Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

January 20, 2020

Contents

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

lemma Ex-sat-model:
  assumes ‹satisfiable (set-mset N)›
  shows ‹∃ M. set M ⊨sm N ∧
           distinct M ∧
           consistent-interp (set M) ∧
           atm-of ‘ set M ⊆ atms-of-mm N›
proof –
  from assms obtain I where
    I-N: ‹I ⊨sm N› and
    consistent: ‹consistent-interp I› and
    ‹total-over-m I (set-mset N)› and
    atms-I-N: ‹atm-of ‘ I = atms-of-mm N›
    unfolding satisfiable-def-min by blast
  have ‹I ⊆ Pos ‘ (atms-of-mm N) ∪ Neg ‘ (atms-of-mm N)›
    using atms-I-N
    by (smt in-set-image-subsetD literal.exhaust-sel subsetI sup-ge1 sup-ge2)
  then have ‹finite I›
    using infinite-super by fastforce
  then obtain I' where I': ‹I = set I'› and dist: ‹distinct I'›
    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
  ‹all-models N = {M. set M ⊨sm N ∧ consistent-interp (set M) ∧
    distinct M ∧ atm-of ‘ set M ⊆ atms-of-mm N}›

lemma finite-all-models:
  ‹finite (all-models N)›
proof –
  let ?n = ‹Pos ‘ (atms-of-mm N) ∪ Neg ‘ (atms-of-mm N)›
  have H: ‹all-models N ⊆ {M. set M ⊆ ?n ∧ length M ≤ card ?n}›
    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 *rtranclp-next-model-filtered-mono*:
‹*next-model-filtered*** *a b* ⟹ *snd a* ⊆# *snd b*›
by (*induction rule*: *rtranclp-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 *rtranclp-next-filtered-same-atoms*:
‹*next-model-filtered*** *a b* ⟹ *atms-of-mm* (*snd b*) = *atms-of-mm* (*snd a*)›
by (*induction rule*: *rtranclp-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 *rtranclp-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*: *rtranclp-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
‹*M* ∈# *snd y* − *snd a*› |
‹*M* ∈# *snd z* − *snd y*›
using *step star M-in*
by (*smt rtranclp-next-model-filtered-mono add-diff-cancel-right in-multiset-minus-notin-snd rtranclp.rtrancl-into-rtrancl 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 rtranclp-next-model-filtered-mono*[*OF star*] *rtranclp-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 *rtranclp-next-model-filtered-next-false*:
‹*next-model-filtered*** *a b* ⟹ *M* ∈# *snd b* − *snd a* ⟹ *M* = *image-mset uminus* (*mset M′*) ⟹ ¬*P* (*uminus* ‘ *set-mset M*)›
proof (*induction arbitrary*: *M M′ rule*: *rtranclp-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
‹*M* ∈# *snd y* − *snd a*› |
‹*M* ∈# *snd z* − *snd y*›
using *step star M-in*
by (*smt rtranclp-next-model-filtered-mono add-diff-cancel-right in-multiset-minus-notin-snd rtranclp.rtrancl-into-rtrancl 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 rtranclp-next-model-filtered-mono*[*OF star*] *rtranclp-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 ‹unsatisfiable (set-mset N)›
by (metis Ex-sat-model Model-Enumeration.next-model-filtered.simps
assms next-model.intros)

lemma unsat-no-step-next-model-filtered:
assumes ‹unsatisfiable (set-mset N)›
shows ‹no-step next-model-filtered (None, N)›
by (metis (no-types, lifting) next-model-filtered.simps assms
next-model.cases satisfiable-carac′ snd-conv)

lemma full-next-model-filtered-no-distinct-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) ∧ distinct M›
proof clarify
fix M
assume
M-N: ‹set M ⊨m N› **and**
P-M: ‹P (set M)› **and**
consistent: ‹consistent-interp (set M)› **and**
dist-M: ‹distinct M›
have st: ‹next-model-filtered** (None, N) (None, N′)› **and**
ns: ‹no-step next-model-filtered (None, N′)›
using no-model **unfolding** full-def **by** blast+
define Ms **where** ‹Ms = N′ − N›
then have N′[simp]: ‹N′ = N + Ms›
using rtranclp-next-model-filtered-mono[OF st] **by** auto
have ‹unsatisfiable (set-mset N′)›
using ns **by** (rule no-step-next-model-filtered-unsat)
then have ‹¬set M ⊨m Ms›
using consistent M-N **by** (auto simp: satisfiable-carac[symmetric])
then obtain M′ **where**
M′-MS: ‹M′ ∈# Ms› **and**
M-M′: ‹¬set M ⊨ M′›
by (auto simp: true-cls-mset-def)
obtain M′′ **where**
[simp]: ‹M′ = mset M′′›
using ex-mset[of M′] **by** auto
let ?M′′ = ‹map uminus M′′›
have ‹next-model ?M′′ (snd (None :: ′v literal list option, N))›
apply (rule rtranclp-next-model-filtered-next-modelD[OF st, of M′])
using M′-MS **by** auto
then have
cons′: ‹consistent-interp (set ?M′′)› **and**
M′′-N: ‹set ?M′′ ⊨sm N› **and**
dist-M′′: ‹distinct ?M′′›
unfolding next-model.simps **by** auto

let ?I = ‹remdups (M @ ?M′′)›
have cons-I: ‹consistent-interp (set ?I)›
using M-M′ consistent cons′ **by** (auto simp: consistent-interp-def true-cls-def)

have ‹P (set ?I)›

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 *rtranclp-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-statisfiable*:
‹(∃*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-statisfiable 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 that 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* :: ‹(*'v*, *'a*) *ann-lits* ⇒ *'v clause*›**where**
‹*DECO-clause M* = (*uminus o lit-of*) ‘# (*filter-mset is-decided* (*mset M*))›

lemma *distinct-mset-DECO*:
‹*distinct-mset* (*DECO-clause M*) ⟷ *distinct-mset* (*lit-of* ‘# *filter-mset is-decided* (*mset M*))›
(**is** ‹*?A* ⟷ *?B*›)
proof −
have ‹*?A* ⟷ *distinct-mset* (*uminus* ‘# *lit-of* ‘# (*filter-mset is-decided* (*mset M*)))›
by (*auto simp*: *DECO-clause-def*)
also have ‹... ⟷ *distinct-mset* (*lit-of* ‘# (*filter-mset is-decided* (*mset M*)))›
apply (*subst distinct-image-mset-inj*)
subgoal by (*auto simp*: *inj-on-def*)
subgoal by *auto*
done
finally show *?thesis*
.
qed

lemma [*twl-st*]:
‹*init-clss* ($state_W$*-of T*) = *get-all-init-clss T*›
‹*learned-clss* ($state_W$*-of T*) = *get-all-learned-clss T*›
by (*cases T*; *auto simp*: $cdcl_W$*-restart-mset-state*; *fail*)+

lemma *atms-of-DECO-clauseD*:
‹*x* ∈ *atms-of* (*DECO-clause U*) ⟹ *x* ∈ *atms-of-s* (*lits-of-l U*)›
‹*x* ∈ *atms-of* (*DECO-clause U*) ⟹ *x* ∈ *atms-of* (*lit-of* ‘# *mset U*)›
by (*auto simp*: *DECO-clause-def atms-of-s-def atms-of-def lits-of-def*)

definition *TWL-DECO-clause* **where**
‹*TWL-DECO-clause M* =
TWL-Clause
((*uminus o lit-of*) ‘# *mset* (*take 2* (*filter is-decided M*)))
((*uminus o lit-of*) ‘# *mset* (*drop 2* (*filter is-decided M*)))›

lemma *clause-TWL-Deco-clause*[*simp*]: ‹*clause* (*TWL-DECO-clause M*) = *DECO-clause M*›
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-twl* :: ‹′*v twl-st* ⇒ ′*v twl-st* ⇒ *bool*› **where**
bj-unit:
‹*negate-model-and-add-twl* (*M*, *N*, *U*, *None*, *NP*, *UP*, *WS*, *Q*)
(*Propagated* (−*K*) (*DECO-clause M*) # *M1*, *N*, *U*, *None*, *add-mset* (*DECO-clause M*) *NP*, *UP*, {#}, {#*K*#})›
if
‹(*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition M*)› **and**
‹*get-level M K* = *count-decided M*› **and**
‹*count-decided M* = *1*› |
bj-nonunit:
‹*negate-model-and-add-twl* (*M*, *N*, *U*, *None*, *NP*, *UP*, *WS*, *Q*)
(*Propagated* (−*K*) (*DECO-clause M*) # *M1*, *add-mset* (*TWL-DECO-clause M*) *N*, *U*, *None*, *NP*, *UP*, {#},
{#*K*#})›
if
‹(*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition M*)› **and**
‹*get-level M K* = *count-decided M*› **and**
‹*count-decided M* ≥ *2*› |
restart-nonunit:
‹*negate-model-and-add-twl* (*M*, *N*, *U*, *None*, *NP*, *UP*, *WS*, *Q*)
(*M1*, *add-mset* (*TWL-DECO-clause M*) *N*, *U*, *None*, *NP*, *UP*, {#}, {#})›
if
‹(*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition M*)› **and**
‹*get-level M K* < *count-decided M*› **and**
‹*count-decided M* > *1*›

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: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* (*M*′, *N*, *U*, *None*)› **and**
stgy-invs: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-stgy-invariant* (*M*′, *N*, *U*, *None*)› **and**
smaller-propa: ‹$cdcl_W$*-restart-mset.no-smaller-propa* (*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**
no-confl: ‹∀ *C*∈#*N*′. ∀ *M1 K M2*. *M*′ = *M2* @ *Decided K* # *M1* ⟶ ¬*M1* ⊨*as CNot C*› **and**
no-propa: ‹∀ *C*∈#*N*′. ∀ *M1 K M2 L*. *M*′ = *M2* @ *Decided K* # *M1* ⟶ *L* ∈# *C* ⟶
¬*M1* ⊨*as CNot* (*remove1-mset L C*)›
shows
‹$cdcl_W$*-restart-mset.*$cdcl_W$*-stgy*** ([], *N*+*N*′, *U*′, *None*) (*drop* (*length M*′ − *n*) *M*′, *N*+ *N*′, *U*′, *None*)›
proof −
let *?S* = ‹λ*n*. (*drop* (*length M*′ − *n*) *M*′, *N*+*N*′, *U*′, *None*)›
note $cdcl_W$*-restart-mset-state*[*simp*]
have
M-lev: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv* (*M*′, *N*, *U*, *None*)› **and**

alien: ‹*cdcl*$_W$*-restart-mset.no-strange-atm* (*M'*, *N*, *U*, *None*)› **and**
confl: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-conflicting* (*M'*, *N*, *U*, *None*)› **and**
learned: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-learned-clause* (*M'*, *N*, *U*, *None*)›
using *inv* **unfolding** *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def* **by** *fast+*

have *smaller-confl*: ‹*cdcl*$_W$*-restart-mset.no-smaller-confl* (*M'*, *N*, *U*, *None*)›
using *stgy-invs* **unfolding** *cdcl*$_W$*-restart-mset.cdcl*$_W$*-stgy-invariant-def* **by** *blast*
have *n-d*: ‹*no-dup M'*›
using *M-lev* **unfolding** *cdcl*$_W$*-restart-mset.cdcl*$_W$*-M-level-inv-def* **by** *simp*
let *?L* = ‹λ*m*. *M'* ! (*length M'* − *Suc m*)›
have *undef-nth-Suc*:
‹*undefined-lit* (*drop* (*length M'* − *m*) *M'*) (*lit-of* (*?L m*))›
if ‹*m* < *length M'*›
for *m*
proof −
define *k* **where**
‹*k* = *length M'* − *Suc m*›
then have *Sk*: ‹*length M'* − *m* = *Suc k*›
using *that* **by** *linarith*
have *k-le-M'*: ‹*k* < *length M'*›
using *that* **unfolding** *k-def* **by** *linarith*
have *n-d'*: ‹*no-dup* (*take k M'* @ *?L m* # *drop* (*Suc k*) *M'*)›
using *n-d*
apply (*subst* (*asm*) *append-take-drop-id*[*symmetric*, *of* - ‹*Suc k*›])
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*:
‹*atm-of* (*lit-of* (*M'* ! *m*)) ∈ *atms-of-mm N*›
if ‹*m* < *length M'*›
for *m*
using *alien that*
by (*auto simp*: *cdcl*$_W$*-restart-mset.no-strange-atm-def lits-of-def*)
then have *atm-in'*:
‹*atm-of* (*lit-of* (*M'* ! *m*)) ∈ *atms-of-mm* (*N* + *N'*)›
if ‹*m* < *length M'*›
for *m*
using *alien that*
by (*auto simp*: *cdcl*$_W$*-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) ‹m < length M'› |
    (ge) ‹m ≥ length M'›
    by linarith
  then show ?case
  proof (cases)
    case ge
    then show ?thesis
      using Suc by auto
  next
    case le
    define k where
      ‹k = length M' − Suc m›
    then have Sk: ‹length M' − m = Suc k›
      using le by linarith
    have k-le-M': ‹k < length M'›
      using le unfolding k-def by linarith
    have kept': ‹∀L E. Propagated L E ∈ set (drop (length M' − m) M') ⟶ E ∈# N + U'›
      using kept k-le-M' unfolding k-def[symmetric] Sk
      by (subst (asm) Cons-nth-drop-Suc[symmetric]) auto
    have M': ‹M' = take (length M' − Suc m) M' @ ?L m # trail (?S m)›
      apply (subst append-take-drop-id[symmetric, of - ‹Suc k›])
      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 ‹cdcl_W-restart-mset.cdcl_W-stgy (?S m) (?S (Suc m))›
    proof (cases ‹?L (m)›)
      case (Decided K) note K = this
      have dec: ‹cdcl_W-restart-mset.decide (?S m) (?S (Suc m))›
        apply (rule cdcl_W-restart-mset.decide-rule[of - ‹lit-of (?L m)›])
        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: ‹M' ! k = Decided K›
        using K unfolding k-def[symmetric] Sk .

      have H: ‹D + {#L#} ∈# N + U ⟶ undefined-lit (trail (?S m)) L ⟶
          ¬ (trail (?S m)) ⊨as CNot D› for D L
        using smaller-propa unfolding cdcl_W-restart-mset.no-smaller-propa-def
          trail.simps clauses-def
          cdcl_W-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: ‹False›
        if
          ‹drop (Suc k) M' ⊨as CNot (remove1-mset L E)› and
          ‹L ∈# E› 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 cdcl_W-restart-mset.propagate (?S m)›
  using no-propa no-new-propa
  by (auto simp: cdcl_W-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 cdcl_W-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 cdcl_W-restart-mset.no-smaller-confl-def
    trail.simps clauses-def cdcl_W-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 cdcl_W-restart-mset.conflict (?S m)›
  using no-new-confl
  by (auto simp: cdcl_W-restart-mset.conflict.simps clauses-def state-eq-def
      k-def[symmetric] Sk)
show ?thesis
  apply (rule cdcl_W-restart-mset.cdcl_W-stgy.other')
    apply (rule nsc)
   apply (rule nss)
  apply (rule cdcl_W-restart-mset.cdcl_W-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 cdcl_W-restart-mset.cdcl_W-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*: ‹$C \in\# N + U'$›
using *learned kept* **unfolding** *cdcl*$_W$*-restart-mset.cdcl*$_W$*-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 ‹*cdcl*$_W$*-restart-mset.propagate* (*?S m*) (*?S* (*Suc m*))›
apply (*rule cdcl*$_W$*-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 cdcl*$_W$*-restart-mset.cdcl*$_W$*-stgy.propagate'*)
qed
then show *?thesis*
using *IH*[*OF kept'*] **by** *simp*
qed
qed
qed

lemma *after-fast-restart-replay'*:
assumes
inv: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*M'*, *N*, *U*, *None*)› **and**
stgy-invs: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-stgy-invariant* (*M'*, *N*, *U*, *None*)› **and**
smaller-propa: ‹*cdcl*$_W$*-restart-mset.no-smaller-propa* (*M'*, *N*, *U*, *None*)› **and**
kept: ‹$\forall L\ E.$ *Propagated L E* $\in$ *set* (*drop* (*length* $M' - n$) *M'*) $\longrightarrow E \in\# N + U'$› **and**
U'-U: ‹$U' \subseteq\# U$› **and**
N-N': ‹$N \subseteq\# N'$› **and**
no-confl: ‹$\forall C{\in}\#N'{-}N.\ \forall M1\ K\ M2.\ M' = M2$ @ *Decided K* # *M1* $\longrightarrow \neg M1 \models as$ *CNot C*› **and**
no-propa: ‹$\forall C{\in}\#N'{-}N.\ \forall M1\ K\ M2\ L.\ M' = M2$ @ *Decided K* # *M1* $\longrightarrow L \in\# C \longrightarrow$
$\neg M1 \models as$ *CNot* (*remove1-mset L C*)›
shows
‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-stgy*** ([], *N'*, *U'*, *None*) (*drop* (*length* $M' - n$) *M'*, *N'*, *U'*, *None*)›
using *after-fast-restart-replay*[*OF inv stgy-invs smaller-propa kept U'-U*, *of* ‹$N' - N$›]
no-confl no-propa N-N'
by *auto*

lemma *after-fast-restart-replay-no-stgy*:
assumes
inv: ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*M'*, *N*, *U*, *None*)› **and**
kept: ‹$\forall L\ E.$ *Propagated L E* $\in$ *set* (*drop* (*length* $M' - n$) *M'*) $\longrightarrow E \in\# N{+}N' + U'$› **and**
U'-U: ‹$U' \subseteq\# U$›
shows
‹*cdcl*$_W$*-restart-mset.cdcl*$_W$** ([], *N+N'*, *U'*, *None*) (*drop* (*length* $M' - n$) *M'*, *N+N'*, *U'*, *None*)›
proof −
let *?S* = ‹$\lambda n.$ (*drop* (*length* $M' - n$) *M'*, *N* + *N'*, *U'*, *None*)›
note *cdcl*$_W$*-restart-mset-state*[*simp*]
have

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

have *n-d*: ‹*no-dup M'*›
using *M-lev* **unfolding** *cdcl*$_W$*-restart-mset.cdcl*$_W$*-M-level-inv-def* **by** *simp*
let *?L* = ‹λ*m*. *M'* ! (*length M'* − *Suc m*)›
have *undef-nth-Suc*:
‹*undefined-lit* (*drop* (*length M'* − *m*) *M'*) (*lit-of* (*?L m*))›
if ‹*m* < *length M'*›
for *m*
proof −
define *k* **where**
‹*k* = *length M'* − *Suc m*›
then have *Sk*: ‹*length M'* − *m* = *Suc k*›
using *that* **by** *linarith*
have *k-le-M'*: ‹*k* < *length M'*›
using *that* **unfolding** *k-def* **by** *linarith*
have *n-d'*: ‹*no-dup* (*take k M'* @ *?L m* # *drop* (*Suc k*) *M'*)›
using *n-d*
apply (*subst* (*asm*) *append-take-drop-id*[*symmetric*, *of* - ‹*Suc k*›])
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*:
‹*atm-of* (*lit-of* (*M'* ! *m*)) ∈ *atms-of-mm* (*N*+*N'*)›
if ‹*m* < *length M'*›
for *m*
using *alien that*
by (*auto simp*: *cdcl*$_W$*-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*) ‹*m* < *length M'*› |
(*ge*) ‹*m* ≥ *length M'*›
by *linarith*
then show *?case*

```
proof cases
  case ge
  then show ?thesis
    using Suc by auto
next
  case le
  define k where
    ‹k = length M' − Suc m›
  then have Sk: ‹length M' − m = Suc k›
    using le by linarith
  have k-le-M': ‹k < length M'›
    using le unfolding k-def by linarith
  have kept': ‹∀L E. Propagated L E ∈ set (drop (length M' − m) M') ⟶ E ∈# N+N' + U'›
    using kept k-le-M' unfolding k-def[symmetric] Sk
    by (subst (asm) Cons-nth-drop-Suc[symmetric]) auto
  have M': ‹M' = take (length M' − Suc m) M' @ ?L m # trail (?S m)›
    apply (subst append-take-drop-id[symmetric, of - ‹Suc k›])
    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 ‹cdcl_W-restart-mset.cdcl_W (?S m) (?S (Suc m))›
  proof (cases ‹?L (m)›)
    case (Decided K) note K = this
    have dec: ‹cdcl_W-restart-mset.decide (?S m) (?S (Suc m))›
      apply (rule cdcl_W-restart-mset.decide-rule[of - ‹lit-of (?L m)›])
      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: ‹M' ! k = Decided K›
      using K unfolding k-def[symmetric] Sk .

    show ?thesis
      apply (rule cdcl_W-restart-mset.cdcl_W.intros(3))
      apply (rule cdcl_W-restart-mset.cdcl_W-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 cdcl_W-restart-mset.cdcl_W-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: ‹C ∈# N+N' + U'›
      using learned kept unfolding cdcl_W-restart-mset.cdcl_W-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 ‹cdcl_W-restart-mset.propagate (?S m) (?S (Suc m))›
          apply (rule cdcl_W-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 cdcl_W-restart-mset.cdcl_W.intros)
      qed
      then show ?thesis
        using IH[OF kept'] by simp
    qed
  qed
qed

lemma after-fast-restart-replay-no-stgy':
  assumes
    inv: ‹cdcl_W-restart-mset.cdcl_W-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
    ‹cdcl_W-restart-mset.cdcl_W** ([], 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: ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (M, N, U + U', D)›
 shows ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (M, N + U', U, D)›
  using assms
  unfolding cdcl_W-restart-mset.cdcl_W-all-struct-inv-def
        cdcl_W-restart-mset.cdcl_W-M-level-inv-def cdcl_W-restart-mset.distinct-cdcl_W-state-def
        cdcl_W-restart-mset.cdcl_W-learned-clause-alt-def cdcl_W-restart-mset.cdcl_W-conflicting-def
        cdcl_W-restart-mset.no-strange-atm-def cdcl_W-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
  subgoal by (auto simp: ac-simps)
  subgoal by (auto simp: ac-simps)
  subgoal by auto
  done

lemma twl-struct-invs-move-to-init:
  assumes ‹twl-struct-invs (M, N, U + U', D, NP, UP, WS, Q)›
  shows ‹twl-struct-invs (M, N + U', U, D, NP, UP, WS, Q)›
proof –
  have H: ‹N + (U + U') = N + U' + U›
    by simp
  have struct-invs:
    ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (M, clauses N + NP, clauses (U + U') + UP, D')⟹
    cdcl_W-restart-mset.cdcl_W-all-struct-inv (M, clauses (N + U') + NP, clauses U + UP, D')›
    for D'
    using cdcl_W-all-struct-inv-move-to-init[of M ‹clauses N + NP› ‹clauses U + UP›
      ‹clauses U'› D']
    by (auto simp: ac-simps)
  have smaller: ‹clauses N + NP + (clauses (U + U') + UP) = clauses (N + U') + NP + (clauses
U + UP)›
    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 state_W-of.simps
      cdcl_W-restart-mset.no-smaller-propa-def cdcl_W-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* :: ‹*'v twl-st*›
assumes
‹*negate-model-and-add-twl S T*› **and**
‹*twl-struct-invs S*›
shows ‹*twl-struct-invs T*›
using *assms*
proof (*induction rule*: *negate-model-and-add-twl.induct*)
fix *K* :: ‹*'v literal*› **and** *M1 M2 M N U NP UP WS Q*
assume
decomp: ‹(*Decided K # M1*, *M2*) ∈ *set* (*get-all-ann-decomposition M*)› **and**
inv: ‹*twl-struct-invs* (*M*, *N*, *U*, *None*, *NP*, *UP*, *WS*, *Q*)›

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
st-invs: ‹*twl-st-inv ?S*› **and**
‹*valid-enqueued ?S*› **and**
struct-invs: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* ($state_W$*-of ?S*)› **and**
no-smaller: ‹$cdcl_W$*-restart-mset.no-smaller-propa* ($state_W$*-of ?S*)› **and**
‹*twl-st-exception-inv ?S*› **and**
‹*no-duplicate-queued ?S*› **and**
‹*distinct-queued ?S*› **and**
‹*confl-cands-enqueued ?S*› **and**
‹*propa-cands-enqueued ?S*› **and**
‹*get-conflict ?S* ≠ *None* ⟶ *clauses-to-update ?S* = {#} ∧ *literals-to-update ?S* = {#}› **and**
entailed: ‹*entailed-clss-inv ?S*› **and**
‹*clauses-to-update-inv ?S*› **and**
past: ‹*past-invs ?S*›
using *inv* **unfolding** *twl-struct-invs-def*
by *fast+*
obtain *M3* **where**
M: ‹*M* = *M3* @ *M2* @ *Decided K # M1*›

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

have *dist-filtered*: ⟨*distinct-mset* (*lit-of* '# *mset* (*filter is-decided M*))⟩
apply (*rule distinct-mset-mono*[*of* - ⟨*lit-of* '# *mset M*⟩])
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'*: ⟨*distinct-mset* (*uminus* '# *lit-of* '# *mset* (*filter is-decided M*))⟩
apply (*subst distinct-image-mset-inj*)
subgoal by (*auto simp*: *inj-on-def*)
subgoal .
done
have *cdcl-W*: ⟨$cdcl_W$*-restart-mset.*$cdcl_W^{**}$ ([], *clauses* (*add-mset* (*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)⟩
apply (*rule after-fast-restart-replay-no-stgy'*[*OF struct-invs*[*unfolded* $state_W$*-of.simps*]])
subgoal
apply (*intro allI impI conjI*)
subgoal for *L E*
by (*use M' struct-invs* $cdcl_W$*-restart-mset.in-get-all-mark-of-propagated-in-trail*[*of E M*]
in ⟨*auto simp add*: $cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause-alt-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def* $cdcl_W$*-restart-mset-state clauses-def*⟩)
done
subgoal by *simp*
subgoal by *simp*
done

have ⟨*distinct-mset* (*DECO-clause M*)⟩
using *dist-filtered'* **unfolding** *DECO-clause-def*
by (*simp add*: *mset-filter*)

then have *struct-invs-S'*:
⟨$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv* ([], *clauses* (*add-mset* (*TWL-DECO-clause M*) *N*) + *NP*, *clauses U* + *UP*, *None*)⟩
using *struct-invs*
by (*auto simp*: $cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-M-level-inv-def* $cdcl_W$*-restart-mset.distinct-*$cdcl_W$*-state-def*
$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clause-alt-def* $cdcl_W$*-restart-mset.*$cdcl_W$*-conflicting-def*
$cdcl_W$*-restart-mset.no-strange-atm-def* $cdcl_W$*-restart-mset-state*)
with *cdcl-W* **have** *struct-invs-add*: ⟨$cdcl_W$*-restart-mset.*$cdcl_W$*-all-struct-inv*
(*M1*, *clauses* (*add-mset* (*TWL-DECO-clause M*) *N*) + *NP*, *clauses U* + *UP*, *None*)⟩
by (*auto intro*: $cdcl_W$*-restart-mset.rtranclp-*$cdcl_W$*-all-struct-inv-inv simp*: *M'*
dest!: $cdcl_W$*-restart-mset.rtranclp-*$cdcl_W$*-*$cdcl_W$*-restart*)
have *no-smaller-M1*:
⟨$cdcl_W$*-restart-mset.no-smaller-propa* ($state_W$*-of* (*M1*, *N*, *U*, *None*, *NP*, *UP*, *WS*, *Q*))⟩
using *no-smaller* **by** (*auto simp*: $cdcl_W$*-restart-mset.no-smaller-propa-def*
$cdcl_W$*-restart-mset-state clauses-def M'*)
have *no-smaller-add*:
⟨$cdcl_W$*-restart-mset.no-smaller-propa*
(*M1*, *clauses* (*add-mset* (*TWL-DECO-clause M*) *N*) + *NP*, *clauses U* + *UP*, *None*)⟩
unfolding $state_W$*-of.simps* $cdcl_W$*-restart-mset.no-smaller-propa-def*
$cdcl_W$*-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*: $cdcl_W$*-restart-mset.no-smaller-propa-def*
$cdcl_W$*-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: ‹*get-level M K* = *count-decided M*› **and**
count-dec: ‹*count-decided M* ≥ *2*›
have [*simp*]: ‹*filter is-decided M2*′ = []›
using *count-dec lev no-dup* **unfolding** *M*′
by (*auto simp*: *TWL-DECO-clause-def count-decided-def add-mset-eq-add-mset M*′)
obtain *L*′ *C* **where**
filter-M: ‹*filter is-decided M* = *Decided K* # *Decided L*′ # *C*›
using *count-dec lev* **unfolding** *M*′
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 M*′
filter-eq-Cons-iff tl-append)
then have *deco-M*: ‹*TWL-DECO-clause M* = *TWL-Clause* {#−*K*, −*L*′#} (*uminus* ‘# *lit-of* ‘#
mset C)›
by (*auto simp*: *TWL-DECO-clause-def*)
have *C-M1*: ‹*C* = *tl* (*filter is-decided M1*)›
using *filter-M* **unfolding** *M*′
by *auto*
then obtain *M1*″ *M1*′ **where**
M1: ‹*M1* = *M1*″ @ *Decided L*′ # *M1*′›
by (*metis* (*no-types*, *lifting*) *M*′ ‹*filter is-decided M2*′ = []› *append-self-conv2*
filter.simps(*2*) *filter-M filter-append filter-eq-Cons-iff list.sel*(*3*))
then have [*simp*]: ‹*count-decided M1*″ = *0*› **and** *filter-M1*″: ‹*filter is-decided M1*″ = []›
using *filter-M no-dup* **unfolding** *C-M1 M1 M*′
by (*auto simp*: *tl-append count-decided-def dest*: *filter-eq-ConsD split*: *list.splits*)
have *C-in-M1*: ‹*lits-of-l C* ⊆ *lits-of-l M1*›
unfolding *C-M1* **by** (*auto simp*: *lits-of-def dest*: *in-set-tlD*)

let *?S*′ = ‹(*M1*, *add-mset* (*TWL-DECO-clause M*) *N*, *U*, *None*, *NP*, *UP*,
add-mset (−*L*′, (*TWL-DECO-clause M*)) {#}, {#})›
let *?T*′ = ‹(*Propagated* (−*K*) (*DECO-clause M*) # *M1*, *add-mset* (*TWL-DECO-clause M*) *N*, *U*,
None,
NP, *UP*, {#}, {#− (−*K*)#})›
have *propa*: ‹*cdcl-twl-cp ?S*′ *?T*′›
unfolding *clause-TWL-Deco-clause*[*symmetric*]
apply (*rule cdcl-twl-cp.propagate*)
subgoal by (*auto simp*: *deco-M*)
subgoal using *no-dup* **unfolding** *M* **by** *auto*
subgoal using *C-in-M1* **unfolding** *deco-M* **by** (*auto simp*: *lits-of-def*)
done

have *struct-invs-S*′: ‹$cdcl_W$-*restart-mset*.$cdcl_W$-*all-struct-inv* ($state_W$-*of ?S*′)›
using *struct-invs-add* **by** *auto*

have *no-smaller-S*′: ‹$cdcl_W$-*restart-mset.no-smaller-propa* ($state_W$-*of ?S*′)›
using *no-smaller-add* **by** *simp*
have [*simp*]: ‹*get-level M1 L*′ = *count-decided M1*›
using *no-dup* **unfolding** *M*′ *M1* **by** *auto*
have ‹*watched-literals-false-of-max-level M1* (*TWL-DECO-clause M*)›
using *no-dup* **apply** (*subst* (*asm*) *M*′)
by (*auto simp*: *deco-M add-mset-eq-add-mset dest*: *in-lits-of-l-defined-litD*)
moreover have ‹*struct-wf-twl-cls* (*TWL-DECO-clause M*)›
using *dist-filtered*′ **unfolding** *deco-M filter-M*
by (*auto simp*: *simp del*: *clause-TWL-Deco-clause*)
ultimately have ‹*twl-st-inv ?S*′›
using *wf-N-U st-invs-M1*′ **unfolding** *twl-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 ‹*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*state*$_W$*-of ?S'*)›
using *struct-invs-S'* **.**
moreover have ‹*cdcl*$_W$*-restart-mset.no-smaller-propa* (*state*$_W$*-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*]: ‹*L′ ∉ lits-of-l M1a*› ‹*K ∉ lits-of-l M1a*›
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: ‹*filter is-decided M2′ = []*›
elim!: *list-match-lel-lel*)
have ‹*twl-lazy-update M1a* (*TWL-DECO-clause M*)›
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 ‹*watched-literals-false-of-max-level M1a* (*TWL-DECO-clause M*)›
unfolding *deco-M* **by** (*auto simp*: *add-mset-eq-add-mset*)
moreover have ‹*twl-exception-inv ?SM1a* (*TWL-DECO-clause M*)›
unfolding *deco-M* **by** (*auto simp*: *add-mset-eq-add-mset twl-exception-inv.simps*)
ultimately have ‹*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› **for** *C*
using *struct*[*of C*]
by (*auto simp*: *twl-exception-inv.simps*)
then 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›
by *blast*
show ‹*confl-cands-enqueued ?SM1a*›
using *confl* **by** (*auto simp*: *deco-M*)
show ‹*propa-cands-enqueued ?SM1a*›
using *propa* **by** (*auto simp*: *deco-M*)
show ‹*clauses-to-update-inv ?SM1a*›
using *clss-to-upd*
by (*auto simp*: *deco-M clauses-to-update-prop.simps add-mset-eq-add-mset*)
qed
moreover have ‹*get-conflict ?S′ = None*›
by *simp*
ultimately have ‹*twl-struct-invs ?S′*›
unfolding *twl-struct-invs-def*
by *meson*
then have ‹*twl-struct-invs ?T′*›
by (*rule cdcl-twl-cp-twl-struct-invs*[*OF propa*])
then show ‹*twl-struct-invs* (*Propagated* (−*K*) (*DECO-clause M*) # *M1*, *add-mset* (*TWL-DECO-clause M*) *N*,
U, *None*, *NP*, *UP*, {#}, {#*K*#})›
by *simp*
}

{
let *?S* = ‹(*Propagated* (− *K*) (*DECO-clause M*) # *M1*, *N*, *U*, *None*, *add-mset* (*DECO-clause M*) *NP*, *UP*,
{#}, {#*K*#})›
assume ‹*count-decided M = 1*›
then have [*simp*]: ‹*DECO-clause M* = {#−*K*#}›
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*]: ‹*get-level M1 L = 0*› ‹*count-decided M1 = 0*› **for** *L*
using *count-decided-ge-get-level*[*of M1 L*] ‹*count-decided M = 1*›
unfolding *M* **by** *auto*

have *K-M*: ⟨$K \in$ *lits-of-l M*⟩
  using *M'* **by** *simp*

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

have ⟨¬*twl-is-an-exception C* {#*K*#} {#} ⟹ *twl-lazy-update* (*Propagated* (− *K*) {#− *K*#} # *M1*) *C*⟩ **if** ⟨*C*∈#*N* + *U*⟩ **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 ⟨*watched-literals-false-of-max-level* (*Propagated* (− *K*) {#− *K*#} # *M1*) *C*⟩ **for** *C*
  by (*cases C*) (*auto simp*: *get-level-cons-if*)
ultimately have ⟨*twl-st-inv ?S*⟩
  using *st-invs-M1' wf-N-U* **by** (*auto simp*: *twl-st-inv.simps*
   simp del: *set-mset-union*)
moreover have ⟨*valid-enqueued ?S*⟩
  by *auto*
moreover have *struct-invs-S*: ⟨*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*state*$_W$*-of ?S*)⟩
  using *struct-invs-add propa*
  by (*auto dest*!: *cdcl*$_W$*-restart-mset.propagate cdcl*$_W$*-restart-mset.cdcl*$_W$*-cdcl*$_W$*-restart*
   simp: *intro*: *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-inv*)
moreover have ⟨*cdcl*$_W$*-restart-mset.no-smaller-propa* (*state*$_W$*-of ?S*)⟩
  using *no-smaller-add propa struct-invs-add*
  by (*auto 5 5 simp*: *dest*!: *cdcl*$_W$*-restart-mset.propagate cdcl*$_W$*-restart-mset.cdcl*$_W$*-stgy.propagate'*
   intro: *cdcl*$_W$*-restart-mset.cdcl*$_W$*-stgy-no-smaller-propa*)
moreover have ⟨*twl-st-exception-inv ?S*⟩
  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 ⟨*no-duplicate-queued ?S*⟩
  by *auto*
moreover have ⟨*distinct-queued ?S*⟩
  by *auto*
moreover have ⟨*confl-cands-enqueued ?S*⟩
  unfolding *confl-cands-enqueued.simps Ball-def*
proof (*intro impI allI*)
  fix *C*
  assume
   C: ⟨*C* ∈# *N* + *U*⟩ **and**

```
    H: ⟨Propagated (− K) (DECO-clause M) # M1 |=as CNot (clause C)⟩
  obtain L1 L2 UW where
     C′: ⟨C = TWL-Clause {#L1, L2#} UW⟩ and dist-C: ⟨distinct-mset (clause C)⟩
    using wf-N-U[OF C]
    apply (cases C)
    by (auto simp: twl-exception-inv.simps size-2-iff cdcl_W-restart-mset-state)
  have M1-C: ⟨¬M1 |=as CNot (clause C)⟩
    using confl-enqueued-M1 C by auto
  define C′ where ⟨C′ = remove1-mset K (clause C)⟩
  then have C-K-C′: ⟨clause C = add-mset K C′⟩ and ⟨K ∉# C′⟩ and
    M1-C′: ⟨M1 |=as CNot C′⟩ and K-C′-C: ⟨add-mset K C′ = clause C⟩
    using dist-C M1-C H by (auto simp: true-annots-true-cls-def-iff-negation-in-model
        dest: in-diffD dest!: multi-member-split)
  have ⟨C′ + {#K#} ∈# clauses (N+U)⟩
    using C M1-C′
    by (auto simp: K-C′-C M′)
  then have ⟨undefined-lit M1 K ⟹ ¬ M1 |=as CNot C′⟩
    using no-smaller
    unfolding cdcl_W-restart-mset.no-smaller-propa-def state_W-of.simps cdcl_W-restart-mset-state
      clauses-def image-mset-union M′ union-iff
    by blast
  then have False
    using no-dup M1-C′ unfolding M′
    by (auto simp: cdcl_W-restart-mset-state clauses-def M′)
  then show ⟨(∃L′. L′ ∈# watched C ∧ L′ ∈# {#K#}) ∨ (∃L. (L, C) ∈# {#})⟩
    by fast
qed
moreover have ⟨propa-cands-enqueued ?S⟩
  unfolding propa-cands-enqueued.simps Ball-def
proof (intro impI allI)
  fix C L
  assume
    C: ⟨C ∈# N + U⟩ and
    L: ⟨L ∈# clause C⟩ and
    H: ⟨Propagated (− K) (DECO-clause M) # M1 |=as CNot (remove1-mset L (clause C))⟩ and
    undef: ⟨undefined-lit (Propagated (− K) (DECO-clause M) # M1) L⟩
  obtain L1 L2 UW where
     C′: ⟨C = TWL-Clause {#L1, L2#} UW⟩ and dist-C: ⟨distinct-mset (clause C)⟩
    using wf-N-U[OF C]
    apply (cases C)
    by (auto simp: twl-exception-inv.simps size-2-iff cdcl_W-restart-mset-state)
  have M1-C: ⟨¬M1 |=as CNot (remove1-mset L (clause C))⟩
    using propa-enqueued-M1 C undef L by auto
  define C′ where ⟨C′ = remove1-mset K (remove1-mset L (clause C))⟩
  then have C-K-C′: ⟨clause C = add-mset K (add-mset L C′)⟩ and ⟨K ∉# C′⟩ and
    M1-C′: ⟨M1 |=as CNot C′⟩ and K-C′-C: ⟨add-mset K (add-mset L C′) = clause C⟩ and
    K-C′-C′: ⟨add-mset K C′ = remove1-mset L (clause C)⟩
    using dist-C M1-C H L by (auto simp: true-annots-true-cls-def-iff-negation-in-model
        dest: in-diffD dest!: multi-member-split)
  have eq2: ⟨{#L1, L2#} = {#L, L′#} ⟷ L = L1 ∧ L′ = L2 ∨ L = L2 ∧ L′ = L1⟩ for L L′
    by (auto simp: add-mset-eq-add-mset)
  have ⟨twl-exception-inv (M1, N, U, None, NP, UP, {#}, {#}) C⟩
    using past C unfolding past-invs.simps M′
    by fast
  moreover have ⟨L2 ∉ lits-of-l M1⟩
```

```
      using H no-dup undef dist-C
      unfolding true-annots-true-cls-def-iff-negation-in-model M′ C′ Ball-def
      by (cases ‹L = L1›; cases ‹L = L2›;
          auto dest: in-lits-of-l-defined-litD no-dup-appendD no-dup-consistentD
          simp: all-conj-distrib)+
    moreover have ‹L1 ∉ lits-of-l M1›
      using H no-dup undef dist-C
      unfolding true-annots-true-cls-def-iff-negation-in-model M′ C′ Ball-def
      apply (cases ‹L = L1›; cases ‹L = L2›)
      by (auto dest: in-lits-of-l-defined-litD no-dup-appendD no-dup-consistentD
          simp: all-conj-distrib)
    moreover {
      have ‹L′ ∈ lits-of-l M1 ⟹ L′ ∈# UW ⟹ False› for L′
        using H no-dup undef dist-C ‹L1 ∉ lits-of-l M1› ‹L2 ∉ lits-of-l M1› n-d1
        unfolding true-annots-true-cls-def-iff-negation-in-model M′ C′ Ball-def
        apply (cases ‹L = L1›; cases ‹L = L2›)
        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 ‹¬ has-blit M1 (clause (TWL-Clause {#L1, L2#} UW)) L1› and
        ‹¬ has-blit M1 (clause (TWL-Clause {#L1, L2#} UW)) L2›
        using ‹L1 ∉ lits-of-l M1› ‹L2 ∉ lits-of-l M1›
        unfolding has-blit-def
        by auto
    }
    ultimately have
       ‹− L1 ∈ lits-of-l M1 ⟹ (∀K∈#UW. − K ∈ lits-of-l M1)›
       ‹− L2 ∈ lits-of-l M1 ⟹ (∀K∈#UW. − K ∈ lits-of-l M1)›
      unfolding C′ twl-exception-inv.simps twl-clause.sel eq2
      by fastforce+
    moreover have ‹L1 ≠ L2›
      using dist-C by (auto simp: C′)
    ultimately have ‹K ≠ L1 ⟹ K ≠ L2 ⟹ False›
      using M1-C′ L undef K-C′-C no-dup[unfolded M′]
      by (cases ‹− L1 ∈ lits-of-l M1›; cases ‹− L2 ∈ lits-of-l M1›;
          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 ‹(∃L′. L′ ∈# watched C ∧ L′ ∈# {#K#}) ∨ (∃L. (L, C) ∈# {#})›
      by (auto simp: C′)
  qed
  moreover have ‹get-conflict ?S = None›
    by simp
  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)
    then have ‹entailed-clss-inv ?S›
      using entailed unfolding entailed-clss-inv.simps by (auto 5 5 simp: get-level-cons-if)
  }
  moreover {
    have ‹¬clauses-to-update-prop {#} (M1) (L, La) ⟹
        clauses-to-update-prop {#K#} (Propagated (− K) {#− K#} # M1) (L, La) ⟹ False› 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*: ‹*TWL-DECO-clause M* = *TWL-Clause* {#−*K1*, −*K2*#} (*uminus* '# *lit-of* '# *mset C*)›
  by (*auto simp*: *TWL-DECO-clause-def*)

let *?S* = ‹(*M1*, *add-mset* (*TWL-DECO-clause M*) *N*, *U*, *None*, *NP*, *UP*, {#}, {#})›

have *struct-invs-S*: ‹$cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv* ($state_W$*-of ?S*)›
  using *struct-invs-add* **by** *auto*

have *no-smaller-S*: ‹$cdcl_W$*-restart-mset.no-smaller-propa* ($state_W$*-of ?S*)›
  using *no-smaller-add* **by** *simp*

obtain *MM3 MM2 MM1* **where** *MM*: ‹*M* = *MM3* @ *Decided K1* # *MM2* @ *Decided K2* # *MM1*› **and**
  [*simp*]: ‹*filter is-decided MM3* = []› **and**
  [*simp*]: ‹*filter is-decided MM2* = []›
  using *filter-M*
  by (*auto simp*: *filter-eq-Cons-iff filter-empty-conv*
    eq-commute[*of - ‹filter is-decided -›*])
then have [*simp*]: ‹*count-decided MM3* = *0*› ‹*count-decided MM2* = *0*›
  by (*auto simp*: *count-decided-0-iff filter-empty-conv*
    simp del: ‹*filter is-decided MM3* = []› ‹*filter is-decided MM2* = []›)
have [*simp*]: ‹*get-level M K* = *Suc* (*count-decided M1*)›
  using *no-dup* **unfolding** *M′*
  by (*auto simp*: *get-level-skip*)
then have [*iff*]: ‹*K1*≠*K*›
  using *lev-K no-dup* **by** (*auto simp*: *MM simp del*: ‹*get-level M K* = *Suc* (*count-decided M1*)›)
have ‹*set M1* ⊆ *set MM1*›
  using *refl*[*of M*] *lev-K no-dup*[*unfolded MM*] *no-dup*[*unfolded M′*] ‹*count-decided MM2* = *0*›
   ‹*count-decided MM3* = *0*›
  apply (*subst* (*asm*) *M′*)
  apply (*subst* (*asm*) *MM*)
  by (*auto simp*: *simp del*: ‹*count-decided MM2* = *0*› ‹*count-decided MM3* = *0*›
    elim!: *list-match-lel-lel*)
then have ‹*undefined-lit MM1 L* ⟹ *undefined-lit M1 L*› **for** *L*
  by (*auto simp*: *Decided-Propagated-in-iff-in-lits-of-l*)
then have [*iff*]: ‹*K1* ∉ *lits-of-l M1*› ‹*K2* ∉ *lits-of-l M1*›
  using *no-dup* **unfolding** *MM*
  by (*auto dest*: *in-lits-of-l-defined-litD*)

have ‹*struct-wf-twl-cls* (*TWL-DECO-clause M*)›
  using *dist-filtered′* **unfolding** *deco-M filter-M*
  by (*auto simp*: *simp del*: *clause-TWL-Deco-clause*)
moreover have ‹*twl-lazy-update M1* (*TWL-DECO-clause M*)›
  by (*auto simp*: *deco-M add-mset-eq-add-mset*)
moreover have ‹*watched-literals-false-of-max-level M1* (*TWL-DECO-clause M*)›
  by (*auto simp*: *deco-M add-mset-eq-add-mset*)
ultimately have ‹*twl-st-inv ?S*›
  using *wf-N-U st-invs-M1′* **unfolding** *twl-st-inv.simps*
  by (*auto simp*: *twl-is-an-exception-def*)
moreover have ‹*valid-enqueued ?S*›
  by *auto*
moreover have *struct-invs-S*: ‹$cdcl_W$*-restart-mset*.$cdcl_W$*-all-struct-inv* ($state_W$*-of ?S*)›
  using *struct-invs-add* **by** *simp*
moreover have ‹$cdcl_W$*-restart-mset.no-smaller-propa* ($state_W$*-of ?S*)›
  using *no-smaller-add* **by** *simp*

moreover have ‹*twl-st-exception-inv ?S*›
  using *st-invs-M1'* **by** (*auto simp*: *twl-exception-inv.simps deco-M add-mset-eq-add-mset*)
moreover have ‹*no-duplicate-queued ?S*›
  by *auto*
moreover have ‹*distinct-queued ?S*›
  by *auto*
moreover have ‹*confl-cands-enqueued ?S*›
  using *confl-enqueued-M1* **by** (*auto simp*: *deco-M*)
moreover have ‹*propa-cands-enqueued ?S*›
  using *propa-enqueued-M1*
  by (*auto simp*: *deco-M true-annots-true-cls-def-iff-negation-in-model Ball-def*
      dest: *in-lits-of-l-defined-litD in-diffD*)
moreover have ‹*get-conflict ?S = None*›
  by *simp*
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-prop* {#} *M1* (*L*, *TWL-DECO-clause M*)› **for** *L*
    by (*auto simp*: *deco-M clauses-to-update-prop.simps add-mset-eq-add-mset*)
  moreover have ‹ *watched* (*TWL-DECO-clause M*) = {#*L*, *L'*#} ⟹
  − *L* ∈ *lits-of-l M1* ⟹ *False*› **for** *L L'*
    by (*auto simp*: *deco-M add-mset-eq-add-mset*)
  ultimately have ‹*clauses-to-update-inv ?S*›
    using *clss-upd no-dup* **by** (*auto simp*: *filter-mset-empty-conv clauses-to-update-prop.simps*
      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 M1a*
    by *fast+*
then have [*iff*]: ‹*K1* ∉ *lits-of-l M1a*› ‹*K2* ∉ *lits-of-l M1a*›
  using ‹*K1* ∉ *lits-of-l M1*› ‹*K2* ∉ *lits-of-l M1*› **unfolding** *M1a*
  by (*auto dest*: *in-lits-of-l-defined-litD*)
  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∈#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 clss-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' cdcl_W-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 ‹*fst* ((*hd o tl*) (*get-all-ann-decomposition M*))›; *cases a*)
(*auto simp*: *count-decided-0-iff*)
ultimately have ‹*get-all-ann-decomposition M* = [(*Decided K* # *M1*, *M2*), ([], *M1*)]›
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*]
in-get-all-ann-decomposition-decided-or-empty[*of* ‹*fst* ((*hd o tl*) (*get-all-ann-decomposition M*))›
‹*snd* ((*hd o tl*) (*get-all-ann-decomposition M*))› *M*] *count-dec-M1*
by (*cases* ‹*get-all-ann-decomposition M*›; *cases* ‹*tl* (*get-all-ann-decomposition M*)›;
cases ‹*fst* ((*hd o tl*) (*get-all-ann-decomposition M*))›)
(*auto simp*: *count-decided-0-iff*)

show ‹*?thesis*›
by (*simp add*: ‹*get-all-ann-decomposition M* = [(*Decided K* # *M1*, *M2*), ([], *M1*)]›
get-all-ann-decomposition-decomp)
qed

lemma *negate-model-and-add-twl-twl-stgy-invs*:
assumes
‹*negate-model-and-add-twl S T*› **and**
‹*twl-struct-invs S*› **and**
‹*twl-stgy-invs S*›
shows ‹*twl-stgy-invs T*›
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* = ‹(*M*, *N*, *U*, *None*, *NP*, *UP*, *WS*, *Q*)›
let *?T* = ‹(*Propagated* (− *K*) (*DECO-clause M*) # *M1*, *N*, *U*, *None*, *add-mset* (*DECO-clause M*)
NP, *UP*,
{#}, {#*K*#})›
have
false-with-lev: ‹$cdcl_W$*-restart-mset.conflict-is-false-with-level* ($state_W$*-of ?S*)› **and**
no-smaller-confl: ‹$cdcl_W$*-restart-mset.no-smaller-confl* ($state_W$*-of ?S*)› **and**
confl0: ‹$cdcl_W$*-restart-mset.conflict-non-zero-unless-level-0* ($state_W$*-of ?S*)›
using *stgy* **unfolding** *twl-stgy-invs-def* $cdcl_W$*-restart-mset.*$cdcl_W$*-stgy-invariant-def*
by *fast+*
have *M*: ‹*M* = *M2* @ *Decided K* # *M1*›
using *decomp count-dec* **by**(*simp add*: *get-all-ann-decomposition-count-decided-1*)
have [*iff*]: ‹*M* = *M′* @ *Decided K′* # *Ma* ⟷ *M′* = *M2* ∧ *K′* = *K* ∧ *Ma* = *M1*› **for** *M′ K′ Ma*
using *count-dec* **unfolding** *M*
by (*auto elim*!: *list-match-lel-lel*)
have [*iff*]: ‹*M1* = *M′* @ *Decided K′* # *Ma* ⟷ *False*› **for** *M′ K′ Ma*
using *count-dec* **unfolding** *M*
by (*auto elim*!: *list-match-lel-lel*)
have
false-with-lev: ‹$cdcl_W$*-restart-mset.conflict-is-false-with-level* ($state_W$*-of ?T*)›
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 ‹$cdcl_W$*-restart-mset.no-smaller-confl* ($state_W$*-of ?T*)›
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 ‹$cdcl_W$*-restart-mset.conflict-non-zero-unless-level-0* ($state_W$*-of ?T*)›
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 (*auto simp*: *cdcl*$_W$*-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: ‹*cdcl*$_W$*-restart-mset.conflict-is-false-with-level* (*state*$_W$*-of ?S*)› **and**
no-smaller-confl: ‹*cdcl*$_W$*-restart-mset.no-smaller-confl* (*state*$_W$*-of ?S*)› **and**
confl0: ‹*cdcl*$_W$*-restart-mset.conflict-non-zero-unless-level-0* (*state*$_W$*-of ?S*)›
using *stgy* **unfolding** *twl-stgy-invs-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-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 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*: ‹*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: ‹*cdcl*$_W$*-restart-mset.conflict-is-false-with-level* (*state*$_W$*-of ?T*)›
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 ‹*cdcl*$_W$*-restart-mset.no-smaller-confl* (*state*$_W$*-of ?T*)›
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 ‹*cdcl*$_W$*-restart-mset.conflict-non-zero-unless-level-0* (*state*$_W$*-of ?T*)›
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*
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: ‹*cdcl*$_W$*-restart-mset.conflict-is-false-with-level* (*state*$_W$*-of ?S*)› **and**
no-smaller-confl: ‹*cdcl*$_W$*-restart-mset.no-smaller-confl* (*state*$_W$*-of ?S*)› **and**
confl0: ‹*cdcl*$_W$*-restart-mset.conflict-non-zero-unless-level-0* (*state*$_W$*-of ?S*)›
using *stgy* **unfolding** *twl-stgy-invs-def cdcl*$_W$*-restart-mset.cdcl*$_W$*-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* $cdcl_W$*-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*: ‹*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: ‹$cdcl_W$*-restart-mset.conflict-is-false-with-level* ($state_W$*-of ?T*)›
  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 ‹$cdcl_W$*-restart-mset.no-smaller-confl* ($state_W$*-of ?T*)›
  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 ‹$cdcl_W$*-restart-mset.conflict-non-zero-unless-level-0* ($state_W$*-of ?T*)›
  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-twl-stgy-*$cdcl_W$*-learned-clauses-entailed-by-init*:
  assumes
    ‹*cdcl-twl-stgy S s*› **and**
    ‹*twl-struct-invs S*› **and**
    ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init* ($state_W$*-of S*)›
  shows
    ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init* ($state_W$*-of s*)›
  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-twl-stgy-$cdcl_W$*-stgy twl-struct-invs-def*)

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

lemma *negate-model-and-add-twl-*$cdcl_W$*-learned-clauses-entailed-by-init*:
  assumes
    ‹*negate-model-and-add-twl S s*› **and**

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

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* ∧
$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init* ($state_W$*-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*)).
mod-restriction (*get-all-init-clss S*) *N* ∧
(*get-conflict S* = *None* ⟶ *M* ≠ *None* ∧ *lit-of* '# *mset* (*get-trail S*) = *mset* (*the M*)) ∧
(*get-conflict S* ≠ *None* ⟶ *M* = *None*) ∧
atms-of-mm (*get-all-init-clss S*) = *atms-of-mm N* ∧
(*get-conflict S* = *None* ⟶ *next-model* (*map lit-of* (*get-trail S*)) *N*) ∧
$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init* ($state_W$*-of S*) ∧
cdcl-twl-enum-inv S}⟩

definition *enum-model-st* :: ⟨((*bool* × *'v twl-st*) × (*'v literal list option* × *'v clauses*)) *set*⟩ **where**
⟨*enum-model-st* = {((*b*, *S*), (*M*, *N*)).
mod-restriction (*get-all-init-clss S*) *N* ∧
(*b* ⟶ *get-conflict S* = *None* ∧ *M* ≠ *None* ∧ *lits-of-l* (*get-trail S*) = *set* (*the M*)) ∧
(*get-conflict S* ≠ *None* ⟶ ¬*b* ∧ *M* = *None*)}⟩

fun *add-to-init-cls* :: ⟨*'v twl-cls* ⇒ *'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
⟨*cdcl-twl-stgy*** *S T*⟩ **and**
final: ⟨*final-twl-state T*⟩ **and**
⟨*twl-struct-invs S*⟩ **and**
⟨*twl-stgy-invs S*⟩ **and**
ent: ⟨*cdcl*$_W$*-restart-mset.cdcl*$_W$*-learned-clauses-entailed-by-init* (*state*$_W$*-of S*)⟩ **and**
Hunsat: ⟨*get-conflict T* ≠ *None* ⟹ *unsatisfiable* (*set-mset* (*get-all-init-clss S*)) ⟹ *P*⟩ **and**
Hsat: ⟨*get-conflict T* = *None* ⟹ *consistent-interp* (*lits-of-l* (*get-trail T*)) ⟹
no-dup (*get-trail T*) ⟹ *atm-of* ‘ (*lits-of-l* (*get-trail T*)) ⊆ *atms-of-mm* (*get-all-init-clss T*) ⟹
get-trail T ⊨*asm get-all-init-clss S* ⟹ *satisfiable* (*set-mset* (*get-all-init-clss S*)) ⟹ *P*⟩
shows *P*
proof −
have ⟨*cdcl*$_W$*-restart-mset.cdcl*$_W$*-stgy*** (*state*$_W$*-of S*) (*state*$_W$*-of T*)⟩
by (*simp add*: *assms*(*1*) *assms*(*3*) *rtranclp-cdcl-twl-stgy-cdcl*$_W$*-stgy*)
have *all-struct-T*: ⟨*cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv* (*state*$_W$*-of T*)⟩
using *assms*(*1*) *assms*(*3*) *rtranclp-cdcl-twl-stgy-twl-struct-invs twl-struct-invs-def* **by** *blast*
then have
M-lev: ⟨*cdcl*$_W$*-restart-mset.cdcl*$_W$*-M-level-inv* (*state*$_W$*-of T*)⟩ **and**
alien: ⟨*cdcl*$_W$*-restart-mset.no-strange-atm* (*state*$_W$*-of T*)⟩
unfolding *cdcl*$_W$*-restart-mset.cdcl*$_W$*-all-struct-inv-def* **by** *fast*+

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

```
      subgoal using sat by auto
      subgoal using M-lev unfolding cdcl_W-restart-mset.cdcl_W-M-level-inv-def
        by (auto simp: twl-st)
      subgoal
        using tr-T M-lev unfolding cdcl_W-restart-mset.cdcl_W-M-level-inv-def by (auto simp: twl-st)
      subgoal using alien unfolding cdcl_W-restart-mset.no-strange-atm-def by (auto simp: twl-st)
      subgoal using tr-T by auto
      subgoal using tr-T M-lev unfolding cdcl_W-restart-mset.cdcl_W-M-level-inv-def
        by (auto simp: satisfiable-carac[symmetric] twl-st true-annots-true-cls)
      done
  next
    case unsat
    have ‹no-step cdcl_W-restart-mset.cdcl_W-stgy (state_W-of T)›
      using assms(1) assms(3) no-step-cdcl-twl-stgy-no-step-cdcl_W-stgy
        rtranclp-cdcl-twl-stgy-twl-struct-invs unsat(1) by blast
    from cdcl_W-restart-mset.cdcl_W-stgy-final-state-conclusive2[OF this]
    have unsat': ‹unsatisfiable (set-mset (cdcl_W-restart-mset.clauses (state_W-of T)))›
      using unsat all-struct-T stgy-T
      unfolding cdcl_W-restart-mset.cdcl_W-all-struct-inv-def twl-stgy-invs-def
        cdcl_W-restart-mset.cdcl_W-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]: ‹cdcl_W-restart-mset.clauses (state_W-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 (cdcl_W-restart-mset.clauses (state_W-of T)))›
        using alien tot unfolding cdcl_W-restart-mset.no-strange-atm-def
        by (auto simp: cdcl_W-restart-mset-state total-over-m-alt-def twl-st)
      ultimately have ‹I ⊨sm cdcl_W-restart-mset.clauses (state_W-of T)›
        using ent' I cons unfolding cdcl_W-restart-mset.cdcl_W-learned-clauses-entailed-by-init-def
          true-clss-clss-def total-over-m-def
        by (auto simp: clauses-def cdcl_W-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 :: ‹'v literal set ⇒ bool›
begin

fun negate-model-and-add :: ‹'v literal list option × 'v clauses ⇒ - × 'v clauses› 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 ← $WHILE_T{}^{cdcl\text{-}twl\text{-}enum\text{-}inv}$
   (λ*S*. *get-conflict S* = *None* ∧ *count-decided*(*get-trail S*) > *0* ∧ ¬*P* (*lits-of-l* (*get-trail S*)))
   (λ*S*. *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 (λ*b*. ∃*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*) ∧
  $cdcl_W$*-restart-mset*.$cdcl_W$*-learned-clauses-entailed-by-init* ($state_W$*-of S*) ∧

atms-of-mm (*get-all-init-clss S*) = *atms-of-mm N*}›

lemma *atms-of-uminus-lit-of*[*simp*]: ‹*atms-of* {#− *lit-of x*. *x* ∈# *A*#} = *atms-of* (*lit-of* ‘# *A*)›
by (*auto simp*: *atms-of-def image-image*)

lemma *lit-of-mset-eq-mset-setD*[*dest*]:
‹*lit-of* ‘# *mset M* = *mset aa* ⟹ *set aa* = *lit-of* ‘ *set M*›
by (*metis set-image-mset set-mset-mset*)

lemma *mod-restriction-add-twice*[*simp*]:
‹*mod-restriction A* (*add-mset C* (*add-mset C N*)) ⟷ *mod-restriction A* (*add-mset C N*)›
by (*auto simp*: *mod-restriction-def*)

lemma
assumes
confl: ‹*get-conflict W* = *None*› **and**
count-dec: ‹*count-decided* (*get-trail W*) = *0*› **and**
enum-inv: ‹*cdcl-twl-enum-inv W*› **and**
mod-rest-U: ‹*mod-restriction* (*get-all-init-clss W*) *N*› **and**
atms-U-U': ‹*atms-of-mm* (*get-all-init-clss W*) = *atms-of-mm N*›
shows
final-level0-add-empty-clause:
‹*mod-restriction* (*add-mset* {#} (*get-all-init-clss W*))
(*add-mset* {#− *lit-of x*. *x* ∈# *mset* (*get-trail W*)#} *N*)› (**is** *?A*) **and**
final-level0-add-empty-clause-unsat:
‹*unsatisfiable* (*set-mset* (*add-mset* {#− *lit-of x*. *x* ∈# *mset* (*get-trail W*)#} *N*))› (**is** *?B*)
proof −
have [*simp*]: ‹*DECO-clause* (*get-trail W*) = {#}› **and**
[*simp*]: ‹{*unmark L* |*L*. *is-decided L* ∧ *L* ∈ *set* (*trail* (*state*$_W$-*of W*))} = {}›
using *count-dec* **by** (*auto simp*: *count-decided-0-iff DECO-clause-def*
filter-mset-empty-conv twl-st)
have *struct-W*: ‹*twl-struct-invs W*› **and**
ent-W: ‹*cdcl*$_W$-*restart-mset.cdcl*$_W$-*learned-clauses-entailed-by-init* (*state*$_W$-*of W*)›
using *enum-inv*
unfolding *cdcl-twl-enum-inv-def* **by** *blast+*
have ‹*cdcl*$_W$-*restart-mset.no-strange-atm* (*state*$_W$-*of W*)› **and**
decomp: ‹*all-decomposition-implies-m* (*cdcl*$_W$-*restart-mset.clauses* (*state*$_W$-*of W*))
(*get-all-ann-decomposition* (*trail* (*state*$_W$-*of W*)))›
using *struct-W* **unfolding** *twl-struct-invs-def cdcl*$_W$-*restart-mset.cdcl*$_W$-*all-struct-inv-def*
by *fast+*
have *alien-W*: ‹*cdcl*$_W$-*restart-mset.no-strange-atm* (*state*$_W$-*of W*)›
using *struct-W*
unfolding *twl-struct-invs-def cdcl*$_W$-*restart-mset.cdcl*$_W$-*all-struct-inv-def*
by *fast*
have *1*: ‹*set-mset* (*cdcl*$_W$-*restart-mset.clauses* (*state*$_W$-*of W*))⊨*ps*
unmark-l (*trail* (*state*$_W$-*of W*))›
using *all-decomposition-implies-propagated-lits-are-implied*[*OF decomp*]
by *simp*
then have *2*: ‹*set-mset* (*get-all-init-clss W*) ⊨*ps*
unmark-l (*trail* (*state*$_W$-*of W*))›
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*: ‹*M* ⊨ {#− *lit-of x*. *x* ∈# *mset* (*get-trail W*)#}› **and**

```
      M-U′: ‹M ⊨m N› and
      tot: ‹total-over-m M (set-mset N)› and
      cons: ‹consistent-interp M›
    for M
  proof −
    have ‹M ⊨sm get-all-init-clss W›
      using mod-rest-U M-U′ cons
      unfolding mod-restriction-def
      apply auto
      using tot apply blast+
      done
    moreover have ‹total-over-m M (set-mset (get-all-init-clss W) ∪
              unmark-l (trail (state_W-of W)))›
      using alien-W atms-U-U′ tot
      unfolding total-over-m-alt-def total-over-set-alt-def
        cdcl_W-restart-mset.no-strange-atm-def
      by (auto 5 5 dest: atms-of-DECO-clauseD simp: lits-of-def twl-st)
    ultimately have ‹M ⊨s unmark-l (trail (state_W-of W))›
      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-cls-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:
  ‹(cdcl-twl-enum, next-model-filtered-nres) ∈
    [λ(M, N). M = None]_f enum-mod-restriction-st-clss → ⟨bool-rel⟩nres-rel›
proof −
  define model-if-exists where
    ‹model-if-exists S ≡ λM.
      (if ∃M. next-model M (snd S)
       then (fst M ≠ None ∧ next-model (the (fst M)) (snd S) ∧ snd M = snd S)
       else (fst M = None ∧ M = S))›
  for S :: ‹- × ′v clauses›

  have ‹full (next-model-filtered P) S U ⟷
        (∃T. model-if-exists S T ∧ full (next-model-filtered P) (None, snd T) U)›
    (is ‹?A ⟷ ?B›)
    if ‹fst S = None›
    for S U
  proof
    assume ?A
    then consider
      (nss) ‹no-step (next-model-filtered P) S› |
      (s1) T where ‹(next-model-filtered P) S T› and ‹full (next-model-filtered P) T U›
      unfolding full-def
      by (metis converse-rtranclpE)
```

```
then show ?B
proof cases
  case nss
  then have SU: ‹S = U›
    using ‹?A›
    apply (subst (asm) no-step-full-iff-eq)
     apply assumption by simp
  have ‹model-if-exists S S› and ‹fst S = None›
    using nss no-step-next-model-filtered-next-model-iff[of ‹(-, snd S)›] that
    unfolding model-if-exists-def
    by (cases S; auto; fail)+
  moreover {
    have ‹no-step (next-model-filtered P) (None, snd S)›
      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-statisfiable
      apply (rule unsatisfiable-mono)
       defer
       apply assumption
      by (cases S; cases ‹fst S›) (auto intro: unsatisfiable-mono)
    then have ‹full (next-model-filtered P) (None, snd S) U›
      apply (subst no-step-full-iff-eq)
       apply assumption
      using SU ‹fst S = None›
      by (cases S) auto
  }
  ultimately show ?B
    by fast
next
  case (s1 T)
  obtain M where
    M: ‹next-model M (snd S)› and
    T: ‹T = (if P (set M) then (Some M, snd S)
        else (None, add-mset (image-mset uminus (mset M)) (snd S)))›
    using s1
    unfolding model-if-exists-def
    apply (cases T)
    apply (auto simp: next-model-filtered.simps)
    done
  let ?T = ‹((Some M, snd S))›
  have nm: ‹model-if-exists S ?T›
    using M T that unfolding model-if-exists-def
    by (cases S) auto
  moreover have ‹full (next-model-filtered P) (negate-model-and-add ?T) U›
    using s1(2) T
    by (auto split: if-splits)
  moreover have ‹next-model-filtered P (None, snd ?T) (negate-model-and-add (Some M, snd S))›
    using nm that by (cases S) (auto simp: next-model-filtered.simps model-if-exists-def
        split: if-splits)
  ultimately show ?B
  proof -
    have (None, snd (Some M, 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* (λ*T*. *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-all-learned-diff-learned 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*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init* ($state_W$*-of S*)›
using *S-MN* **unfolding** *enum-mod-restriction-st-clss-def* **by** *blast*
then have *ent-s*: ‹$cdcl_W$*-restart-mset.*$cdcl_W$*-learned-clauses-entailed-by-init* ($state_W$*-of s*)›
using *rtranclp-cdcl-twl-stgy-*$cdcl_W$*-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-twl-enum-inv-def*
by (*auto intro*: *rtranclp-cdcl-twl-stgy-twl-struct-invs*
rtranclp-cdcl-twl-stgy-twl-stgy-invs)
show *?thesis*
using *struct-S stgy-S ent*
proof (*rule cdcl-twl-stgy-final-twl-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-twl-stgy-cdcl*$_W$*-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-statisfiable*[*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 bexI*[*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-twl-stgy-cdcl$_W$*-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-statisfiable*[*symmetric*])
ultimately show *?thesis*
using *nm*
apply −

by (*rule bexI*[*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*: ‹*WHILE*$_T$$^{cdcl\text{-}twl\text{-}enum\text{-}inv}$
(λ*S*. *get-conflict S* = *None* ∧ *count-decided* (*get-trail S*) > *0* ∧
¬*P* (*lits-of-l* (*get-trail S*)))
(λ*S*. *SPEC* (*negate-model-and-add-twl S*) ≫=
conclusive-TWL-run) *T*
≤ *SPEC*
(λ*y*. ∃ *x*. (*y*, *x*) ∈ {(*y*, *x*).
(((*get-conflict y* ≠ *None* ∧ *fst x* = *None*) ∨
(*fst x* ≠ *None* ∧ *P* (*lits-of-l* (*get-trail y*))) ∧
(*y*, *x*) ∈ *enum-mod-restriction-st-clss-after*) ∨
(*get-conflict y* = *None* ∧ *count-decided* (*get-trail y*) = *0* ∧
¬*P* (*lits-of-l* (*get-trail y*)) ∧ *fst x* = *None* ∧
(*y*, (*None*, *remove1-mset* (*uminus* ‘# *lit-of* ‘# *mset* (*get-trail y*)) (*snd x*)))
∈ *enum-mod-restriction-st-clss-after*))
} ∧
full (*next-model-filtered P*) (*None*, *snd M*) *x*)›
(**is** ‹*WHILE*$_T$$^-$ *?Cond* - - ≤ *SPEC ?Spec*›
is ‹- ≤ *SPEC* (λ*y*. ∃ *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-twl*› **and**
M: ‹*M* ∈ *Collect* (*model-if-exists MN*)›
for *S T* :: ‹′*v twl-st*› **and** *MN M*
proof −
define *R* **where**
‹*R* = {(*T* :: ′*v twl-st*, *S* :: ′*v twl-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* (λ*N*. *card* (*all-models N*))} ∪
{(*T* :: ′*v twl-st*, *S* :: ′*v twl-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* (λ*N*. *card* (*all-models N*))› - ‹*get-all-init-clss*›])
auto

subgoal
by *auto*
done
define *I* **where** ‹*I s* = (∃ *x*. (*s*, *x*) ∈ *enum-mod-restriction-st-clss-after* ∧
(*next-model-filtered P*)** (*None*, *snd M*) (*negate-model-and-add x*) ∧
(*next-model-filtered P*)** (*None*, *snd M*) (*None*, *snd* (*negate-model-and-add x*)) ∧
(*get-conflict s* = *None* ⟶ *next-model* (*map lit-of* (*get-trail s*)) (*snd x*)) ∧
(*get-conflict s* ≠ *None* ⟶ *unsatisfiable* (*set-mset* (*snd x*))) ∧
final-twl-state s)› **for** *s*
let *?Q* = ‹λ*U V s*′. *cdcl-twl-enum-inv s*′ ∧ *final-twl-state s*′ ∧ *cdcl-twl-stgy*** *V s*′ ∧ (*s*′, *U*) ∈ *R*›
have
conc-run: ‹*conclusive-TWL-run V* ≤ *SPEC* (*?Q U V*)›
(**is** *?conc-run* **is** ‹- ≤ *SPEC ?Q*›) **and**
inv-I: ‹*?Q U V W* ⟹ *I W*› (**is** ‹- ⟹ *?I*›)
if
U: ‹*cdcl-twl-enum-inv U*› **and**
confl: ‹*?Cond U*› **and**
neg: ‹*negate-model-and-add-twl U V*› **and**
I-U: ‹*I U*›
for *U V W*
proof −
{
have ‹*clauses-to-update V* = {#}›
using *neg* **by** (*auto simp*: *negate-model-and-add-twl.simps*)
have
ent-V: ‹$cdcl_W$*-restart-mset*.$cdcl_W$*-learned-clauses-entailed-by-init* ($state_W$*-of V*)› **and**
struct-U: ‹*twl-struct-invs U*› **and**
ent-U: ‹$cdcl_W$*-restart-mset*.$cdcl_W$*-learned-clauses-entailed-by-init* ($state_W$*-of U*)›
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*: ‹*twl-struct-invs V*› ‹*twl-stgy-invs V*›
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*]: ‹*get-all-init-clss V* = *add-mset* (*DECO-clause* (*get-trail U*))(*get-all-init-clss U*)›
using *neg* **by** (*auto simp*: *negate-model-and-add-twl.simps*)

have *next-mod-U*: ‹*next-model* (*map lit-of* (*get-trail U*)) (*get-all-init-clss U*)›
if *None*: ‹*get-conflict U* = *None*›
proof (*rule cdcl-twl-stgy-final-twl-stateE*[*of U U*])
show ‹*cdcl-twl-stgy*** *U U*›
by *simp*
show ‹*final-twl-state U*› ‹*twl-struct-invs U*› ‹*twl-stgy-invs U*›
‹$cdcl_W$*-restart-mset*.$cdcl_W$*-learned-clauses-entailed-by-init* ($state_W$*-of U*)›
using *U* **unfolding** *cdcl-twl-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*))›

```
    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 ‹cdcl_W-restart-mset.no-strange-atm (state_W-of U)› and
  decomp: ‹all-decomposition-implies-m (cdcl_W-restart-mset.clauses (state_W-of U))
     (get-all-ann-decomposition (trail (state_W-of U)))›
  using struct-U unfolding twl-struct-invs-def cdcl_W-restart-mset.cdcl_W-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-twl-stgy-final-twl-stateE[of U U])
  show ‹cdcl-twl-stgy** U U›
    by simp
  show ‹final-twl-state U› ‹twl-struct-invs U› ‹twl-stgy-invs U›
    ‹cdcl_W-restart-mset.cdcl_W-learned-clauses-entailed-by-init (state_W-of U)›
    using U unfolding cdcl-twl-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
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-twl-stgy** V WW› and final: ‹final-twl-state WW›
  have ent-W: ‹cdcl_W-restart-mset.cdcl_W-learned-clauses-entailed-by-init (state_W-of WW)›
    using U ent-V neg invs-V rtranclp-cdcl-twl-stgy-cdcl_W-learned-clauses-entailed-by-init
      star
    unfolding cdcl-twl-enum-inv-def by blast
  then have H1: ‹cdcl-twl-enum-inv WW›
```

using *star final invs-V* **unfolding** *cdcl-twl-enum-inv-def*
using *rtranclp-cdcl-twl-stgy-twl-stgy-invs rtranclp-cdcl-twl-stgy-twl-struct-invs* **by** *blast*
have *init-clss-WW-V*[*simp*]: ⟨*get-all-init-clss WW* = *get-all-init-clss V*⟩
by (*metis rtranclp-cdcl-twl-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-twl-stgy-final-twl-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-cls 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-twl-stgy-final-twl-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-cls 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
twl-enum: ⟨*cdcl-twl-enum-inv W*⟩ **and**
final: ⟨*final-twl-state W*⟩ **and**
st: ⟨*cdcl-twl-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-clause* (*get-trail U*))›
by (*force simp*: *DECO-clause-def twl-st CNot-def*)
have *ent3-gnerealise*: ‹A ∪ B ∪ C ⊨ps D ⟹ A ⊨ps B ⟹ A ∪ C ⊨ps D› **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 ‹*set-mset* (*cdcl*$_W$*-restart-mset.clauses* (*state*$_W$*-of U*)) ∪
CNot (*DECO-clause* (*get-trail U*)) ⊨ps *unmark-l* (*trail* (*state*$_W$*-of U*))›
using *all-decomposition-implies-propagated-lits-are-implied*[*OF decomp*]
unfolding *1* **.**
then have *2*: ‹*set-mset* (*get-all-init-clss U*) ∪ *CNot* (*DECO-clause* (*get-trail U*)) ⊨ps
unmark-l (*trail* (*state*$_W$*-of U*))›
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*]: ‹*unmark-l* (*get-trail U*) = *CNot* {#− *lit-of x*. *x* ∈# *mset* (*get-trail U*)#}›
by (*force simp*: *CNot-def*)
have *mod-U*: ‹*mod-restriction* (*get-all-init-clss U*) (*snd U'*)› **and**
atms-U-U': ‹*atms-of-mm* (*get-all-init-clss U*) = *atms-of-mm* (*snd U'*)›
using *U-U' confl* **unfolding** *enum-mod-restriction-st-clss-after-def* **by** (*cases U'*; *auto*; *fail*)+
have *alien-U*: ‹*cdcl*$_W$*-restart-mset.no-strange-atm* (*state*$_W$*-of U*)›
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*: ‹*M* ⊨ *DECO-clause* (*get-trail U*)›
if
total: ‹*total-over-m M* (*set-mset* (*snd U'*))› **and**
consistent: ‹*consistent-interp M*› **and**
M-tr: ‹*M* ⊨ {#− *lit-of x*. *x* ∈# *mset* (*get-trail U*)#}› **and**
M-U': ‹*M* ⊨m *snd U'*›
for *M*
proof (*rule ccontr*)
assume ‹¬*?thesis*›
moreover have *tot-tr*: ‹*total-over-m M* {*DECO-clause* (*get-trail U*)}›
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 ‹*M* ⊨s *CNot* (*DECO-clause* (*get-trail U*))›
by (*simp add*: *total-not-true-cls-true-clss-CNot*)
moreover have ‹*M* ⊨sm *get-all-init-clss U*›
using *mod-U total consistent M-U'* **unfolding** *mod-restriction-def*
by *blast*
moreover have ‹*total-over-m M* (*set-mset* (*get-all-init-clss U*))›
using *total atms-U-U'* **by** (*simp add*: *total-over-m-def*)
moreover have ‹*total-over-m M* (*unmark-l* (*trail* (*state*$_W$*-of U*)))›
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 ‹*M* ⊨s *unmark-l* (*trail* (*state*$_W$*-of U*))›
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-twl-enum-inv W* ∧ *final-twl-state W* ∧ *cdcl-twl-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-twl-enum-inv-def mod-rest-U
dest: *atms-of-DECO-clauseD*)
defer
apply (*smt U atms-of-def cdcl-twl-enum-inv-def cdcl-twl-stgy-final-twl-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)
    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: ‹*I s*› **and**
cond: ‹¬ (*?Cond s*)› **and**
enum: ‹*cdcl-twl-enum-inv s*›
for *s*
proof −
obtain *x* **where**
sx: ‹(*s*, *x*) ∈ *enum-mod-restriction-st-clss-after*› **and**
st': ‹(*next-model-filtered P*)** (*None*, *snd M*) (*None*, *snd* (*negate-model-and-add x*))› **and**
st: ‹(*next-model-filtered P*)** (*None*, *snd M*) (*negate-model-and-add x*)› **and**
final: ‹*final-twl-state s*› **and**
nm: ‹*get-conflict s* = *None* ⟹ *next-model* (*map lit-of* (*get-trail s*)) (*snd x*)› **and**
unsat: ‹*get-conflict s* ≠ *None* ⟹ *unsatisfiable* (*set-mset* (*snd x*))›
using *I* **unfolding** *I-def* **by** *meson*
let *?x* = ‹*if get-conflict s* = *None*
then (*Some* (*map lit-of* (*get-trail s*)), *snd x*)
else (*None*, *snd x*)›
let *?y* = ‹*negate-model-and-add ?x*›
have *step*: ‹(*next-model-filtered P*) (*None*, *snd* (*negate-model-and-add x*)) *?y*›
if ‹*get-conflict s* = *None*› **and** ‹*P* (*lits-of-l* (*get-trail s*))›
using *cond that sx final nm* **unfolding** *enum-mod-restriction-st-clss-after-def*
enum-model-st-def
by (*cases x*; *cases* ‹*fst x*›)
(*auto simp*: *next-model-filtered.simps lits-of-def*
conclusive-TWL-run-def conc-fun-RES
intro!: *exI*[*of* - ‹*map lit-of* (*get-trail s*)›])
moreover have *step*: ‹(*next-model-filtered P*)** (*negate-model-and-add x*) *?y*›
if ‹*get-conflict s* ≠ *None*›
using *cond that sx* **unfolding** *enum-mod-restriction-st-clss-after-def*
enum-model-st-def
by (*cases x*; *cases* ‹*fst x*›)
(*auto simp*: *next-model-filtered.simps lits-of-def*)
moreover have *step*: ‹(*next-model-filtered P*) (*negate-model-and-add x*) *?y* ∨
(*negate-model-and-add x*) = *?y*›
if ‹*get-conflict s* = *None*› **and** ‹¬*P* (*lits-of-l* (*get-trail s*))›
using *cond that sx nm* **unfolding** *enum-mod-restriction-st-clss-after-def*
enum-model-st-def
apply (*cases x*; *cases* ‹*fst x*›)
by (*auto simp*: *next-model-filtered.simps lits-of-def*
conclusive-TWL-run-def conc-fun-RES
intro!: *exI*[*of* - ‹*map lit-of* (*get-trail s*)›])
ultimately have *st*: ‹(*next-model-filtered P*)** (*None*, *snd M*) *?y*›
using *st st'* **by** *force*
have *1*: ‹(*s*, *?x*) ∈ *enum-mod-restriction-st-clss-after*›
if ‹*count-decided* (*get-trail s*) ≠ *0* ∨ *get-conflict s* ≠ *None* ∨ *P* (*lit-of* ‘ *set* (*get-trail s*))›
using *sx cond nm that* **unfolding** *enum-mod-restriction-st-clss-after-def*
enum-model-st-def
by (*cases x*; *cases* ‹*fst x*›) (*auto simp*: *lits-of-def*)
have *unsat'*: ‹*unsatisfiable* (*set-mset* (*add-mset* {#− *lit-of x*. *x* ∈# *mset* (*get-trail s*)#} (*snd x*)))›
if ‹*get-conflict s* = *None*› **and** ‹*count-decided* (*get-trail s*) = *0*› **and**
‹¬*P* (*lit-of* ‘ *set* (*get-trail s*))›
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* ‹*fst x*›)
(*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-twl-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-twl-enum-def
  apply (intro frefI 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*: ‹$(S, S') \in$ *convert-lits-l M N* $\Longrightarrow$
map ($\lambda x.$ $-$*lit-of x*) (*filter is-decided S'*) = *map* ($\lambda x.$ $-$*lit-of x*) (*filter is-decided S*)›
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*]:
‹$(S, S') \in$ *convert-lits-l M N* $\Longrightarrow$ *DECO-clause S'* = *DECO-clause S*›
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*]:
‹$(S, S') \in$ *convert-lits-l M N* $\Longrightarrow$ *TWL-DECO-clause S'* = *TWL-DECO-clause S*›
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*]:
‹$(S, S') \in$ *twl-st-l b* $\Longrightarrow$ *DECO-clause* (*get-trail S'*) = *DECO-clause* (*get-trail-l S*)›
by (*auto simp*: *twl-st-l-def convert-lits-l-DECO-clause*)

lemma [*twl-st-l*]:
‹$(S, S') \in$ *twl-st-l b* $\Longrightarrow$ *TWL-DECO-clause* (*get-trail S'*) = *TWL-DECO-clause* (*get-trail-l S*)›
by (*auto simp*: *twl-st-l-def convert-lits-l-DECO-clause*)

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

definition *find-decomp-target* :: ‹*nat* ⇒ *'v twl-st-l* ⇒ (*'v twl-st-l* × *'v literal*) *nres*› **where**
‹*find-decomp-target* = (λi *S*.
SPEC(λ(*T*, *K*). $\exists$ *M2 M1*. *equality-except-trail S T* $\wedge$ *get-trail-l T* = *M1* $\wedge$
(*Decided K* # *M1*, *M2*) $\in$ *set* (*get-all-ann-decomposition* (*get-trail-l S*)) $\wedge$
get-level (*get-trail-l S*) *K* = *i*))›

```
fun propagate-unit-and-add :: ‹'v literal ⇒ 'v twl-st ⇒ 'v twl-st› where
  ‹propagate-unit-and-add K (M, N, U, D, NE, UE, WS, Q) =
      (Propagated (−K) {#−K#} # M, N, U, None, add-mset {#−K#} NE, UE, {#}, {#K#})›

fun propagate-unit-and-add-l :: ‹'v literal ⇒ 'v twl-st-l ⇒ 'v twl-st-l› where
  ‹propagate-unit-and-add-l K (M, N, D, NE, UE, WS, Q) =
      (Propagated (−K) 0 # M, N, None, add-mset {#−K#} NE, UE, {#}, {#K#})›

definition negate-mode-bj-unit-l-inv :: ‹'v twl-st-l ⇒ bool› where
  ‹negate-mode-bj-unit-l-inv S ⟷
     (∃(S'::'v twl-st) b. (S, S') ∈ twl-st-l b ∧ twl-list-invs S ∧ twl-stgy-invs S' ∧
        twl-struct-invs S' ∧ get-conflict-l S = None)›

definition negate-mode-bj-unit-l   :: ‹'v twl-st-l ⇒ 'v twl-st-l nres› where
‹negate-mode-bj-unit-l = (λS. do {
    ASSERT(negate-mode-bj-unit-l-inv S);
    (S, K) ← find-decomp-target 1 S;
    RETURN (propagate-unit-and-add-l K S)
  })›


lemma negate-mode-bj-unit-l:
  fixes S :: ‹'v twl-st-l› and S' :: ‹'v twl-st›
  assumes ‹count-decided (get-trail-l S) = 1› and
    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›
  shows
    ‹negate-mode-bj-unit-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'))›
proof −
    have H: ‹∃y∈Collect (negate-model-and-add-twl S').
            (propagate-unit-and-add-l x2 x1, y)
            ∈ {(S, S''). (S, S'') ∈ twl-st-l None ∧ twl-list-invs S ∧ clauses-to-update-l S = {#}}›
    if
        count-dec: ‹count-decided (get-trail-l S) = 1› and
        S-S': ‹(S, S') ∈ twl-st-l b› and
        ‹twl-struct-invs S'› and
        ‹twl-list-invs S› and
        ‹twl-stgy-invs S'› and
        x-S: ‹x ∈ {(T, K).
            ∃M2 M1.
                equality-except-trail S T ∧
                get-trail-l T = M1 ∧
                (Decided K # M1, M2)
                ∈ set (get-all-ann-decomposition (get-trail-l S)) ∧
                get-level (get-trail-l S) K = 1}› and
        x: ‹x = (x1, x2)›
    for x :: ‹'v twl-st-l × 'v literal› and x1 :: ‹'v twl-st-l› and x2 :: ‹'v literal›
    proof −
      let ?y0 = ‹(λ(M, Oth). (drop (length M − length (get-trail-l x1)) (get-trail S'), Oth)) S'›
      let ?y1 = ‹propagate-unit-and-add x2 ?y0›
```

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

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


definition DECO-clause-l :: ‹('v, 'a) ann-lits ⇒ 'v clause-l› where
  ‹DECO-clause-l M = map (uminus o lit-of) (filter is-decided M)›
```

fun *propagate-nonunit-and-add* :: ‹′v literal ⇒ ′v literal multiset twl-clause ⇒ ′v twl-st ⇒ ′v twl-st›
where
‹propagate-nonunit-and-add K C (M, N, U, D, NE, UE, WS, Q) = do {
(Propagated (−K) (clause C) # M, add-mset C N, U, None,
NE, UE, {#}, {#K#})
}›

fun *propagate-nonunit-and-add-l* :: ‹′v literal ⇒ ′v clause-l ⇒ nat ⇒ ′v twl-st-l ⇒ ′v twl-st-l› **where**
‹propagate-nonunit-and-add-l K C i (M, N, D, NE, UE, WS, Q) = do {
(Propagated (−K) i # M, fmupd i (C, True) N, None,
NE, UE, {#}, {#K#})
}›

definition *negate-mode-bj-nonunit-l-inv* **where**
‹negate-mode-bj-nonunit-l-inv S ⟷
(∃ S′′ b. (S, S′′) ∈ twl-st-l b ∧ twl-list-invs S ∧ count-decided (get-trail-l S) > 1 ∧
twl-struct-invs S′′ ∧ twl-stgy-invs S′′ ∧ get-conflict-l S = None)›

definition *negate-mode-bj-nonunit-l* :: ‹′v twl-st-l ⇒ ′v twl-st-l nres› **where**
‹negate-mode-bj-nonunit-l = (λS. do {
ASSERT(negate-mode-bj-nonunit-l-inv S);
let C = DECO-clause-l (get-trail-l S);
(S, K) ← find-decomp-target (count-decided (get-trail-l S)) S;
i ← get-fresh-index (get-clauses-l S);
RETURN (propagate-nonunit-and-add-l K C i S)
})›

lemma *DECO-clause-l-DECO-clause*[*simp*]:
‹mset (DECO-clause-l M1) = DECO-clause M1›
by (*induction M1*) (*auto simp*: *DECO-clause-l-def DECO-clause-def convert-lits-l-def*)

lemma *TWL-DECO-clause-alt-def*:
‹TWL-DECO-clause M1 =
TWL-Clause (mset (watched-l (DECO-clause-l M1)))
(mset (unwatched-l (DECO-clause-l M1)))›
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*]:
‹length (DECO-clause-l M) = count-decided M›
unfolding *DECO-clause-l-def count-decided-def* **by** *auto*

lemma *negate-mode-bj-nonunit-l*:
fixes *S* :: ‹′v twl-st-l› **and** *S′* :: ‹′v twl-st›
assumes
count-dec: ‹count-decided (get-trail-l S) > 1› **and**
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›
shows
‹negate-mode-bj-nonunit-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′))›

proof −
  have *H*: ‹*RETURN* (*propagate-nonunit-and-add-l x2* (*DECO-clause-l* (*get-trail-l S*)) *i x1*)
      ≤ ⇓ {(*S*, *S''*). (*S*, *S''*) ∈ *twl-st-l None* ∧ *twl-list-invs S* ∧
      clauses-to-update-l S = {#}}
        (*SPEC* (*negate-model-and-add-twl S'*))›
   if
    x-S: ‹*x* ∈ {(*T*, *K*).
          ∃ *M2 M1*.
            equality-except-trail S T ∧
            get-trail-l T = *M1* ∧
            (*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*get-trail-l S*)) ∧
            get-level (*get-trail-l S*) *K* = *count-decided* (*get-trail-l S*)}› **and**
    x: ‹*x* = (*x1*, *x2*)› **and**
    i: ‹*i* ∈ {*i*. *0* < *i* ∧ *i* ∉# *dom-m* (*get-clauses-l x1*)}›
   for *x* :: ‹*'v twl-st-l* × *'v literal*› **and**
     x1 :: ‹*'v twl-st-l*› **and** *x2* :: ‹*'v literal*› **and** *i* :: ‹*nat*›
  proof −
   obtain *M N U D NE UE Q* **where**
    x1: ‹*x1* = (*M*, *N*, *U*, *D*, *NE*, *UE*, *Q*)›
    by (*cases x1*)

   obtain *M1 M2* **where**
    S-x1: ‹*equality-except-trail S x1*› **and**
    tr-M1: ‹*get-trail-l x1* = *M1*› **and**
    decomp: ‹(*Decided x2* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*get-trail-l S*))› **and**
    lev-K: ‹*get-level* (*get-trail-l S*) *x2* = *count-decided* (*get-trail-l S*)›
    using *x-S* **unfolding** *x* **by** *blast*
   let *?y0* = ‹(λ(*M*, *Oth*). (*drop* (*length M* − *length* (*get-trail-l x1*)) (*get-trail S'*), *Oth*)) *S'*›
   let *?y1* = ‹*propagate-nonunit-and-add x2* (*TWL-DECO-clause* (*get-trail S'*)) *?y0*›
   obtain *M3* **where**
    M3: ‹*get-trail-l S* = *M3* @ *M2* @ *Decided x2* # *M1*›
    using *decomp* **by** *blast*
   have *confl'*: ‹*get-conflict S'* = *None*› **and**
    trail-S': ‹(*get-trail-l S*, *get-trail S'*) ∈ *convert-lits-l* (*get-clauses-l S*) (*get-unit-clauses-l S*)›
    using *confl SS'* **by** (*auto simp*: *twl-st-l-def*)
   have ‹*no-dup* (*trail* ($state_W$-*of S'*))›
    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 ‹*no-dup* (*get-trail-l S*)›
    using *SS'* **by** (*auto simp*: *twl-st twl-st-l*)
   then have [*simp*]: ‹*count-decided M3* = *0*› ‹*count-decided M2* = *0*›
    ‹*filter is-decided M3* = []›
    ‹*filter is-decided M2* = []›
    using *lev-K*
    by (*auto simp*: *M3 count-decided-0-iff*)
   obtain *M2'* **where**
    decomp': ‹(*Decided x2* # *drop* (*length* (*get-trail S'*) − *length M1*) (*get-trail S'*), *M2'*)
       ∈ *set* (*get-all-ann-decomposition* (*get-trail S'*))› **and**
    conv: ‹(*get-trail-l S*, *get-trail S'*) ∈ *convert-lits-l* (*get-clauses-l S*)
     (*get-unit-clauses-l S*)› **and**
    conv-M1: ‹(*M1*, *drop* (*length* (*get-trail S'*) − *length M1*) (*get-trail S'*))
        ∈ *convert-lits-l* (*get-clauses-l S*) (*get-unit-clauses-l S*)›
    using *convert-lits-l-decomp-ex*[*OF decomp*, *of* ‹*get-trail S'*› ‹*get-clauses-l S*›
     ‹*get-unit-clauses-l S*›] *SS'*
    by (*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-twl 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-twl.bj-nonunit[of - - M2'])
    ultimately have ‹∃ y∈Collect (negate-model-and-add-twl 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 :: ‹'v twl-st-l ⇒ bool› where
‹negate-mode-restart-nonunit-l-inv S ⟷
  (∃ S' b. (S, S') ∈ twl-st-l b ∧ twl-struct-invs S' ∧ twl-list-invs S ∧ twl-stgy-invs S' ∧
     count-decided (get-trail-l S) > 1 ∧ get-conflict-l S = None)›

definition negate-mode-restart-nonunit-l :: ‹'v twl-st-l ⇒ 'v twl-st-l nres› where
‹negate-mode-restart-nonunit-l = (λS. do {
    ASSERT(negate-mode-restart-nonunit-l-inv S);
    let C = DECO-clause-l (get-trail-l S);
    i ← SPEC(λi. i < count-decided (get-trail-l S));
    (S, K) ← find-decomp-target i S;
    i ← get-fresh-index (get-clauses-l S);
    RETURN (restart-nonunit-and-add-l C i S)
  })›

lemma negate-mode-restart-nonunit-l:
  fixes S :: ‹'v twl-st-l› and S' :: ‹'v twl-st›
  assumes
    count-dec: ‹count-decided (get-trail-l S) > 1› and
    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›
  shows
    ‹negate-mode-restart-nonunit-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'))›
proof -
  have H: ‹RETURN (restart-nonunit-and-add-l (DECO-clause-l (get-trail-l S)) i x1)
        ≤ ⇓ {(S, S''). (S, S'') ∈ twl-st-l None ∧ twl-list-invs S ∧
        clauses-to-update-l S = {#}}
          (SPEC (negate-model-and-add-twl S'))›
    if
      j: ‹j ∈ {i. i < count-decided (get-trail-l S)}› and
      x-S: ‹x ∈ {(T, K).
            ∃ M2 M1.
              equality-except-trail S T ∧
              get-trail-l T = M1 ∧
              (Decided K # M1, M2) ∈ set (get-all-ann-decomposition (get-trail-l S)) ∧
              get-level (get-trail-l S) K = j}› and
      x: ‹x = (x1, x2)› and
      i: ‹i ∈ {i. 0 < i ∧ i ∉# dom-m (get-clauses-l x1)}›
    for x :: ‹'v twl-st-l × 'v literal› and
       x1 :: ‹'v twl-st-l› and x2 :: ‹'v literal› and i j :: ‹nat›
  proof -
    obtain M N U D NE UE Q where
      x1: ‹x1 = (M, N, U, D, NE, UE, Q)›
      by (cases x1)

    obtain M1 M2 where
      S-x1: ‹equality-except-trail S x1› and
      tr-M1: ‹get-trail-l x1 = M1› and
      decomp: ‹(Decided x2 # M1, M2) ∈ set (get-all-ann-decomposition (get-trail-l S))› and
      lev-K: ‹get-level (get-trail-l S) x2 = j›
```

```
  using x-S unfolding x by blast
let ?y0 = ‹(λ(M, Oth). (drop (length (get-trail S') − length M1) (get-trail S'), Oth)) S'›
let ?y1 = ‹restart-nonunit-and-add (TWL-DECO-clause (get-trail S')) ?y0›

obtain M3 where
  M3: ‹get-trail-l S = M3 @ M2 @ Decided x2 # M1›
  using decomp by blast
have ‹M = M1›
  using S-x1 SS' decomp tr-M1 unfolding x1
  by (cases S; cases S') auto
have confl': ‹get-conflict S' = None› and
  trail-S': ‹(get-trail-l S, get-trail S') ∈ convert-lits-l (get-clauses-l S) (get-unit-clauses-l S)›
  using confl SS' by (auto simp: twl-st-l)
have ‹no-dup (trail (state_W-of S'))›
  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 ‹no-dup (get-trail-l S)›
  using SS' by (auto simp: twl-st twl-st-l)
obtain M2' where
  decomp': ‹(Decided x2 # drop (length (get-trail S') − length M1) (get-trail S'), M2')
      ∈ set (get-all-ann-decomposition (get-trail S'))› and
  conv: ‹(get-trail-l S, get-trail S') ∈ convert-lits-l (get-clauses-l S)
    (get-unit-clauses-l S)› and
  conv-M1: ‹(M1, drop (length (get-trail S') − length M1) (get-trail S'))
        ∈ convert-lits-l (get-clauses-l S) (get-unit-clauses-l S)›
  using convert-lits-l-decomp-ex[OF decomp, of ‹get-trail S'› ‹get-clauses-l S›
    ‹get-unit-clauses-l S›] SS'
  by (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 ‹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 ‹(restart-nonunit-and-add-l (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 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 ‹D+NE›])
moreover {
  have ‹get-level (get-trail-l S) x2 < count-decided (get-trail-l S)›
    using lev-K j by auto
  then have ‹?y1 ∈ Collect (negate-model-and-add-twl S')›
    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-twl.restart-nonunit[of x2 - ‹M2'›])
}
ultimately have ‹∃ y∈Collect (negate-model-and-add-twl S').
    (restart-nonunit-and-add-l (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
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*⟩
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 :: ‹′*v literal set* ⇒ *bool*›
begin

definition *cdcl-twl-enum-inv-l* :: ‹′*v twl-st-l* ⇒ *bool*› **where**
‹*cdcl-twl-enum-inv-l S* ⟷
(∃ *S*′. (*S*, *S*′) ∈ *twl-st-l None* ∧ *cdcl-twl-enum-inv S*′) ∧
twl-list-invs S›

definition *cdcl-twl-enum-l* :: ‹′*v twl-st-l* ⇒ *bool nres*› **where**
‹*cdcl-twl-enum-l S* = *do* {
S ← *cdcl-twl-stgy-prog-l S*;
S ← $WHILE_T{}^{cdcl\text{-}twl\text{-}enum\text{-}inv\text{-}l}$
(λ*S*. *get-conflict-l S* = *None* ∧ *count-decided*(*get-trail-l S*) > *0* ∧
¬*P* (*lits-of-l* (*get-trail-l S*)))
(λ*S*. *do* {
S ← *negate-mode-l S*;
cdcl-twl-stgy-prog-l S
})
S;
if get-conflict-l S = *None*
then RETURN (*if count-decided*(*get-trail-l S*) = *0 then P* (*lits-of-l* (*get-trail-l S*)) *else True*)
else RETURN (*False*)
}›

lemma *negate-model-and-add-twl-resultD*:
‹*negate-model-and-add-twl S T* ⟹
clauses-to-update T = {#} ∧ *get-conflict T* = *None*›
by (*auto simp*: *negate-model-and-add-twl.simps*)

lemma *cdcl-twl-enum-l*:
fixes *S* :: ‹′*v twl-st-l*› **and** *S*′ :: ‹′*v twl-st*›
assumes
SS′: ‹(*S*, *S*′) ∈ *twl-st-l None*› **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*› **and**
‹*clauses-to-update-l S* = {#}›

shows
‹*cdcl-twl-enum-l S* ≤ ⇓ *bool-rel*
(*cdcl-twl-enum P S′*)›
unfolding *cdcl-twl-enum-l-def cdcl-twl-enum-def*
apply (*refine-vcg cdcl-twl-stgy-prog-l-spec-final′ negate-mode-l*)
subgoal
using *assms* **unfolding** *cdcl-twl-stgy-prog-l-pre-def*
by *fast*
apply *assumption*
subgoal for *S S′ U U′*
using *assms* **unfolding** *cdcl-twl-enum-inv-l-def*
apply −
apply (*intro conjI*)
apply (*rule exI*[*of - U′*])
by *auto*
subgoal by (*auto simp*: *twl-st-l*)
apply *auto*[]
subgoal unfolding *cdcl-twl-enum-inv-def* **by** *auto*
subgoal by *fast*
subgoal by (*auto simp*: *twl-st-l cdcl-twl-enum-inv-def*)
subgoal by (*auto simp*: *twl-st-l*)
subgoal by (*auto simp*: *twl-st-l*)
subgoal for *S S′ T T′ U U′*
by (*rule cdcl-twl-stgy-prog-l-spec-final′*[*THEN order.trans*])
(*auto simp*: *twl-st twl-st-l cdcl-twl-stgy-prog-l-pre-def cdcl-twl-enum-inv-def*
intro: *negate-model-and-add-twl-twl-struct-invs*
negate-model-and-add-twl-twl-stgy-invs conc-fun-R-mono
dest: *negate-model-and-add-twl-resultD*)
subgoal by (*auto simp*: *twl-st-l*)
subgoal by (*auto simp*: *twl-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 twl-st-wl* ⇒ (′*v twl-st-wl* × ′*v literal*) *nres*› **where**
‹*find-decomp-target-wl* = (λ*i 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 twl-st-wl* ⇒ ′*v twl-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 twl-st-wl* ⇒ ′*v twl-st-wl nres*› **where**
‹*negate-mode-bj-unit-wl* = (λ*S*. *do* {
(*S*, *K*) ← *find-decomp-target-wl 1 S*;
ASSERT(*K* ∈# *all-lits-of-mm* (*clause* ‘# *twl-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**
⟨*find-decomp-target-wl-ref S* ≡
{((*T*, *K*), (*T'*, *K'*)). (*T*, *T'*) ∈ {(*T*, *T'*). (*T*, *T'*) ∈ *state-wl-l None* ∧ *correct-watching T*} ∧
(*K* , *K'*) ∈ *Id* ∧
K ∈# *all-lits-of-mm* (*clause* ‘# *twl-clause-of* ‘# *ran-mf* (*get-clauses-wl T*) +
get-unit-clauses-wl T) ∧
K ∈# *all-lits-of-mm* (*clause* ‘# *twl-clause-of* ‘# *ran-mf* (*get-clauses-wl T*) +
get-unit-init-clss-wl T) ∧ *equality-except-trail-wl S T* ∧
atms-of (*DECO-clause* (*get-trail-wl S*)) ⊆ *atms-of-mm* (*clause* ‘# *twl-clause-of* ‘# *ran-mf* (*get-clauses-wl T*) +
get-unit-init-clss-wl T) ∧ *distinct-mset* (*DECO-clause* (*get-trail-wl S*)) ∧
correct-watching T}⟩

lemma *DECO-clause-nil*[*simp*]: ⟨*DECO-clause* [] = {#}⟩
by (*auto simp*: *DECO-clause-def*)

lemma *in-DECO-clauseD*: ⟨*x* ∈# *DECO-clause M* ⟹ −*x* ∈ *lits-of-l M*⟩
by (*auto simp*: *DECO-clause-def lits-of-def*)

lemma *in-atms-of-DECO-clauseD*: ⟨*x* ∈ *atms-of* (*DECO-clause M*) ⟹ *x* ∈ *atm-of* ‘ (*lits-of-l M*)⟩
by (*auto simp*: *DECO-clause-def lits-of-def atms-of-def*)

lemma *no-dup-distinct-mset-DECO-clause*:
assumes ⟨*no-dup M*⟩
shows ⟨*distinct-mset* (*DECO-clause M*)⟩
proof −
have ⟨*distinct* (*map lit-of* (*filter is-decided M*))⟩
using *no-dup-map-lit-of*[*OF assms*] *distinct-map-filter* **by** *blast*
moreover have ⟨*?thesis* ⟷ *distinct* (*map lit-of* (*filter is-decided M*))⟩
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': ⟨(*S*, *S'*) ∈ {(*S*, *S''*). (*S*, *S''*) ∈ *state-wl-l None* ∧ *correct-watching S*}⟩ **and**
inv: ⟨∃ *S''* *b*. (*S'*, *S''*) ∈ *twl-st-l b* ∧ *twl-struct-invs S''*⟩ **and**
[*simp*]: ⟨*a* = *a'*⟩
shows ⟨*find-decomp-target-wl a S* ≤
⇓ (*find-decomp-target-wl-ref S*) (*find-decomp-target a' S'*)⟩
(**is** ⟨- ≤ ⇓ *?negate* -⟩)
proof −
let *?y0* = ⟨λ*S S'*. (λ(*M*, *Oth*). (*get-trail-wl S*, *Oth*)) *S'*⟩
have *K*: ⟨⋀*K*. *K* ∈ *lits-of-l* (*get-trail-wl S*) ⟹
K ∈# *all-lits-of-mm* (*clause* ‘# *twl-clause-of* ‘# *ran-mf* (*get-clauses-wl S*) +
get-unit-init-clss-wl S)⟩ (**is** ⟨⋀*K*. *?HK K* ⟹ *?K K*⟩) **and**
DECO:
⟨*atms-of* (*DECO-clause* (*get-trail-wl S*)) ⊆ *atms-of-mm* (*clause* ‘# *twl-clause-of* ‘# *ran-mf* (*get-clauses-wl S*) +
get-unit-init-clss-wl S)⟩ (**is** *?DECO*) **and**

```
    distinct-DECO:
      ‹distinct-mset (DECO-clause (get-trail-wl S))› (is ?dist-DECO)
  proof −
    obtain b S'' where
      S'-S'': ‹(S', S'') ∈ twl-st-l b› and
      struct: ‹twl-struct-invs S''›
      using inv unfolding negate-mode-bj-unit-l-inv-def by blast
    then have no-alien: ‹cdcl_W-restart-mset.cdcl_W-all-struct-inv (state_W-of S'')›
      using struct unfolding twl-struct-invs-def by fast
    then have no-alien: ‹cdcl_W-restart-mset.no-strange-atm (state_W-of S'')› and
      M-lev: ‹cdcl_W-restart-mset.cdcl_W-M-level-inv (state_W-of S'')›
      unfolding cdcl_W-restart-mset.cdcl_W-all-struct-inv-def by fast+
    moreover have ‹atms-of-mm (get-all-init-clss S'') =
          atms-of-mm (mset `# (ran-mf (get-clauses-wl S)) + get-unit-init-clss-wl S)›
      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)
        (auto simp: mset-take-mset-drop-mset' cdcl_W-restart-mset-state
        in-all-lits-of-mm-ain-atms-of-iff twl-st-l-def state-wl-l-def)
    ultimately show ‹⋀K. ?HK K ⟹ ?K K›
      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'
        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'
        in-all-lits-of-mm-ain-atms-of-iff get-unit-clauses-wl-alt-def
        dest: in-atms-of-DECO-clauseD)

    show ?dist-DECO
      by (rule no-dup-distinct-mset-DECO-clause)
       (use M-lev S'-S'' SS' in ‹auto simp: cdcl_W-restart-mset.cdcl_W-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 DECO distinct-DECO
      by (cases S; cases S')
       (force simp: state-wl-l-def correct-watching.simps clause-to-update-def
          mset-take-mset-drop-mset' all-lits-of-mm-union
          dest!: get-all-ann-decomposition-exists-prepend)+
    subgoal
      by (cases S; cases S')
        (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 :: ‹'v twl-st-wl› and S' :: ‹'v twl-st-l›
  assumes ‹count-decided (get-trail-wl S) = 1› and
    SS': ‹(S, S') ∈ {(S, S'). (S, S') ∈ state-wl-l None ∧ correct-watching S}›
```

shows
  ‹*negate-mode-bj-unit-wl* $S \leq \Downarrow\{(S, S'). (S, S') \in$ *state-wl-l None* $\wedge$ *correct-watching* $S\}$
    (*negate-mode-bj-unit-l* S')›
    (**is** ‹- $\leq \Downarrow$ *?R* -›)
proof −
  have *2*: ‹(*propagate-unit-and-add-wl x2a x1a*, *propagate-unit-and-add-l x2 x1*)
      $\in \{(S, S''). (S, S'') \in$ *state-wl-l None* $\wedge$ *correct-watching* $S\}$›
    if
      ‹$(x, x') \in$ *find-decomp-target-wl-ref S*› **and**
      ‹$x' = (x1, x2)$› **and**
      ‹$x = (x1a, x2a)$›
    for *x2a x1a x2 x1* **and** *x* :: ‹*'v twl-st-wl* × *'v literal*› **and** *x'* :: ‹*'v twl-st-l* × *'v literal*›
  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* ⇒ *'v clause-l* ⇒ *nat* ⇒ *'v twl-st-wl* ⇒ *bool*› **where**
  ‹*propagate-nonunit-and-add-wl-pre K C i S* ⟷
    length C $\geq 2 \wedge i > 0 \wedge i \notin\#$ *dom-m* (*get-clauses-wl S*) ∧
    atms-of (*mset C*) ⊆ *atms-of-mm* (*clause* ‘# *twl-clause-of* ‘# *ran-mf* (*get-clauses-wl S*) +
      get-unit-init-clss-wl S)›

fun *propagate-nonunit-and-add-wl*
  :: ‹*'v literal* ⇒ *'v clause-l* ⇒ *nat* ⇒ *'v twl-st-wl* ⇒ *'v twl-st-wl nres*›
where
  ‹*propagate-nonunit-and-add-wl K C i* (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*) = *do* {
    ASSERT(*propagate-nonunit-and-add-wl-pre K C i* (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*));
    let b = (*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*, *fmupd i* (*C*, *True*) *N*, *None*,
    NE, *UE*, {#*K*#}, *W*)
  }›

lemma *twl-st-l-splitD*:
  ‹(⋀*M N D NE UE Q W*. *f* (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*) = *P M N D NE UE Q W*) ⟹
   f S = *P* (*get-trail-l S*) (*get-clauses-l S*) (*get-conflict-l S*) (*get-unit-init-clauses-l S*)
   (*get-unit-learned-clauses-l S*) (*clauses-to-update-l S*) (*literals-to-update-l S*)›
  by (*cases S*) *auto*

lemma *twl-st-wl-splitD*:
‹(⋀*M N D NE UE Q W*. *f* (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*) = *P M N D NE UE Q W*) ⟹
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*)›
by (*cases S*) *auto*

definition *negate-mode-bj-nonunit-wl-inv* **where**
‹*negate-mode-bj-nonunit-wl-inv S* ⟷
(∃ *S''* *b*. (*S*, *S''*) ∈ *state-wl-l b* ∧ *negate-mode-bj-nonunit-l-inv S''* ∧ *correct-watching S*)›

definition *negate-mode-bj-nonunit-wl* :: ‹'*v twl-st-wl* ⇒ '*v twl-st-wl nres*› **where**
‹*negate-mode-bj-nonunit-wl* = (λ*S*. *do* {
ASSERT(*negate-mode-bj-nonunit-wl-inv S*);
let C = *DECO-clause-l* (*get-trail-wl S*);
(*S*, *K*) ← *find-decomp-target-wl* (*count-decided* (*get-trail-wl S*)) *S*;
i ← *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
})›

lemmas *propagate-nonunit-and-add-wl-def* =
twl-st-wl-splitD[*of* ‹*propagate-nonunit-and-add-wl* - - -›, *OF propagate-nonunit-and-add-wl.simps*]

lemmas *propagate-nonunit-and-add-l-def* =
twl-st-l-splitD[*of* ‹*propagate-nonunit-and-add-l* - - -›, *OF propagate-nonunit-and-add-l.simps*,
rule-format]

lemma *atms-of-subset-in-atms-ofI*:
‹*atms-of C* ⊆ *atms-of-ms N* ⟹ *L* ∈# *C* ⟹ *atm-of L* ∈ *atms-of-ms N*›
by (*auto dest*!: *multi-member-split*)

lemma *in-DECO-clause-l-in-DECO-clause-iff*:
‹*x* ∈ *set* (*DECO-clause-l M*) ⟷ *x* ∈# (*DECO-clause M*)›
by (*metis DECO-clause-l-DECO-clause set-mset-mset*)

lemma *distinct-DECO-clause-l*:
‹*no-dup M* ⟹ *distinct* (*DECO-clause-l M*)›
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': ‹(*S*, *S'*) ∈ *state-wl-l None*› **and**
inv: ‹*negate-mode-bj-nonunit-wl-inv 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 ‹*propagate-nonunit-and-add-wl K'* (*DECO-clause-l* (*get-trail-wl S*)) *i T'*
≤ *SPEC* (λ*c*. (*c*, *propagate-nonunit-and-add-l K*
(*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› **and** j: ‹j ∉# dom-m (get-clauses-wl T')›
using ij **by** auto
have [simp]: ‹DECO-clause-l (get-trail-l S') = DECO-clause-l (get-trail-wl S)›
using SS' **by** auto
obtain T U b b' **where**
ST: ‹(S, T) ∈ state-wl-l b› **and**
corr: ‹correct-watching S› **and**
TU: ‹(T, U) ∈ twl-st-l b'› **and**
‹twl-list-invs T› **and**
ge1: ‹1 < count-decided (get-trail-l T)› **and**
st: ‹twl-struct-invs U› **and**
‹twl-stgy-invs U› **and**
‹get-conflict-l T = None›
using inv **unfolding** negate-mode-bj-nonunit-wl-inv-def negate-mode-bj-nonunit-l-inv-def **apply** −
by blast
have ‹length (DECO-clause-l (get-trail-wl S)) > 1›
using ST ge1 **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
have ‹no-dup (trail (state$_W$-of U))›
using st **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 neq: False **if** ‹DECO-clause-l (get-trail-wl S) ! 0 = DECO-clause-l (get-trail-wl S) ! Suc
0›
using that
apply (subst (asm) nth-eq-iff-index-eq)
using ge1 ST TU **by** (auto simp: twl-st twl-st-l twl-st-wl distinct-DECO-clause-l)

show ?thesis
using TK j corr ge1 ST
apply (simp only: propagate-nonunit-and-add-wl-def
propagate-nonunit-and-add-l-def Let-def
assert-bind-spec-conv)
apply (intro conjI)
subgoal using j ij TK **unfolding** propagate-nonunit-and-add-wl-pre-def **by** 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 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 ‹get-trail-wl S›]
by (cases ‹DECO-clause-l (get-trail-wl S)›; cases ‹tl (DECO-clause-l (get-trail-wl S))›)
(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*[*THEN iffD2*])
apply (*rule atms-of-subset-in-atms-ofI*[*of* ‹*DECO-clause* (*get-trail-wl S*)›])
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*:
‹*watched-by T L* = *get-watched-wl T L*›
by (*cases T*) *auto*

lemma *negate-mode-bj-nonunit-wl-negate-mode-bj-nonunit-l*:
fixes *S* :: ‹′*v twl-st-wl*› **and** *S′* :: ‹′*v twl-st-l*›
assumes
SS′: ‹(*S*, *S′*) ∈ {(*S*, *S′′*). (*S*, *S′′*) ∈ *state-wl-l None* ∧ *correct-watching S*}›
shows
‹*negate-mode-bj-nonunit-wl S* ≤ ⇓{(*S*, *S′′*). (*S*, *S′′*) ∈ *state-wl-l None* ∧ *correct-watching S*}
(*negate-mode-bj-nonunit-l S′*)›
proof −
have *fresh*: ‹*get-fresh-index-wl* (*get-clauses-wl T*) (*get-unit-clauses-wl T*) (*get-watched-wl T*)
≤ ⇓ {(*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*))}
(*get-fresh-index* (*get-clauses-l T′*))›
if ‹(*TK*, *TK′*) ∈ *find-decomp-target-wl-ref S*› **and**
‹*TK* = (*T*, *K*)› **and**
‹*TK′* =(*T′*, *K′*)›

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* :: ‹*′v twl-st-wl* ⇒ *bool*› **where**
‹*negate-mode-restart-nonunit-wl-inv S* ⟷
(∃ *S′ b*. (*S*, *S′*) ∈ *state-wl-l b* ∧ *negate-mode-restart-nonunit-l-inv S′* ∧ *correct-watching S*)›

definition *restart-nonunit-and-add-wl-inv* **where**
‹*restart-nonunit-and-add-wl-inv C i S* ⟷
length C ≥ *2* ∧ *correct-watching S* ∧
atms-of (*mset C*) ⊆ *atms-of-mm* (*clause* ‘# *twl-clause-of* ‘# *ran-mf* (*get-clauses-wl S*) + *get-unit-init-clss-wl S*)›

fun *restart-nonunit-and-add-wl* :: ‹*′v clause-l* ⇒ *nat* ⇒ *′v twl-st-wl* ⇒ *′v twl-st-wl nres*› **where**
‹*restart-nonunit-and-add-wl C i* (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*) = *do* {
ASSERT(*restart-nonunit-and-add-wl-inv C i* (*M*, *N*, *D*, *NE*, *UE*, *Q*, *W*));
let b = (*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 (*M*, *fmupd i* (*C*, *True*) *N*, *None*, *NE*, *UE*, {#}, *W*)
}›

definition *negate-mode-restart-nonunit-wl* :: ‹*′v twl-st-wl* ⇒ *′v twl-st-wl nres*› **where**
‹*negate-mode-restart-nonunit-wl* = (λ*S*. *do* {
ASSERT(*negate-mode-restart-nonunit-wl-inv S*);
let C = *DECO-clause-l* (*get-trail-wl S*);
i ← *SPEC*(λ*i*. *i* < *count-decided* (*get-trail-wl S*));
(*S*, *K*) ← *find-decomp-target-wl i S*;
i ← *get-fresh-index-wl* (*get-clauses-wl S*) (*get-unit-clauses-wl S*) (*get-watched-wl S*);
restart-nonunit-and-add-wl C i S
})›

definition *negate-mode-wl-inv* **where**
‹*negate-mode-wl-inv S* ⟷
(∃ *S′ b*. (*S*, *S′*) ∈ *state-wl-l b* ∧ *negate-mode-l-inv S′* ∧ *correct-watching S*)›

definition *negate-mode-wl* :: ‹*′v twl-st-wl* ⇒ *′v twl-st-wl nres*› **where**
‹*negate-mode-wl S* = *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* (λ*c*. (*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: ‹(*S*, *T*) ∈ *state-wl-l b*› **and**
‹*no-dup* (*trail* ($state_W$*-of U*))› **and**
TU: ‹(*T*, *U*) ∈ *twl-st-l b'*›
using *inv* **unfolding** *negate-mode-restart-nonunit-wl-inv-def negate-mode-restart-nonunit-l-inv-def*
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 *neq*: *False* **if** ‹*DECO-clause-l* (*get-trail-wl S*) ! *0* = *DECO-clause-l* (*get-trail-wl S*) ! *Suc 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* ‹*restart-nonunit-and-add-wl - -*›,
OF restart-nonunit-and-add-wl.simps]
twl-st-l-splitD[*of* ‹*restart-nonunit-and-add-l - -*›,
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* ‹*get-trail-wl S*›] *le TK*
by (*cases* ‹*DECO-clause-l* (*get-trail-wl S*)›; *cases* ‹*tl* (*DECO-clause-l* (*get-trail-wl S*))›)
(*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* ‹*DECO-clause* (*get-trail-wl S*)›])
subgoal using *TK* **by** (*solves* ‹*auto simp add*: *mset-take-mset-drop-mset'*›)
subgoal using *TK le* **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 TK le* **in** ‹*auto simp add*: *mset-take-mset-drop-mset' 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 neq* **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* :: ‹′*v twl-st-wl*› **and** *S′* :: ‹′*v twl-st-l*›
assumes
SS′: ‹(*S*, *S′*) ∈ {(*S*, *S″*). (*S*, *S″*) ∈ *state-wl-l None* ∧ *correct-watching S*}›
shows
‹*negate-mode-restart-nonunit-wl S* ≤
⇓ {(*S*, *S″*). (*S*, *S″*) ∈ *state-wl-l None* ∧ *correct-watching S*}
(*negate-mode-restart-nonunit-l S′*)›
proof −
have *fresh*: ‹*get-fresh-index-wl* (*get-clauses-wl T*) (*get-unit-clauses-wl T*) (*get-watched-wl T*)
≤ ⇓ {(*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*))}
(*get-fresh-index* (*get-clauses-l T′*))›
if ‹(*TK*, *TK′*) ∈ *find-decomp-target-wl-ref S*› **and**
‹*TK* = (*T*, *K*)› **and**
‹*TK′* =(*T′*, *K′*)›
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* :: ‹′*v twl-st-wl*› **and** *S′* :: ‹′*v twl-st-l*›

assumes
SS′: ‹(S, S′) ∈ {(S, S′′). (S, S′′) ∈ *state-wl-l None* ∧ *correct-watching S*}› **and**
confl: ‹*get-conflict-wl S = None*›
shows
‹*negate-mode-wl S* ≤
⇓ {(S, S′′). (S, S′′) ∈ *state-wl-l None* ∧ *correct-watching S*}
(*negate-mode-l S′*)›
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* :: ‹′*v literal set* ⇒ *bool*›
begin

definition *cdcl-twl-enum-inv-wl* :: ‹′*v twl-st-wl* ⇒ *bool*› **where**
‹*cdcl-twl-enum-inv-wl S* ⟷
(∃ S′. (S, S′) ∈ *state-wl-l None* ∧ *cdcl-twl-enum-inv-l S′*) ∧
correct-watching S›

definition *cdcl-twl-enum-wl* :: ‹′*v twl-st-wl* ⇒ *bool nres*› **where**
‹*cdcl-twl-enum-wl S* = *do* {
S ← *cdcl-twl-stgy-prog-wl S*;
S ← $WHILE_T^{cdcl\text{-}twl\text{-}enum\text{-}inv\text{-}wl}$
(λS. *get-conflict-wl S = None* ∧ *count-decided*(*get-trail-wl S*) > 0 ∧
¬P (*lits-of-l* (*get-trail-wl S*)))
(λS. *do* {
S ← *negate-mode-wl S*;
cdcl-twl-stgy-prog-wl S
})
S;
if get-conflict-wl S = None
then RETURN (*if count-decided*(*get-trail-wl S*) = 0 *then P* (*lits-of-l* (*get-trail-wl S*)) *else True*)
else RETURN (*False*)
}›

lemma *cdcl-twl-enum-wl-cdcl-twl-enum-l*:
assumes
SS′: ‹(S, S′) ∈ *state-wl-l None*› **and**
corr: ‹*correct-watching S*›
shows
‹*cdcl-twl-enum-wl S* ≤ ⇓ *bool-rel*
(*cdcl-twl-enum-l P S′*)›
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-twl-enum-inv-wl-def* **by** *blast*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
subgoal by *auto*
done

end

end