\langle \text{get-conflict-l } T = \text{None} \rangle$ 
  using inv unfolding negate-mode-bj-nonunit-wl-inv-def negate-mode-bj-nonunit-l-inv-def apply –
  by blast
have  $\langle \text{length (DECO-clause-l (get-trail-wl } S)) > 1 \rangle$ 
  using  $ST \text{ ge1}$  by auto
then have 1:  $\langle \text{DECO-clause-l (get-trail-wl } S) =$ 
   $\text{DECO-clause-l (get-trail-wl } S) ! 0 \#$ 
   $\text{DECO-clause-l (get-trail-wl } S) ! \text{Suc } 0 \# \text{drop } 2 \text{ (DECO-clause-l (get-trail-wl } S))} \rangle$ 
  by (cases  $\langle \text{DECO-clause-l (get-trail-wl } S) \rangle$ ; cases  $\langle \text{tl (DECO-clause-l (get-trail-wl } S))} \rangle$ )
  auto
have  $\langle \text{no-dup (trail (state}_W\text{-of } U)) \rangle$ 
  using st unfolding twl-struct-invs-def cdcl}_W\text{-restart-mset.cdcl}_W\text{-all-struct-inv-def}
   $\text{cdcl}_W\text{-restart-mset.cdcl}_W\text{-M-level-inv-def}$ 
  by fast
then have  $\text{neg: False}$  if  $\langle \text{DECO-clause-l (get-trail-wl } S) ! 0 = \text{DECO-clause-l (get-trail-wl } S) ! \text{Suc } 0 \rangle$ 
  using that
  apply ( $\text{subst (asm) nth-eq-iff-index-eq}$ )
  using  $\text{ge1 } ST \ TU$  by ( $\text{auto simp: twl-st twl-st-l twl-st-wl distinct-DECO-clause-l}$ )

show  $?thesis$ 
  using  $TK \ j \ \text{corr} \ \text{ge1 } ST$ 
  apply ( $\text{simp only: propagate-nonunit-and-add-wl-def}$ 
     $\text{propagate-nonunit-and-add-l-def Let-def}$ 
     $\text{assert-bind-spec-conv}$ )
  apply ( $\text{intro conjI}$ )
  subgoal using  $j \ ij \ TK$  unfolding  $\text{propagate-nonunit-and-add-wl-pre-def}$  by auto
  subgoal
    unfolding  $\text{RETURN-def less-eq-nres.simps mem-Collect-eq prod.simps singleton-iff}$ 
    apply ( $\text{subst subset-iff}$ )
    unfolding  $\text{RETURN-def less-eq-nres.simps mem-Collect-eq prod.simps singleton-iff}$ 
    apply ( $\text{intro conjI impI allI}$ )
    subgoal by ( $\text{auto simp: state-wl-l-def}$ )
    subgoal
      apply ( $\text{simp only: }$ )
      apply ( $\text{subst 1}$ )
      apply ( $\text{subst One-nat-def[symmetric]}+$ )
      apply ( $\text{subst fun-upd-other}$ )
      subgoal
        using  $SS' \ \text{length-DECO-clause-l[of (get-trail-wl } S)]$ 
        by (cases  $\langle \text{DECO-clause-l (get-trail-wl } S) \rangle$ ; cases  $\langle \text{tl (DECO-clause-l (get-trail-wl } S))} \rangle$ )
          ( $\text{auto simp: DECO-clause-l-DECO-clause[symmetric] twl-st-l twl-st}$ 
             $\text{simp del: DECO-clause-l-DECO-clause}$ )

```

```

apply (rule correct-watching-learn[THEN iffD2])
apply (rule atms-of-subset-in-atms-ofI[of  $\langle \text{DECO-clause } (\text{get-trail-wl } S) \rangle$ ])
subgoal by (auto simp add: mset-take-mset-drop-mset' get-unit-clauses-wl-alt-def
  DECO-clause-l-DECO-clause[symmetric]
  simp del: DECO-clause-l-DECO-clause)
subgoal by (solves (auto simp add: mset-take-mset-drop-mset'
  DECO-clause-l-DECO-clause[symmetric]
  simp del: DECO-clause-l-DECO-clause))
subgoal apply (use in (auto simp add: mset-take-mset-drop-mset' DECO-clause-l-DECO-clause[symmetric]
  simp del: DECO-clause-l-DECO-clause))
  by (metis (no-types, lifting) 1 UnE add-mset-commute image-eqI mset.simps(2)
    set-mset-mset subsetCE union-single-eq-member)
subgoal — TODO Proof
apply (auto simp: mset-take-mset-drop-mset' in-DECO-clause-l-in-DECO-clause-iff
  dest!: in-set-dropD)
  by (metis UnE atms-of-ms-union atms-of-subset-in-atms-ofI)
subgoal by simp
subgoal using corr ij
  by (cases S; cases T; cases T')
    (auto simp: equality-except-trail-wl.simps state-wl-l-def correct-watching.simps
      clause-to-update-def)
subgoal using corr neq
  by (cases S; cases T; cases T')
    (auto simp: equality-except-trail-wl.simps state-wl-l-def correct-watching.simps
      clause-to-update-def)
subgoal
  by (subst 1) auto
subgoal using corr
  by (cases S; cases T; cases T')
    (auto simp: equality-except-trail-wl.simps state-wl-l-def correct-watching.simps
      clause-to-update-def)
done
done
done
qed

```

lemma watched-by-alt-def:
 $\langle \text{watched-by } T L = \text{get-watched-wl } T L \rangle$
by (cases T) auto

lemma negate-mode-bj-nonunit-wl-negate-mode-bj-nonunit-l:
fixes $S :: \langle 'v \text{ twl-st-wl} \rangle$ **and** $S' :: \langle 'v \text{ twl-st-l} \rangle$
assumes
 $SS': \langle (S, S') \in \{(S, S''). (S, S'') \in \text{state-wl-l None} \wedge \text{correct-watching } S\} \rangle$
shows
 $\langle \text{negate-mode-bj-nonunit-wl } S \leq \Downarrow \{(S, S''). (S, S'') \in \text{state-wl-l None} \wedge \text{correct-watching } S\} \rangle$
 $\langle \text{negate-mode-bj-nonunit-l } S' \rangle$

proof —
have fresh: $\langle \text{get-fresh-index-wl } (\text{get-clauses-wl } T) (\text{get-unit-clauses-wl } T) (\text{get-watched-wl } T) \leq \Downarrow \{(i, j). i = j \wedge i \notin \# \text{dom-}m (\text{get-clauses-wl } T) \wedge i > 0 \wedge$
 $(\forall L \in \# \text{all-lits-of-mm } (mset \text{ ' \# ran-mf } (\text{get-clauses-wl } T) + \text{get-unit-clauses-wl } T) .$
 $i \notin \text{fst ' set } (\text{watched-by } T L))\} \rangle$
 $\langle \text{get-fresh-index } (\text{get-clauses-l } T') \rangle \rangle$
if $\langle (TK, TK') \in \text{find-decomp-target-wl-ref } S \rangle$ **and**
 $\langle TK = (T, K) \rangle$ **and**
 $\langle TK' = (T', K') \rangle$

```

for  $T \ T' \ K \ K' \ TK \ TK'$ 
using that by (auto simp: get-fresh-index-def equality-except-trail-wl-get-clauses-wl
  get-fresh-index-wl-def watched-by-alt-def
  intro!: RES-refine)
show ?thesis
using  $SS'$ 
unfolding negate-mode-bj-nonunit-wl-def negate-mode-bj-nonunit-l-def
apply (refine-rcg find-decomp-target-wl-find-decomp-target-l fresh
  propagate-nonunit-and-add-wl-propagate-nonunit-and-add-l)
subgoal
  using  $SS'$  unfolding negate-mode-bj-unit-l-inv-def negate-mode-bj-nonunit-wl-inv-def
  by blast
subgoal
  using  $SS'$  unfolding negate-mode-bj-nonunit-l-inv-def by blast
subgoal using  $SS'$  by (auto simp add: twl-st-wl)
apply assumption+
apply (auto simp add: equality-except-trail-wl-get-clauses-wl)
done
qed

definition negate-mode-restart-nonunit-wl-inv ::  $\langle 'v \ twl\text{-}st\text{-}wl \Rightarrow bool \rangle$  where
 $\langle \text{negate-mode-restart-nonunit-wl-inv } S \longleftrightarrow$ 
 $(\exists S' \ b. (S, S') \in \text{state-wl-l } b \wedge \text{negate-mode-restart-nonunit-l-inv } S' \wedge \text{correct-watching } S) \rangle$ 

definition restart-nonunit-and-add-wl-inv where
 $\langle \text{restart-nonunit-and-add-wl-inv } C \ i \ S \longleftrightarrow$ 
 $\text{length } C \geq 2 \wedge \text{correct-watching } S \wedge$ 
 $\text{atms-of } (\text{mset } C) \subseteq \text{atms-of-mm } (\text{clause } \# \ twl\text{-}clause\text{-of } \# \ \text{ran-mf } (\text{get-clauses-wl } S) +$ 
 $\text{get-unit-init-clss-wl } S) \rangle$ 

fun restart-nonunit-and-add-wl ::  $\langle 'v \ clause\text{-}l \Rightarrow nat \Rightarrow 'v \ twl\text{-}st\text{-}wl \Rightarrow 'v \ twl\text{-}st\text{-}wl \ nres \rangle$  where
 $\langle \text{restart-nonunit-and-add-wl } C \ i \ (M, N, D, NE, UE, Q, W) = \text{do } \{$ 
 $\text{ASSERT}(\text{restart-nonunit-and-add-wl-inv } C \ i \ (M, N, D, NE, UE, Q, W));$ 
 $\text{let } b = (\text{length } C = 2);$ 
 $\text{let } W = W(C!0 := W(C!0) @ [(i, C!1, b)]);$ 
 $\text{let } W = W(C!1 := W(C!1) @ [(i, C!0, b)]);$ 
 $\text{RETURN } (M, \text{fmupd } i \ (C, \text{True}) \ N, \text{None}, NE, UE, \{\#\}, W)$ 
 $\} \rangle$ 

definition negate-mode-restart-nonunit-wl ::  $\langle 'v \ twl\text{-}st\text{-}wl \Rightarrow 'v \ twl\text{-}st\text{-}wl \ nres \rangle$  where
 $\langle \text{negate-mode-restart-nonunit-wl} = (\lambda S. \text{do } \{$ 
 $\text{ASSERT}(\text{negate-mode-restart-nonunit-wl-inv } S);$ 
 $\text{let } C = \text{DECO-clause-l } (\text{get-trail-wl } S);$ 
 $i \leftarrow \text{SPEC}(\lambda i. i < \text{count-decided } (\text{get-trail-wl } S));$ 
 $(S, K) \leftarrow \text{find-decomp-target-wl } i \ S;$ 
 $i \leftarrow \text{get-fresh-index-wl } (\text{get-clauses-wl } S) (\text{get-unit-clauses-wl } S) (\text{get-watched-wl } S);$ 
 $\text{restart-nonunit-and-add-wl } C \ i \ S$ 
 $\} \rangle$ 

definition negate-mode-wl-inv where
 $\langle \text{negate-mode-wl-inv } S \longleftrightarrow$ 
 $(\exists S' \ b. (S, S') \in \text{state-wl-l } b \wedge \text{negate-mode-l-inv } S' \wedge \text{correct-watching } S) \rangle$ 

definition negate-mode-wl ::  $\langle 'v \ twl\text{-}st\text{-}wl \Rightarrow 'v \ twl\text{-}st\text{-}wl \ nres \rangle$  where
 $\langle \text{negate-mode-wl } S = \text{do } \{$ 

```

```

  ASSERT(negate-mode-wl-inv S);
  if count-decided (get-trail-wl S) = 1
  then negate-mode-bj-unit-wl S
  else do {
    b ← SPEC(λ-. True);
    if b then negate-mode-bj-nonunit-wl S else negate-mode-restart-nonunit-wl S
  }
}
```

lemma *correct-watching-learn-no-propa:*

```

assumes
  L1: ⟨atm-of L1 ∈ atms-of-mm (mset ‘# ran-mf N + NE)⟩ and
  L2: ⟨atm-of L2 ∈ atms-of-mm (mset ‘# ran-mf N + NE)⟩ and
  UW: ⟨atms-of (mset UW) ⊆ atms-of-mm (mset ‘# ran-mf N + NE)⟩ and
  ⟨L1 ≠ L2⟩ and
  i-dom: ⟨i ∉ dom-m N⟩ and
  ⟨∧ L. L ∈ # all-lits-of-mm (mset ‘# ran-mf N + (NE + UE)) ⇒ i ∉ fst ‘ set (W L)⟩ and
  ⟨b ⇔ length (L1 # L2 # UW) = 2⟩
shows
  ⟨correct-watching (M, fmupd i (L1 # L2 # UW, b') N,
    D, NE, UE, Q, W (L1 := W L1 @ [(i, L2, b)], L2 := W L2 @ [(i, L1, b)])) ⇔
  correct-watching (M, N, D, NE, UE, Q, W)⟩
apply (subst correct-watching-learn[OF assms(1–3, 5–6), symmetric])
unfolding correct-watching.simps clause-to-update-def
by (auto simp: assms)
```

lemma *restart-nonunit-and-add-wl-restart-nonunit-and-add-l:*

```

assumes
  SS': ⟨(S, S') ∈ {(S, S'). (S, S') ∈ state-wl-l None ∧ correct-watching S}⟩ and
  l-inv: ⟨negate-mode-restart-nonunit-l-inv S'⟩ and
  inv: ⟨negate-mode-restart-nonunit-wl-inv S⟩ and
  ⟨(m, n) ∈ nat-rel⟩ and
  ⟨m ∈ {i. i < count-decided (get-trail-wl S)}⟩ and
  ⟨n ∈ {i. i < count-decided (get-trail-l S')}⟩ and
  TK: ⟨(TK, TK') ∈ find-decomp-target-wl-ref S⟩ and
  [simp]: ⟨TK' = (T, K)⟩ and
  [simp]: ⟨TK = (T', K')⟩ and
  ij: ⟨(i, j) ∈ {(i, j). i = j ∧ i ∉ dom-m (get-clauses-wl T') ∧ i > 0 ∧
    (∀ L ∈ # all-lits-of-mm (mset ‘# ran-mf (get-clauses-wl T') + get-unit-clauses-wl T') .
      i ∉ fst ‘ set (watched-by T' L))}⟩
shows ⟨restart-nonunit-and-add-wl (DECO-clause-l (get-trail-wl S)) i T'
  ≤ SPEC (λc. (c, restart-nonunit-and-add-l
    (DECO-clause-l (get-trail-l S')) j T)
    ∈ {(S, S'').
      (S, S'') ∈ state-wl-l None ∧ correct-watching S})⟩
```

proof –

```

have [simp]: ⟨i = j⟩
using ij by auto
have le: ⟨length (DECO-clause-l (get-trail-wl S)) > 1⟩
using SS' l-inv unfolding negate-mode-restart-nonunit-l-inv-def by auto
then have 1: ⟨DECO-clause-l (get-trail-wl S) =
  DECO-clause-l (get-trail-wl S) ! 0 #
  DECO-clause-l (get-trail-wl S) ! Suc 0 # drop 2 (DECO-clause-l (get-trail-wl S))⟩
by (cases ⟨DECO-clause-l (get-trail-wl S)⟩; cases ⟨tl (DECO-clause-l (get-trail-wl S))⟩)
  auto
obtain T U b b' where
```

```

    ST:  $\langle (S, T) \in \text{state-wl-l } b \rangle$  and
     $\langle \text{no-dup } (\text{trail } (\text{state}_W\text{-of } U)) \rangle$  and
    TU:  $\langle (T, U) \in \text{twl-st-l } b' \rangle$ 
using inv unfolding negate-mode-restart-nonunit-wl-inv-def negate-mode-restart-nonunit-l-inv-def
unfolding twl-struct-invs-def cdclW-restart-mset.cdclW-all-struct-inv-def
      cdclW-restart-mset.cdclW-M-level-inv-def
by fast
then have neg: False if  $\langle \text{DECO-clause-l } (\text{get-trail-wl } S) ! 0 = \text{DECO-clause-l } (\text{get-trail-wl } S) ! \text{Suc } 0 \rangle$ 
0)
  using that
  apply (subst (asm) nth-eq-iff-index-eq)
  using le ST TU by (auto simp: twl-st twl-st-l twl-st-wl distinct-DECO-clause-l)

show ?thesis
  apply (simp only: twl-st-wl-splitD[of  $\langle \text{restart-nonunit-and-add-wl } - \rangle$ ,
    OF restart-nonunit-and-add-wl.simps]
    twl-st-l-splitD[of  $\langle \text{restart-nonunit-and-add-l } - \rangle$ ,
    OF restart-nonunit-and-add-l.simps] Let-def
    assert-bind-spec-conv)
  apply (intro conjI)
  subgoal
    using TK SS' l-inv unfolding negate-mode-restart-nonunit-l-inv-def
      restart-nonunit-and-add-wl-inv-def
    by (cases T') auto
  subgoal
    unfolding RETURN-def less-eq-nres.simps mem-Collect-eq prod.simps singleton-iff
    apply (subst subset-iff)
    unfolding RETURN-def less-eq-nres.simps mem-Collect-eq prod.simps singleton-iff
    apply (intro conjI impI allI)
    subgoal using TK SS' by (auto simp: state-wl-l-def)
    subgoal
      apply (simp only: )
      apply (subst 1)
      apply (subst One-nat-def[symmetric])+
      apply (subst fun-upd-other)
      subgoal
        using SS' length-DECO-clause-l[of  $\langle \text{get-trail-wl } S \rangle$ ] le TK
        by (cases  $\langle \text{DECO-clause-l } (\text{get-trail-wl } S) \rangle$ ; cases  $\langle \text{tl } (\text{DECO-clause-l } (\text{get-trail-wl } S)) \rangle$ )
          (auto simp: DECO-clause-l-DECO-clause[symmetric] twl-st-l twl-st
            simp del: DECO-clause-l-DECO-clause)
        apply (rule correct-watching-learn-no-propa[THEN iffD2])
        apply (rule atms-of-subset-in-atms-ofI[of  $\langle \text{DECO-clause } (\text{get-trail-wl } S) \rangle$ ])
        subgoal using TK by (solves  $\langle \text{auto simp add: mset-take-mset-drop-mset}' \rangle$ )
        subgoal using TK le by (solves  $\langle \text{auto simp add: mset-take-mset-drop-mset}'$ 
          DECO-clause-l-DECO-clause[symmetric]
          simp del: DECO-clause-l-DECO-clause)
        subgoal apply (use TK le in  $\langle \text{auto simp add: mset-take-mset-drop-mset}' \text{ DECO-clause-l-DECO-clause[symmetric]}$ 
          simp del: DECO-clause-l-DECO-clause)
          apply (smt 1 UnE add-mset-add-single image-eqI mset.simps(2) set-mset-mset subsetCE
            union-iff union-single-eq-member)
          done
        subgoal — TODO Proof
        using TK le apply (auto simp: mset-take-mset-drop-mset' in-DECO-clause-l-in-DECO-clause-iff
          dest!: in-set-dropD)
          by (metis UnE atms-of-ms-union atms-of-subset-in-atms-ofI)
        subgoal using SS' TK neg by (auto simp add: equality-except-trail-wl-get-clauses-wl)

```



```

subgoal using inv TK SS' ij unfolding negate-mode-restart-nonunit-wl-inv-def
  by (cases S; cases T; cases T')
    (auto simp: state-wl-l-def correct-watching.simps
      clause-to-update-def)
subgoal using inv TK SS' ij unfolding negate-mode-restart-nonunit-wl-inv-def
  by (cases S; cases T; cases T')
    (auto simp: state-wl-l-def correct-watching.simps
      clause-to-update-def)
subgoal by (subst 1) auto
subgoal using inv TK SS' unfolding negate-mode-restart-nonunit-wl-inv-def
  by (cases S; cases T; cases T')
    (auto simp: state-wl-l-def correct-watching.simps
      clause-to-update-def)
done
done
done
qed

```

lemma *negate-mode-restart-nonunit-wl-negate-mode-restart-nonunit-l:*

fixes $S :: \langle 'v \text{ twl-st-wl} \rangle$ **and** $S' :: \langle 'v \text{ twl-st-l} \rangle$

assumes

$SS': \langle (S, S') \in \{(S, S''). (S, S'') \in \text{state-wl-l None} \wedge \text{correct-watching } S\} \rangle$

shows

$\langle \text{negate-mode-restart-nonunit-wl } S \leq \downarrow \{(S, S''). (S, S'') \in \text{state-wl-l None} \wedge \text{correct-watching } S\} \rangle$
 $\langle \text{negate-mode-restart-nonunit-l } S' \rangle$

proof –

```

have fresh: \langle get-fresh-index-wl (get-clauses-wl T) (get-unit-clauses-wl T) (get-watched-wl T) \rangle
   $\leq \downarrow \{(i, j). i = j \wedge i \notin \# \text{dom-}m \text{ (get-clauses-wl } T) \wedge i > 0 \wedge$ 
     $(\forall L \in \# \text{all-lits-of-mm (mset '\# ran-mf (get-clauses-wl T) + get-unit-clauses-wl T) .}$ 
     $i \notin \text{fst 'set (watched-by T L)})\}$ 
     $\langle \text{get-fresh-index (get-clauses-l T')} \rangle$ 
if  $\langle (TK, TK') \in \text{find-decomp-target-wl-ref } S \rangle$  and
   $\langle TK = (T, K) \rangle$  and
   $\langle TK' = (T', K') \rangle$ 
for  $T \ T' \ K \ K' \ TK \ TK'$ 
using that by (auto simp: get-fresh-index-def equality-except-trail-wl-get-clauses-wl
  get-fresh-index-wl-def watched-by-alt-def
  intro!: RES-refine)

```

show *?thesis*

```

unfolding negate-mode-restart-nonunit-wl-def negate-mode-restart-nonunit-l-def
apply (refine-rcg find-decomp-target-wl-find-decomp-target-l fresh
  restart-nonunit-and-add-wl-restart-nonunit-and-add-l)
subgoal using  $SS'$  unfolding negate-mode-restart-nonunit-wl-inv-def by blast
subgoal using  $SS'$  by auto
subgoal using  $SS'$  by simp
subgoal unfolding negate-mode-restart-nonunit-l-inv-def by blast
subgoal using  $SS'$  by fast
apply assumption+
apply (rule SS')
apply assumption+
done

```

qed

lemma *negate-mode-wl-negate-mode-l:*

fixes $S :: \langle 'v \text{ twl-st-wl} \rangle$ **and** $S' :: \langle 'v \text{ twl-st-l} \rangle$

```

assumes
  SS':  $\langle (S, S') \in \{(S, S''). (S, S'') \in \text{state-wl-l None} \wedge \text{correct-watching } S\} \rangle$  and
  confl:  $\langle \text{get-conflict-wl } S = \text{None} \rangle$ 
shows
   $\langle \text{negate-mode-wl } S \leq$ 
     $\Downarrow \{(S, S''). (S, S'') \in \text{state-wl-l None} \wedge \text{correct-watching } S\}$ 
     $\langle \text{negate-mode-l } S' \rangle$ 
proof –
  show ?thesis
  using SS'
  unfolding negate-mode-wl-def negate-mode-l-def
  apply (refine-vcg negate-mode-bj-nonunit-wl-negate-mode-bj-nonunit-l
    negate-mode-bj-unit-wl-negate-mode-bj-unit-l
    negate-mode-restart-nonunit-wl-negate-mode-restart-nonunit-l)
  subgoal unfolding negate-mode-wl-inv-def by blast
  subgoal by auto
  subgoal by auto
  done
qed

context
  fixes P ::  $\langle 'v \text{ literal set} \Rightarrow \text{bool} \rangle$ 
begin

definition cdcl-twl-enum-inv-wl ::  $\langle 'v \text{ twl-st-wl} \Rightarrow \text{bool} \rangle$  where
   $\langle \text{cdcl-twl-enum-inv-wl } S \longleftrightarrow$ 
     $(\exists S'. (S, S') \in \text{state-wl-l None} \wedge \text{cdcl-twl-enum-inv-l } S') \wedge$ 
     $\text{correct-watching } S \rangle$ 

definition cdcl-twl-enum-wl ::  $\langle 'v \text{ twl-st-wl} \Rightarrow \text{bool nres} \rangle$  where
   $\langle \text{cdcl-twl-enum-wl } S = \text{do } \{$ 
     $S \leftarrow \text{cdcl-twl-stgy-prog-wl } S;$ 
     $S \leftarrow \text{WHILE}_T^{\text{cdcl-twl-enum-inv-wl}}$ 
     $(\lambda S. \text{get-conflict-wl } S = \text{None} \wedge \text{count-decided}(\text{get-trail-wl } S) > 0 \wedge$ 
     $\neg P (\text{lits-of-l } (\text{get-trail-wl } S)))$ 
     $(\lambda S. \text{do } \{$ 
     $S \leftarrow \text{negate-mode-wl } S;$ 
     $\text{cdcl-twl-stgy-prog-wl } S$ 
     $\})$ 
     $S;$ 
     $\text{if } \text{get-conflict-wl } S = \text{None}$ 
     $\text{then RETURN } (\text{if } \text{count-decided}(\text{get-trail-wl } S) = 0 \text{ then } P (\text{lits-of-l } (\text{get-trail-wl } S)) \text{ else True})$ 
     $\text{else RETURN } (\text{False})$ 
     $\}$ 
   $\rangle$ 

lemma cdcl-twl-enum-wl-cdcl-twl-enum-l:
assumes
  SS':  $\langle (S, S') \in \text{state-wl-l None} \rangle$  and
  corr:  $\langle \text{correct-watching } S \rangle$ 
shows
   $\langle \text{cdcl-twl-enum-wl } S \leq \Downarrow \text{bool-rel}$ 
     $\langle \text{cdcl-twl-enum-l } P \ S' \rangle$ 
unfolding cdcl-twl-enum-wl-def cdcl-twl-enum-l-def
apply (refine-vcg cdcl-twl-stgy-prog-wl-spec'[unfolded fref-param1, THEN fref-to-Down]
  negate-mode-wl-negate-mode-l)
subgoal by fast

```

subgoal using SS' *corr* by *auto*
subgoal using *corr unfolding* *cdcl-tw1-enum-inv-wl-def* by *blast*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done

end

end