Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

March 23, 2016

Contents

theory *Partial-Annotated-Clausal-Logic*
imports *Partial-Clausal-Logic*

begin

1 Partial Clausal Logic

We here define marked literals (that will be used in both DPLL and CDCL) and the entailment corresponding to it.

1.1 Marked Literals

1.1.1 Definition

datatype ($'v$, $'lvl$, $'mark$) *ann-literal* =
  is-marked: *Marked* (*lit-of*: $'v$ *literal*) (*level-of*: $'lvl$) |
  is-proped: *Propagated* (*lit-of*: $'v$ *literal*) (*mark-of*: $'mark$)

lemma *ann-literal-list-induct*[*case-names nil marked proped*]:
  assumes P [] **and**
  $\bigwedge L\ l\ xs.\ P\ xs \Longrightarrow P\ (Marked\ L\ l\ \#\ xs)$ **and**
  $\bigwedge L\ m\ xs.\ P\ xs \Longrightarrow P\ (Propagated\ L\ m\ \#\ xs)$
  shows $P\ xs$
  using *assms* **apply** (*induction xs*, *simp*)
  by (*rename-tac a xs*, *case-tac a*) *auto*

lemma *is-marked-ex-Marked*:
is-marked $L \Longrightarrow \exists K\ lvl.\ L = Marked\ K\ lvl$
by (*cases L*) *auto*

type-synonym $('v, 'l, 'm)$ *ann-literals* = $('v, 'l, 'm)$ *ann-literal list*

definition *lits-of* :: $('a, 'b, 'c)$ *ann-literal list* $\Rightarrow$ *'a literal set* **where**
lits-of Ls = *lit-of* ‘ (*set Ls*)

lemma *lits-of-empty*[*simp*]:
lits-of [] = {} **unfolding** *lits-of-def* **by** *auto*

lemma *lits-of-cons*[*simp*]:
lits-of (*L* # *Ls*) = *insert* (*lit-of L*) (*lits-of Ls*)
unfolding *lits-of-def* **by** *auto*

lemma *lits-of-append*[*simp*]:
lits-of $(l\ @\ l')$ = *lits-of* $l \cup$ *lits-of* l'
unfolding *lits-of-def* **by** *auto*

lemma *finite-lits-of-def*[*simp*]: *finite* (*lits-of L*)
unfolding *lits-of-def* **by** *auto*

lemma *lits-of-rev*[*simp*]: *lits-of* (*rev M*) = *lits-of M*
unfolding *lits-of-def* **by** *auto*

lemma *set-map-lit-of-lits-of*[*simp*]:
set (*map lit-of T*) = *lits-of T*
unfolding *lits-of-def* **by** *auto*

abbreviation *unmark* **where**
unmark M $\equiv$ (λa. {#*lit-of a*#}) ‘ *set M*

lemma *atms-of-ms-lambda-lit-of-is-atm-of-lit-of*[*simp*]:
atms-of-ms (*unmark* M') = *atm-of* ‘ *lits-of* M'
unfolding *atms-of-ms-def lits-of-def* **by** *auto*

lemma *lits-of-empty-is-empty*[*iff*]:
lits-of M = {} $\longleftrightarrow$ *M* = []
by (*induct M*) *auto*

1.1.2 Entailment

definition *true-annot* :: $('a, 'l, 'm)$ *ann-literals* $\Rightarrow$ *'a clause* $\Rightarrow$ *bool* (**infix** $\models a$ *49*) **where**
$I \models a\ C \longleftrightarrow$ (*lits-of I*) $\models C$

definition *true-annots* :: $('a, 'l, 'm)$ *ann-literals* $\Rightarrow$ *'a clauses* $\Rightarrow$ *bool* (**infix** $\models as$ *49*) **where**
$I \models as\ CC \longleftrightarrow (\forall C \in CC.\ I \models a\ C)$

lemma *true-annot-empty-model*[*simp*]:
$\neg[] \models a\ \psi$
unfolding *true-annot-def true-cls-def* **by** *simp*

lemma *true-annot-empty*[*simp*]:
$\neg I \models a$ {#}
unfolding *true-annot-def true-cls-def* **by** *simp*

lemma *empty-true-annots-def*[*iff*]:
$[] \models as\ \psi \longleftrightarrow \psi = \{\}$
unfolding *true-annots-def* **by** *auto*

lemma *true-annots-empty*[*simp*]:
$I \models as\ \{\}$
unfolding *true-annots-def* **by** *auto*

lemma *true-annots-single-true-annot*[*iff*]:
$I \models as\ \{C\} \longleftrightarrow I \models a\ C$
unfolding *true-annots-def* **by** *auto*

lemma *true-annot-insert-l*[*simp*]:
$M \models a\ A \Longrightarrow L\ \#\ M \models a\ A$
unfolding *true-annot-def* **by** *auto*

lemma *true-annots-insert-l* [*simp*]:
$M \models as\ A \Longrightarrow L\ \#\ M \models as\ A$
unfolding *true-annots-def* **by** *auto*

lemma *true-annots-union*[*iff*]:
$M \models as\ A \cup B \longleftrightarrow (M \models as\ A \wedge M \models as\ B)$
unfolding *true-annots-def* **by** *auto*

lemma *true-annots-insert*[*iff*]:
$M \models as\ insert\ a\ A \longleftrightarrow (M \models a\ a \wedge M \models as\ A)$
unfolding *true-annots-def* **by** *auto*

Link between $\models as$ and $\models s$:

lemma *true-annots-true-cls*:
$I \models as\ CC \longleftrightarrow (lits\text{-}of\ I) \models s\ CC$
unfolding *true-annots-def Ball-def true-annot-def true-clss-def* **by** *auto*

lemma *in-lit-of-true-annot*:
$a \in lits\text{-}of\ M \longleftrightarrow M \models a\ \{\#a\#\}$
unfolding *true-annot-def lits-of-def* **by** *auto*

lemma *true-annot-lit-of-notin-skip*:
$L\ \#\ M \models a\ A \Longrightarrow lit\text{-}of\ L \notin\#\ A \Longrightarrow M \models a\ A$
unfolding *true-annot-def true-cls-def* **by** *auto*

lemma *true-clss-singleton-lit-of-implies-incl*:
$I \models s\ unmark\ MLs \Longrightarrow lits\text{-}of\ MLs \subseteq I$
unfolding *true-clss-def lits-of-def* **by** *auto*

lemma *true-annot-true-clss-cls*:
$MLs \models a\ \psi \Longrightarrow set\ (map\ (\lambda a.\ \{\#lit\text{-}of\ a\#\})\ MLs) \models p\ \psi$
unfolding *true-annot-def true-clss-cls-def true-cls-def*
by (*auto dest*: *true-clss-singleton-lit-of-implies-incl*)

lemma *true-annots-true-clss-cls*:
$MLs \models as\ \psi \Longrightarrow set\ (map\ (\lambda a.\ \{\#lit\text{-}of\ a\#\})\ MLs) \models ps\ \psi$
by (*auto*

dest: *true-clss-singleton-lit-of-implies-incl*
simp add: *true-clss-def true-annots-def true-annot-def lits-of-def true-cls-def true-clss-clss-def*)

lemma *true-annots-marked-true-cls*[*iff*]:
map (λM. *Marked M a*) *M* $\models$*as N* $\longleftrightarrow$ *set M* $\models$*s N*
proof −
have $*$: *lits-of* (*map* (λM. *Marked M a*) *M*) = *set M* **unfolding** *lits-of-def* **by** *force*
show *?thesis* **by** (*simp add*: *true-annots-true-cls* $*$)
qed

lemma *true-annot-singleton*[*iff*]: *M* $\models$*a* {#*L*#} $\longleftrightarrow$ *L* $\in$ *lits-of M*
unfolding *true-annot-def lits-of-def* **by** *auto*

lemma *true-annots-true-clss-clss*:
A $\models$*as* Ψ $\Longrightarrow$ *unmark A* $\models$*ps* Ψ
unfolding *true-clss-clss-def true-annots-def true-clss-def*
by (*auto*
dest!: *true-clss-singleton-lit-of-implies-incl*
simp add: *lits-of-def true-annot-def true-cls-def*)

lemma *true-annot-commute*:
M @ *M'* $\models$*a D* $\longleftrightarrow$ *M'* @ *M* $\models$*a D*
unfolding *true-annot-def* **by** (*simp add*: *Un-commute*)

lemma *true-annots-commute*:
M @ *M'* $\models$*as D* $\longleftrightarrow$ *M'* @ *M* $\models$*as D*
unfolding *true-annots-def* **by** (*auto simp add*: *true-annot-commute*)

lemma *true-annot-mono*[*dest*]:
set I $\subseteq$ *set I'* $\Longrightarrow$ *I* $\models$*a N* $\Longrightarrow$ *I'* $\models$*a N*
using *true-cls-mono-set-mset-l* **unfolding** *true-annot-def lits-of-def*
by (*metis* (*no-types*) *Un-commute Un-upper1 image-Un sup.orderE*)

lemma *true-annots-mono*:
set I $\subseteq$ *set I'* $\Longrightarrow$ *I* $\models$*as N* $\Longrightarrow$ *I'* $\models$*as N*
unfolding *true-annots-def* **by** *auto*

1.1.3 Defined and undefined literals

definition *defined-lit* :: (*'a*, *'l*, *'m*) *ann-literal list* $\Rightarrow$ *'a literal* $\Rightarrow$ *bool*
where
defined-lit I L $\longleftrightarrow$ ($\exists$ *l*. *Marked L l* $\in$ *set I*) $\vee$ ($\exists$ *P*. *Propagated L P* $\in$ *set I*)
$\vee$ ($\exists$ *l*. *Marked* ($-L$) *l* $\in$ *set I*) $\vee$ ($\exists$ *P*. *Propagated* ($-L$) *P* $\in$ *set I*)

abbreviation *undefined-lit* :: (*'a*, *'l*, *'m*) *ann-literal list* $\Rightarrow$ *'a literal* $\Rightarrow$ *bool*
where *undefined-lit I L* $\equiv$ $\neg$*defined-lit I L*

lemma *defined-lit-rev*[*simp*]:
defined-lit (*rev M*) *L* $\longleftrightarrow$ *defined-lit M L*
unfolding *defined-lit-def* **by** *auto*

lemma *atm-imp-marked-or-proped*:
assumes $x \in$ *set I*
shows
($\exists$ *l*. *Marked* (− *lit-of x*) *l* $\in$ *set I*)

$\lor$ ($\exists$ *l*. *Marked* (*lit-of x*) *l* $\in$ *set I*)
$\lor$ ($\exists$ *l*. *Propagated* ($-$ *lit-of x*) *l* $\in$ *set I*)
$\lor$ ($\exists$ *l*. *Propagated* (*lit-of x*) *l* $\in$ *set I*)
using *assms ann-literal.exhaust-sel* **by** *metis*

lemma *literal-is-lit-of-marked*:
assumes L = *lit-of x*
shows ($\exists$ *l*. *x* = *Marked L l*) $\lor$ ($\exists$ *l'*. *x* = *Propagated L l'*)
using *assms* **by** (*cases x*) *auto*

lemma *true-annot-iff-marked-or-true-lit*:
defined-lit I L $\longleftrightarrow$ ((*lits-of I*) $\models$*l L* $\lor$ (*lits-of I*) $\models$*l* $-L$)
unfolding *defined-lit-def* **by** (*auto simp add*: *lits-of-def rev-image-eqI*
dest!: *literal-is-lit-of-marked*)

lemma *consistent-interp* (*lits-of I*) $\Longrightarrow$ *I* $\models$*as N* $\Longrightarrow$ *satisfiable N*
by (*simp add*: *true-annots-true-cls*)

lemma *defined-lit-map*:
defined-lit Ls L $\longleftrightarrow$ *atm-of L* $\in$ (λ*l*. *atm-of* (*lit-of l*)) ' *set Ls*
unfolding *defined-lit-def* **apply** (*rule iffI*)
using *image-iff* **apply** *fastforce*
by (*fastforce simp add*: *atm-of-eq-atm-of dest*: *atm-imp-marked-or-proped*)

lemma *defined-lit-uminus*[*iff*]:
defined-lit I ($-L$) $\longleftrightarrow$ *defined-lit I L*
unfolding *defined-lit-def* **by** *auto*

lemma *Marked-Propagated-in-iff-in-lits-of*:
defined-lit I L $\longleftrightarrow$ (*L* $\in$ *lits-of I* $\lor$ $-L$ $\in$ *lits-of I*)
unfolding *lits-of-def defined-lit-def*
by (*auto simp*: *rev-image-eqI*) (*rename-tac x*, *case-tac x*, *auto*)+

lemma *consistent-add-undefined-lit-consistent*[*simp*]:
assumes
consistent-interp (*lits-of Ls*) **and**
undefined-lit Ls L
shows *consistent-interp* (*insert L* (*lits-of Ls*))
using *assms* **unfolding** *consistent-interp-def* **by** (*auto simp*: *Marked-Propagated-in-iff-in-lits-of*)

lemma *decided-empty*[*simp*]:
$\neg$*defined-lit* [] *L*
unfolding *defined-lit-def* **by** *simp*

1.2 Backtracking

fun *backtrack-split* :: (*'v*, *'l*, *'m*) *ann-literals*
$\Rightarrow$ (*'v*, *'l*, *'m*) *ann-literals* $\times$ (*'v*, *'l*, *'m*) *ann-literals* **where**
backtrack-split [] = ([], []) |
backtrack-split (*Propagated L P* # *mlits*) = *apfst* ((*op* #) (*Propagated L P*)) (*backtrack-split mlits*) |
backtrack-split (*Marked L l* # *mlits*) = ([], *Marked L l* # *mlits*)

lemma *backtrack-split-fst-not-marked*: *a* $\in$ *set* (*fst* (*backtrack-split l*)) $\Longrightarrow$ $\neg$*is-marked a*
by (*induct l rule*: *ann-literal-list-induct*) *auto*

lemma *backtrack-split-snd-hd-marked*:

snd (*backtrack-split l*) $\neq$ [] $\Longrightarrow$ *is-marked* (*hd* (*snd* (*backtrack-split l*)))
by (*induct l rule*: *ann-literal-list-induct*) *auto*

lemma *backtrack-split-list-eq*[*simp*]:
fst (*backtrack-split l*) @ (*snd* (*backtrack-split l*)) = *l*
by (*induct l rule*: *ann-literal-list-induct*) *auto*

lemma *backtrack-snd-empty-not-marked*:
backtrack-split M = (*M''*, []) $\Longrightarrow$ $\forall$ *l*$\in$*set M*. $\neg$ *is-marked l*
by (*metis append-Nil2 backtrack-split-fst-not-marked backtrack-split-list-eq snd-conv*)

lemma *backtrack-split-some-is-marked-then-snd-has-hd*:
$\exists$ *l*$\in$*set M*. *is-marked l* $\Longrightarrow$ $\exists$ *M' L' M''*. *backtrack-split M* = (*M''*, *L'* # *M'*)
by (*metis backtrack-snd-empty-not-marked list.exhaust prod.collapse*)

Another characterisation of the result of *backtrack-split*. This view allows some simpler proofs, since *takeWhile* and *dropWhile* are highly automated:

lemma *backtrack-split-takeWhile-dropWhile*:
backtrack-split M = (*takeWhile* (*Not o is-marked*) *M*, *dropWhile* (*Not o is-marked*) *M*)
proof (*induct M*)
case *Nil* **show** *?case* **by** *simp*
next
case (*Cons L M*) **thus** *?case* **by** (*cases L*) *auto*
qed

1.3 Decomposition with respect to the marked literals

The pattern *get-all-marked-decomposition* [] = [([], [])] is necessary otherwise, we can call the *hd* function in the other pattern.

fun *get-all-marked-decomposition* :: (*'a*, *'l*, *'m*) *ann-literals*
$\Rightarrow$ ((*'a*, *'l*, *'m*) *ann-literals* $\times$ (*'a*, *'l*, *'m*) *ann-literals*) *list* **where**
get-all-marked-decomposition (*Marked L l* # *Ls*) =
(*Marked L l* # *Ls*, []) # *get-all-marked-decomposition Ls* |
get-all-marked-decomposition (*Propagated L P*# *Ls*) =
(*apsnd* ((*op* #) (*Propagated L P*)) (*hd* (*get-all-marked-decomposition Ls*)))
tl (*get-all-marked-decomposition Ls*) |
get-all-marked-decomposition [] = [([], [])]

value *get-all-marked-decomposition* [*Propagated A5 B5*, *Marked C4 D4*, *Propagated A3 B3*,
Propagated A2 B2, *Marked C1 D1*, *Propagated A0 B0*]

lemma *get-all-marked-decomposition-never-empty*[*iff*]:
get-all-marked-decomposition M = [] $\longleftrightarrow$ *False*
by (*induct M*, *simp*) (*rename-tac a xs*, *case-tac a*, *auto*)

lemma *get-all-marked-decomposition-never-empty-sym*[*iff*]:
[] = *get-all-marked-decomposition M* $\longleftrightarrow$ *False*
using *get-all-marked-decomposition-never-empty*[*of M*] **by** *presburger*

lemma *get-all-marked-decomposition-decomp*:
hd (*get-all-marked-decomposition S*) = (*a*, *c*) $\Longrightarrow$ *S* = *c* @ *a*
proof (*induct S arbitrary*: *a c*)
case *Nil*
thus *?case* **by** *simp*

next
  case (*Cons x A*)
  thus *?case* **by** (*cases x*; *cases hd* (*get-all-marked-decomposition A*)) *auto*
qed

lemma *get-all-marked-decomposition-backtrack-split*:
  backtrack-split $S = (M, M') \longleftrightarrow$ *hd* (*get-all-marked-decomposition* S) $= (M', M)$
proof (*induction S arbitrary*: M M')
  case *Nil*
  thus *?case* **by** *auto*
next
  case (*Cons a S*)
  thus *?case* **using** *backtrack-split-takeWhile-dropWhile* **by** (*cases a*) *force+*
qed

lemma *get-all-marked-decomposition-nil-backtrack-split-snd-nil*:
  get-all-marked-decomposition $S = [([], A)] \Longrightarrow$ *snd* (*backtrack-split S*) $= []$
  by (*simp add*: *get-all-marked-decomposition-backtrack-split sndI*)

lemma *get-all-marked-decomposition-length-1-fst-empty-or-length-1*:
  assumes *get-all-marked-decomposition* $M = (a, b)$ # []
  shows $a = [] \lor$ (*length* $a = 1 \land$ *is-marked* (*hd a*) $\land$ *hd a* $\in$ *set M*)
  using *assms*
proof (*induct M arbitrary*: *a b*)
  case *Nil* **thus** *?case* **by** *simp*
next
  case (*Cons m M*)
  show *?case*
    proof (*cases m*)
      case (*Marked l mark*)
      thus *?thesis* **using** *Cons* **by** *simp*
    next
      case (*Propagated l mark*)
      thus *?thesis* **using** *Cons* **by** (*cases get-all-marked-decomposition M*) *force+*
    qed
qed

lemma *get-all-marked-decomposition-fst-empty-or-hd-in-M*:
  assumes *get-all-marked-decomposition* $M = (a, b)$ # *l*
  shows $a = [] \lor$ (*is-marked* (*hd a*) $\land$ *hd a* $\in$ *set M*)
  using *assms* **apply** (*induct M arbitrary*: *a b rule*: *ann-literal-list-induct*)
    apply *auto*[*2*]
  by (*metis UnCI backtrack-split-snd-hd-marked get-all-marked-decomposition-backtrack-split*
    get-all-marked-decomposition-decomp hd-in-set list.sel(*1*) *set-append snd-conv*)

lemma *get-all-marked-decomposition-snd-not-marked*:
  assumes $(a, b) \in$ *set* (*get-all-marked-decomposition M*)
  and $L \in$ *set b*
  shows $\neg$*is-marked L*
  using *assms* **apply** (*induct M arbitrary*: *a b rule*: *ann-literal-list-induct*, *simp*)
  by (*rename-tac* L' *l xs a b*, *case-tac get-all-marked-decomposition xs*; *fastforce*)+

lemma *tl-get-all-marked-decomposition-skip-some*:
  assumes $x \in$ *set* (*tl* (*get-all-marked-decomposition M1*))
  shows $x \in$ *set* (*tl* (*get-all-marked-decomposition* (*M0* @ *M1*)))

```
  using assms
  by (induct M0 rule: ann-literal-list-induct)
     (auto simp add: list.set-sel(2))

lemma hd-get-all-marked-decomposition-skip-some:
  assumes (x, y) = hd (get-all-marked-decomposition M1)
  shows (x, y) ∈ set (get-all-marked-decomposition (M0 @ Marked K i # M1))
  using assms
proof (induct M0)
  case Nil
  thus ?case by auto
next
  case (Cons L M0)
  hence xy: (x, y) ∈ set (get-all-marked-decomposition (M0 @ Marked K i # M1)) by blast
  show ?case
    proof (cases L)
      case (Marked l m)
      thus ?thesis using xy by auto
    next
      case (Propagated l m)
      thus ?thesis
        using xy Cons.prems
        by (cases get-all-marked-decomposition (M0 @ Marked K i # M1))
           (auto dest!: get-all-marked-decomposition-decomp
              arg-cong[of get-all-marked-decomposition - - hd])
    qed
qed

lemma get-all-marked-decomposition-snd-union:
  set M = ⋃(set ‘ snd ‘ set (get-all-marked-decomposition M)) ∪ {L |L. is-marked L ∧ L ∈ set M}
  (is ?M M = ?U M ∪ ?Ls M)
proof (induct M arbitrary:)
  case Nil
  thus ?case by simp
next
  case (Cons L M)
  show ?case
    proof (cases L)
      case (Marked a l) note L = this
      hence L ∈ ?Ls (L#M) by auto
      moreover have ?U (L#M) = ?U M unfolding L by auto
      moreover have ?M M = ?U M ∪ ?Ls M using Cons.hyps by auto
      ultimately show ?thesis by auto
    next
      case (Propagated a P)
      thus ?thesis using Cons.hyps by (cases (get-all-marked-decomposition M)) auto
    qed
qed

lemma in-get-all-marked-decomposition-in-get-all-marked-decomposition-prepend:
  (a, b) ∈ set (get-all-marked-decomposition M') ⟹
    ∃ b'. (a, b' @ b) ∈ set (get-all-marked-decomposition (M @ M'))
  apply (induction M rule: ann-literal-list-induct)
    apply (metis append-Nil)
   apply auto[]
```

by (*rename-tac* L' *m xs*, *case-tac get-all-marked-decomposition* (*xs* @ M')) *auto*

lemma *get-all-marked-decomposition-remove-unmarked-length*:
assumes $\forall$ *l* $\in$ *set* M'. $\neg$*is-marked l*
shows *length* (*get-all-marked-decomposition* (M' @ M''))
= *length* (*get-all-marked-decomposition* M'')
using *assms* **by** (*induct* M' *arbitrary*: M'' *rule*: *ann-literal-list-induct*) *auto*

lemma *get-all-marked-decomposition-not-is-marked-length*:
assumes $\forall$ *l* $\in$ *set* M'. $\neg$*is-marked l*
shows *1* + *length* (*get-all-marked-decomposition* (*Propagated* ($-L$) *P* # *M*))
= *length* (*get-all-marked-decomposition* (M' @ *Marked L l* # *M*))
using *assms get-all-marked-decomposition-remove-unmarked-length* **by** *fastforce*

lemma *get-all-marked-decomposition-last-choice*:
assumes *tl* (*get-all-marked-decomposition* (M' @ *Marked L l* # *M*)) $\neq$ []
and $\forall$ *l* $\in$ *set* M'. $\neg$*is-marked l*
and *hd* (*tl* (*get-all-marked-decomposition* (M' @ *Marked L l* # *M*))) = (*M0'*, *M0*)
shows *hd* (*get-all-marked-decomposition* (*Propagated* ($-L$) *P* # *M*)) = (*M0'*, *Propagated* ($-L$) *P* # *M0*)
using *assms* **by** (*induct* M' *rule*: *ann-literal-list-induct*) *auto*

lemma *get-all-marked-decomposition-except-last-choice-equal*:
assumes $\forall$ *l* $\in$ *set* M'. $\neg$*is-marked l*
shows *tl* (*get-all-marked-decomposition* (*Propagated* ($-L$) *P* # *M*))
= *tl* (*tl* (*get-all-marked-decomposition* (M' @ *Marked L l* # *M*)))
using *assms* **by** (*induct* M' *rule*: *ann-literal-list-induct*) *auto*

lemma *get-all-marked-decomposition-hd-hd*:
assumes *get-all-marked-decomposition Ls* = (*M*, *C*) # (*M0*, *M0'*) # *l*
shows *tl M* = *M0'* @ *M0* $\wedge$ *is-marked* (*hd M*)
using *assms*
proof (*induct Ls arbitrary*: *M C M0 M0' l*)
case *Nil*
thus *?case* **by** *simp*
next
case (*Cons a Ls M C M0 M0' l*) **note** *IH* = *this*(*1*) **and** *g* = *this*(*2*)
{ fix *L level*
assume *a*: *a* = *Marked L level*
have *Ls* = *M0'* @ *M0*
using *g a* **by** (*force intro*: *get-all-marked-decomposition-decomp*)
hence *tl M* = *M0'* @ *M0* $\wedge$ *is-marked* (*hd M*) **using** *g a* **by** *auto*
}
moreover {
fix *L P*
assume *a*: *a* = *Propagated L P*
have *tl M* = *M0'* @ *M0* $\wedge$ *is-marked* (*hd M*)
using *IH Cons.prems* **unfolding** *a* **by** (*cases get-all-marked-decomposition Ls*) *auto*
}
ultimately show *?case* **by** (*cases a*) *auto*
qed

lemma *get-all-marked-decomposition-exists-prepend*[*dest*]:
assumes (*a*, *b*) $\in$ *set* (*get-all-marked-decomposition M*)
shows $\exists$ *c*. *M* = *c* @ *b* @ *a*

using *assms* **apply** (*induct M rule*: *ann-literal-list-induct*)
apply *simp*
by (*rename-tac L' m xs*, *case-tac get-all-marked-decomposition xs*;
auto dest!: *arg-cong*[*of get-all-marked-decomposition - - hd*]
get-all-marked-decomposition-decomp)+

lemma *get-all-marked-decomposition-incl*:
assumes (*a*, *b*) ∈ *set* (*get-all-marked-decomposition M*)
shows *set b* ⊆ *set M* **and** *set a* ⊆ *set M*
using *assms get-all-marked-decomposition-exists-prepend* **by** *fastforce*+

lemma *get-all-marked-decomposition-exists-prepend'*:
assumes (*a*, *b*) ∈ *set* (*get-all-marked-decomposition M*)
obtains *c* **where** *M* = *c* @ *b* @ *a*
using *assms* **apply** (*induct M rule*: *ann-literal-list-induct*)
apply *auto*[*1*]
by (*rename-tac L' m xs*, *case-tac hd* (*get-all-marked-decomposition xs*),
auto dest!: *get-all-marked-decomposition-decomp simp add*: *list.set-sel*(*2*))+

lemma *union-in-get-all-marked-decomposition-is-subset*:
assumes (*a*, *b*) ∈ *set* (*get-all-marked-decomposition M*)
shows *set a* ∪ *set b* ⊆ *set M*
using *assms* **by** *force*

definition *all-decomposition-implies* :: *'a literal multiset set*
⇒ ((*'a*, *'l*, *'m*) *ann-literal list* × (*'a*, *'l*, *'m*) *ann-literal list*) *list* ⇒ *bool* **where**
all-decomposition-implies N S
⟷ (∀ (*Ls*, *seen*) ∈ *set S*. *unmark Ls* ∪ *N* ⊨*ps unmark seen*)

lemma *all-decomposition-implies-empty*[*iff*]:
all-decomposition-implies N [] **unfolding** *all-decomposition-implies-def* **by** *auto*

lemma *all-decomposition-implies-single*[*iff*]:
all-decomposition-implies N [(*Ls*, *seen*)]
⟷ *unmark Ls* ∪ *N* ⊨*ps unmark seen*
unfolding *all-decomposition-implies-def* **by** *auto*

lemma *all-decomposition-implies-append*[*iff*]:
all-decomposition-implies N (*S* @ *S'*)
⟷ (*all-decomposition-implies N S* ∧ *all-decomposition-implies N S'*)
unfolding *all-decomposition-implies-def* **by** *auto*

lemma *all-decomposition-implies-cons-pair*[*iff*]:
all-decomposition-implies N ((*Ls*, *seen*) # *S'*)
⟷ (*all-decomposition-implies N* [(*Ls*, *seen*)] ∧ *all-decomposition-implies N S'*)
unfolding *all-decomposition-implies-def* **by** *auto*

lemma *all-decomposition-implies-cons-single*[*iff*]:
all-decomposition-implies N (*l* # *S'*) ⟷
(*unmark* (*fst l*) ∪ *N* ⊨*ps unmark* (*snd l*) ∧
all-decomposition-implies N S')
unfolding *all-decomposition-implies-def* **by** *auto*

lemma *all-decomposition-implies-trail-is-implied*:

```
  assumes all-decomposition-implies N (get-all-marked-decomposition M)
  shows N ∪ {{#lit-of L#} |L. is-marked L ∧ L ∈ set M}
    ⊨ps (λa. {#lit-of a#}) ‘ ⋃(set ‘ snd ‘ set (get-all-marked-decomposition M))
using assms
proof (induct length (get-all-marked-decomposition M) arbitrary: M)
  case 0
  thus ?case by auto
next
  case (Suc n) note IH = this(1) and length = this(2)
  {
    assume length (get-all-marked-decomposition M) ≤ 1
    then obtain a b where g: get-all-marked-decomposition M = (a, b) # []
      by (cases get-all-marked-decomposition M) auto
    moreover {
      assume a = []
      hence ?case using Suc.prems g by auto
    }
    moreover {
      assume l: length a = 1 and m: is-marked (hd a) and hd: hd a ∈ set M
      hence (λa. {#lit-of a#}) (hd a) ∈ {{#lit-of L#} |L. is-marked L ∧ L ∈ set M} by auto
      hence H: unmark a ∪ N ⊆ N ∪ {{#lit-of L#} |L. is-marked L ∧ L ∈ set M}
        using l by (cases a) auto
      have f1: (λm. {#lit-of m#}) ‘ set a ∪ N ⊨ps (λm. {#lit-of m#}) ‘ set b
        using Suc.prems unfolding all-decomposition-implies-def g by simp
      have ?case
        unfolding g apply (rule true-clss-clss-subset) using f1 H by auto
    }
    ultimately have ?case using get-all-marked-decomposition-length-1-fst-empty-or-length-1 by blast
  }
  moreover {
    assume length (get-all-marked-decomposition M) > 1
    then obtain Ls0 seen0 M' where
      Ls0: get-all-marked-decomposition M = (Ls0, seen0) # get-all-marked-decomposition M' and
      length': length (get-all-marked-decomposition M') = n and
      M'-in-M: set M' ⊆ set M
      using length apply (induct M)
        apply simp
      by (rename-tac a M, case-tac a, case-tac hd (get-all-marked-decomposition M))
         (auto simp add: subset-insertI2)
    {
      assume n = 0
      hence get-all-marked-decomposition M' = [] using length' by auto
      hence ?case using Suc.prems unfolding all-decomposition-implies-def Ls0 by auto
    }
    moreover {
      assume n: n > 0
      then obtain Ls1 seen1 l where Ls1: get-all-marked-decomposition M' = (Ls1, seen1) # l
        using length' by (induct M', simp) (rename-tac a xs, case-tac a, auto)

      have all-decomposition-implies N (get-all-marked-decomposition M')
        using Suc.prems unfolding Ls0 all-decomposition-implies-def by auto
      hence N: N ∪ {{#lit-of L#} |L. is-marked L ∧ L ∈ set M'}
          ⊨ps (λa. {#lit-of a#}) ‘ ⋃(set ‘ snd ‘ set (get-all-marked-decomposition M'))
        using IH length' by auto
```

have *l*: *N* ∪ {{#*lit-of L*#} |*L*. *is-marked L* ∧ *L* ∈ *set M'*}
  ⊆ *N* ∪ {{#*lit-of L*#} |*L*. *is-marked L* ∧ *L* ∈ *set M*}
  using *M'-in-M* **by** *auto*
hence Ψ*N*: *N* ∪ {{#*lit-of L*#} |*L*. *is-marked L* ∧ *L* ∈ *set M*}
  ⊨*ps* (λ*a*. {#*lit-of a*#}) ‘ ⋃(*set* ‘ *snd* ‘ *set* (*get-all-marked-decomposition M'*))
  using *true-clss-clss-subset*[*OF l N*] **by** *auto*
have *is-marked* (*hd Ls0*) **and** *LS*: *tl Ls0* = *seen1* @ *Ls1*
  using *get-all-marked-decomposition-hd-hd*[*of M*] **unfolding** *Ls0 Ls1* **by** *auto*

have *LSM*: *seen1* @ *Ls1* = *M'* **using** *get-all-marked-decomposition-decomp*[*of M'*] *Ls1* **by** *auto*
have *M'*: *set M'* = *Union* (*set* ‘ *snd* ‘ *set* (*get-all-marked-decomposition M'*))
  ∪ {*L* |*L*. *is-marked L* ∧ *L* ∈ *set M'*}
  using *get-all-marked-decomposition-snd-union* **by** *auto*

{
  assume *Ls0* ≠ []
  hence *hd Ls0* ∈ *set M* **using** *get-all-marked-decomposition-fst-empty-or-hd-in-M Ls0* **by** *blast*
  hence *N* ∪ {{#*lit-of L*#} |*L*. *is-marked L* ∧ *L* ∈ *set M*} ⊨*p* (λ*a*. {#*lit-of a*#}) (*hd Ls0*)
    using ‹*is-marked* (*hd Ls0*)› **by** (*metis* (*mono-tags*, *lifting*) *UnCI mem-Collect-eq*
      true-clss-cls-in)
} **note** *hd-Ls0* = *this*

have *l*: (λ*a*. {#*lit-of a*#}) ‘ (⋃(*set* ‘ *snd* ‘ *set* (*get-all-marked-decomposition M'*))
    ∪ {*L* |*L*. *is-marked L* ∧ *L* ∈ *set M'*})
  = (λ*a*. {#*lit-of a*#}) ‘
    ⋃(*set* ‘ *snd* ‘ *set* (*get-all-marked-decomposition M'*))
    ∪ {{#*lit-of L*#} |*L*. *is-marked L* ∧ *L* ∈ *set M'*}
  by *auto*
have *N* ∪ {{#*lit-of L*#} |*L*. *is-marked L* ∧ *L* ∈ *set M'*} ⊨*ps*
      (λ*a*. {#*lit-of a*#}) ‘ (⋃(*set* ‘ *snd* ‘ *set* (*get-all-marked-decomposition M'*))
        ∪ {*L* |*L*. *is-marked L* ∧ *L* ∈ *set M'*})
  unfolding *l* **using** *N* **by** (*auto simp add*: *all-in-true-clss-clss*)
hence *N* ∪ {{#*lit-of L*#} |*L*. *is-marked L* ∧ *L* ∈ *set M'*} ⊨*ps unmark* (*tl Ls0*)
  using *M'* **unfolding** *LS LSM* **by** *auto*
hence *t*: *N* ∪ {{#*lit-of L*#} |*L*. *is-marked L* ∧ *L* ∈ *set M'*}
  ⊨*ps unmark* (*tl Ls0*)
  by (*blast intro*: *all-in-true-clss-clss*)
hence *N* ∪ {{#*lit-of L*#} |*L*. *is-marked L* ∧ *L* ∈ *set M*}
  ⊨*ps unmark* (*tl Ls0*)
  using *M'-in-M true-clss-clss-subset*[*OF - t*,
    of N ∪ {{#*lit-of L*#} |*L*. *is-marked L* ∧ *L* ∈ *set M*}]
  by *auto*
hence *N* ∪ {{#*lit-of L*#} |*L*. *is-marked L* ∧ *L* ∈ *set M*} ⊨*ps unmark Ls0*
  using *hd-Ls0* **by** (*cases Ls0*, *auto*)

moreover have *unmark Ls0* ∪ *N* ⊨*ps unmark seen0*
  using *Suc.prems* **unfolding** *Ls0 all-decomposition-implies-def* **by** *simp*
moreover have ⋀*M Ma*. (*M*::'*a literal multiset set*) ∪ *Ma* ⊨*ps M*
  by (*simp add*: *all-in-true-clss-clss*)
ultimately have Ψ: *N* ∪ {{#*lit-of L*#} |*L*. *is-marked L* ∧ *L* ∈ *set M*} ⊨*ps*
    unmark seen0
  by (*meson true-clss-clss-left-right true-clss-clss-union-and true-clss-clss-union-l-r*)
have (λ*a*. {#*lit-of a*#})‘(*set seen0*
    ∪ (⋃*x*∈*set* (*get-all-marked-decomposition M'*). *set* (*snd x*)))
  = *unmark seen0*

$\cup$ ($\lambda a.$ {#*lit-of a*#}) ‘ ($\bigcup x{\in}$*set* (*get-all-marked-decomposition M′*). *set* (*snd x*))
by *auto*

hence *?case* **unfolding** *Ls0* **using** Ψ Ψ*N* **by** *simp*
}
ultimately have *?case* **by** *auto*
}
ultimately show *?case* **by** *arith*
qed

lemma *all-decomposition-implies-propagated-lits-are-implied*:
assumes *all-decomposition-implies N* (*get-all-marked-decomposition M*)
shows *N* ∪ {{#*lit-of L*#} |*L. is-marked L* ∧ *L* ∈ *set M*} ⊨*ps unmark M*
(**is** *?I* ⊨*ps ?A*)
proof −
have *?I* ⊨*ps* (λ*a.* {#*lit-of a*#}) ‘ {*L* |*L. is-marked L* ∧ *L* ∈ *set M*}
by (*auto intro*: *all-in-true-clss-clss*)
moreover have *?I* ⊨*ps* (λ*a.* {#*lit-of a*#}) ‘ ⋃(*set* ‘ *snd* ‘ *set* (*get-all-marked-decomposition M*))
using *all-decomposition-implies-trail-is-implied assms* **by** *blast*
ultimately have *N* ∪ {{#*lit-of m*#} |*m. is-marked m* ∧ *m* ∈ *set M*}
⊨*ps* (λ*m.* {#*lit-of m*#}) ‘ ⋃(*set* ‘ *snd* ‘ *set* (*get-all-marked-decomposition M*))
∪ (λ*m.* {#*lit-of m*#}) ‘ {*m* |*m. is-marked m* ∧ *m* ∈ *set M*}
by *blast*
thus *?thesis*
by (*metis* (*no-types*) *get-all-marked-decomposition-snd-union*[*of M*] *image-Un*)
qed

lemma *all-decomposition-implies-insert-single*:
all-decomposition-implies N M ⟹ *all-decomposition-implies* (*insert C N*) *M*
unfolding *all-decomposition-implies-def* **by** *auto*

1.4 Negation of Clauses

definition *CNot* :: ′*v clause* ⇒ ′*v clauses* **where**
CNot ψ = { {#−*L*#} | *L*. *L* ∈# ψ }

lemma *in-CNot-uminus*[*iff*]:
shows {#*L*#} ∈ *CNot* ψ ⟷ −*L* ∈# ψ
using *assms* **unfolding** *CNot-def* **by** *force*

lemma *CNot-singleton*[*simp*]: *CNot* {#*L*#} = {{#−*L*#}} **unfolding** *CNot-def* **by** *auto*
lemma *CNot-empty*[*simp*]: *CNot* {#} = {} **unfolding** *CNot-def* **by** *auto*
lemma *CNot-plus*[*simp*]: *CNot* (*A* + *B*) = *CNot A* ∪ *CNot B* **unfolding** *CNot-def* **by** *auto*

lemma *CNot-eq-empty*[*iff*]:
CNot D = {} ⟷ *D* = {#}
unfolding *CNot-def* **by** (*auto simp add*: *multiset-eqI*)

lemma *in-CNot-implies-uminus*:
assumes *L* ∈# *D*
and *M* ⊨*as CNot D*
shows *M* ⊨*a* {#−*L*#} **and** −*L* ∈ *lits-of M*
using *assms* **by** (*auto simp add*: *true-annots-def true-annot-def CNot-def*)

lemma *CNot-remdups-mset*[*simp*]:
CNot (*remdups-mset A*) = *CNot A*

unfolding *CNot-def* **by** *auto*

lemma *Ball-CNot-Ball-mset*[*simp*] :
$(\forall x{\in}CNot\ D.\ P\ x) \longleftrightarrow (\forall L{\in}\#\ D.\ P\ \{\#{-}L\#\})$
unfolding *CNot-def* **by** *auto*

lemma *consistent-CNot-not*:
assumes *consistent-interp I*
shows $I \models s\ CNot\ \varphi \Longrightarrow \neg I \models \varphi$
using *assms* **unfolding** *consistent-interp-def true-clss-def true-cls-def* **by** *auto*

lemma *total-not-true-cls-true-clss-CNot*:
assumes *total-over-m I* $\{\varphi\}$ **and** $\neg I \models \varphi$
shows $I \models s\ CNot\ \varphi$
using *assms* **unfolding** *total-over-m-def total-over-set-def true-clss-def true-cls-def CNot-def*
apply *clarify*
by (*rename-tac x L*, *case-tac L*) (*force intro*: *pos-lit-in-atms-of neg-lit-in-atms-of*)+

lemma *total-not-CNot*:
assumes *total-over-m I* $\{\varphi\}$ **and** $\neg I \models s\ CNot\ \varphi$
shows $I \models \varphi$
using *assms total-not-true-cls-true-clss-CNot* **by** *auto*

lemma *atms-of-ms-CNot-atms-of*[*simp*]:
atms-of-ms (*CNot C*) = *atms-of C*
unfolding *atms-of-ms-def atms-of-def CNot-def* **by** *fastforce*

lemma *true-clss-clss-contradiction-true-clss-cls-false*:
$C \in D \Longrightarrow D \models ps\ CNot\ C \Longrightarrow D \models p\ \{\#\}$
unfolding *true-clss-clss-def true-clss-cls-def total-over-m-def*
by (*metis Un-commute atms-of-empty atms-of-ms-CNot-atms-of atms-of-ms-insert atms-of-ms-union consistent-CNot-not insert-absorb sup-bot.left-neutral true-clss-def*)

lemma *true-annots-CNot-all-atms-defined*:
assumes $M \models as\ CNot\ T$ **and** *a1*: $L \in\#\ T$
shows *atm-of L* $\in$ *atm-of* ‘ *lits-of M*
by (*metis assms atm-of-uminus image-eqI in-CNot-implies-uminus*(*1*) *true-annot-singleton*)

lemma *true-clss-clss-false-left-right*:
assumes $\{\{\#L\#\}\} \cup B \models p\ \{\#\}$
shows $B \models ps\ CNot\ \{\#L\#\}$
unfolding *true-clss-clss-def true-clss-cls-def*
proof (*intro allI impI*)
fix *I*
assume
tot: *total-over-m I* $(B \cup CNot\ \{\#L\#\})$ **and**
cons: *consistent-interp I* **and**
I: $I \models s\ B$
have *total-over-m I* $(\{\{\#L\#\}\} \cup B)$ **using** *tot* **by** *auto*
hence $\neg I \models s$ *insert* $\{\#L\#\}\ B$
using *assms cons* **unfolding** *true-clss-cls-def* **by** *simp*
thus $I \models s\ CNot\ \{\#L\#\}$
using *tot I* **by** (*cases L*) *auto*
qed

lemma *true-annots-true-cls-def-iff-negation-in-model*:
  $M \models as\ CNot\ C \longleftrightarrow (\forall L \in\#\ C.\ -L \in lits\text{-}of\ M)$
  unfolding *CNot-def true-annots-true-cls true-clss-def* **by** *auto*

lemma *consistent-CNot-not-tautology*:
  $consistent\text{-}interp\ M \Longrightarrow M \models s\ CNot\ D \Longrightarrow \neg tautology\ D$
  by (*metis atms-of-ms-CNot-atms-of consistent-CNot-not satisfiable-carac′ satisfiable-def tautology-def total-over-m-def*)

lemma *atms-of-ms-CNot-atms-of-ms*: $atms\text{-}of\text{-}ms\ (CNot\ CC) = atms\text{-}of\text{-}ms\ \{CC\}$
  by *simp*

lemma *total-over-m-CNot-toal-over-m*[*simp*]:
  $total\text{-}over\text{-}m\ I\ (CNot\ C) = total\text{-}over\text{-}set\ I\ (atms\text{-}of\ C)$
  unfolding *total-over-m-def total-over-set-def* **by** *auto*

lemma *uminus-lit-swap*: $-(a::'a\ literal) = i \longleftrightarrow a = -i$
  by *auto*

lemma *true-clss-cls-plus-CNot*:
  assumes *CC-L*: $A \models p\ CC + \{\#L\#\}$
  and *CNot-CC*: $A \models ps\ CNot\ CC$
  shows $A \models p\ \{\#L\#\}$
  unfolding *true-clss-clss-def true-clss-cls-def CNot-def total-over-m-def*
proof (*intro allI impI*)
  fix I
  assume *tot*: $total\text{-}over\text{-}set\ I\ (atms\text{-}of\text{-}ms\ (A \cup \{\{\#L\#\}\}))$
  and *cons*: $consistent\text{-}interp\ I$
  and I: $I \models s\ A$
  let $?I = I \cup \{Pos\ P|P.\ P \in atms\text{-}of\ CC \wedge P \notin atm\text{-}of\ `\ I\}$
  have *cons′*: $consistent\text{-}interp\ ?I$
    using *cons* **unfolding** *consistent-interp-def*
    by (*auto simp add*: *uminus-lit-swap atms-of-def rev-image-eqI*)
  have I': $?I \models s\ A$
    using *I true-clss-union-increase* **by** *blast*
  have *tot-CNot*: $total\text{-}over\text{-}m\ ?I\ (A \cup CNot\ CC)$
    using *tot atms-of-s-def* **by** (*fastforce simp add*: *total-over-m-def total-over-set-def*)

  hence *tot-I-A-CC-L*: $total\text{-}over\text{-}m\ ?I\ (A \cup \{CC + \{\#L\#\}\})$
    using *tot* **unfolding** *total-over-m-def total-over-set-atm-of* **by** *auto*
  hence $?I \models CC + \{\#L\#\}$ **using** *CC-L cons′ I′* **unfolding** *true-clss-cls-def* **by** *blast*
  moreover
    have $?I \models s\ CNot\ CC$ **using** *CNot-CC cons′ I′ tot-CNot* **unfolding** *true-clss-clss-def* **by** *auto*
    hence $\neg A \models p\ CC$
      by (*metis* (*no-types*, *lifting*) *I′ atms-of-ms-CNot-atms-of-ms atms-of-ms-union cons′ consistent-CNot-not tot-CNot total-over-m-def true-clss-cls-def*)
    hence $\neg ?I \models CC$ **using** ‹$?I \models s\ CNot\ CC$› *cons′ consistent-CNot-not* **by** *blast*
  ultimately have $?I \models \{\#L\#\}$ **by** *blast*
  thus $I \models \{\#L\#\}$
    by (*metis* (*no-types*, *lifting*) *atms-of-ms-union cons′ consistent-CNot-not tot total-not-CNot total-over-m-def total-over-set-union true-clss-union-increase*)
qed

lemma *true-annots-CNot-lit-of-notin-skip*:
  assumes *LM*: $L\ \#\ M \models as\ CNot\ A$ **and** *LA*: $lit\text{-}of\ L \notin\#\ A\ -lit\text{-}of\ L \notin\#\ A$

shows *M* $\models$*as CNot A*
using *LM* **unfolding** *true-annots-def Ball-def*
proof (*intro allI impI*)
fix *l*
assume *H*: $\forall x.\ x \in$ *CNot A* $\longrightarrow$ *L* # *M* $\models$*a x* **and** *l*: *l* $\in$ *CNot A*
hence *L* # *M* $\models$*a l* **by** *auto*
thus *M* $\models$*a l* **using** *LA l* **by** (*cases L*) (*auto simp add*: *CNot-def*)
qed

lemma *true-clss-clss-union-false-true-clss-clss-cnot*:
A $\cup$ {*B*} $\models$*ps* {{#}} $\longleftrightarrow$ *A* $\models$*ps CNot B*
using *total-not-CNot consistent-CNot-not* **unfolding** *total-over-m-def true-clss-clss-def*
by *fastforce*

lemma *true-annot-remove-hd-if-notin-vars*:
assumes *a* # *M′*$\models$*a D*
and *atm-of* (*lit-of a*) $\notin$ *atms-of D*
shows *M′* $\models$*a D*
using *assms true-cls-remove-hd-if-notin-vars* **unfolding** *true-annot-def* **by** *auto*

lemma *true-annot-remove-if-notin-vars*:
assumes *M* @ *M′*$\models$*a D*
and $\forall$*x*$\in$*atms-of D*. *x* $\notin$ *atm-of* ‘ *lits-of M*
shows *M′* $\models$*a D*
using *assms* **apply** (*induct M*, *simp*)
using *true-annot-remove-hd-if-notin-vars* **by** *force+*

lemma *true-annots-remove-if-notin-vars*:
assumes *M* @ *M′*$\models$*as D*
and $\forall$*x*$\in$*atms-of-ms D*. *x* $\notin$ *atm-of* ‘ *lits-of M*
shows *M′* $\models$*as D* **unfolding** *true-annots-def*
using *assms true-annot-remove-if-notin-vars*[*of M M′*]
unfolding *true-annots-def atms-of-ms-def* **by** *force*

lemma *all-variables-defined-not-imply-cnot*:
assumes $\forall$ *s* $\in$ *atms-of-ms* {*B*}. *s* $\in$ *atm-of* ‘ *lits-of A*
and ¬ *A* $\models$*a B*
shows *A* $\models$*as CNot B*
unfolding *true-annot-def true-annots-def Ball-def CNot-def true-lit-def*
proof (*clarify*, *rule ccontr*)
fix *L*
assume *LB*: *L* $\in$# *B* **and** ¬ *lits-of A* $\models$*l* − *L*
hence *atm-of L* $\in$ *atm-of* ‘ *lits-of A*
using *assms*(*1*) **by** (*simp add*: *atm-of-lit-in-atms-of lits-of-def*)
hence *L* $\in$ *lits-of A* $\vee$ −*L* $\in$ *lits-of A*
using *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set* **by** *metis*
hence *L* $\in$ *lits-of A* **using** ‹ ¬ *lits-of A* $\models$*l* − *L*› **by** *auto*
thus *False*
using *LB assms*(*2*) **unfolding** *true-annot-def true-lit-def true-cls-def Bex-mset-def*
by *blast*
qed

lemma *CNot-union-mset*[*simp*]:
CNot (*A* #$\cup$ *B*) = *CNot A* $\cup$ *CNot B*
unfolding *CNot-def* **by** *auto*

1.5 Other

abbreviation *no-dup L* ≡ *distinct* (*map* (*λl. atm-of* (*lit-of l*)) *L*)

lemma *no-dup-rev*[*simp*]:
  no-dup (*rev M*) ⟷ *no-dup M*
  by (*auto simp*: *rev-map*[*symmetric*])

lemma *no-dup-length-eq-card-atm-of-lits-of*:
  assumes *no-dup M*
  shows *length M* = *card* (*atm-of* ' *lits-of M*)
  using *assms* **unfolding** *lits-of-def* **by** (*induct M*) (*auto simp add*: *image-image*)

lemma *distinctconsistent-interp*:
  no-dup M ⟹ *consistent-interp* (*lits-of M*)
proof (*induct M*)
  case *Nil*
  show *?case* **by** *auto*
next
  case (*Cons L M*)
  hence *a1*: *consistent-interp* (*lits-of M*) **by** *auto*
  have *a2*: *atm-of* (*lit-of L*) ∉ (*λl. atm-of* (*lit-of l*)) ' *set M* **using** *Cons.prems* **by** *auto*
  have *undefined-lit M* (*lit-of L*)
    using *a2 image-iff* **unfolding** *defined-lit-def* **by** *fastforce*
  thus *?case*
    using *a1* **by** *simp*
qed

lemma *distinct-get-all-marked-decomposition-no-dup*:
  assumes (*a*, *b*) ∈ *set* (*get-all-marked-decomposition M*)
  and *no-dup M*
  shows *no-dup* (*a* @ *b*)
  using *assms* **by** *force*

lemma *true-annots-lit-of-notin-skip*:
  assumes *L* # *M* ⊨*as CNot A*
  and −*lit-of L* ∉# *A*
  and *no-dup* (*L* # *M*)
  shows *M* ⊨*as CNot A*
proof −
  have ∀ *l* ∈# *A*. −*l* ∈ *lits-of* (*L* # *M*)
    using *assms*(*1*) *in-CNot-implies-uminus*(*2*) **by** *blast*
  moreover
    have *atm-of* (*lit-of L*) ∉ *atm-of* ' *lits-of M*
      using *assms*(*3*) **unfolding** *lits-of-def* **by** *force*
    hence − *lit-of L* ∉ *lits-of M* **unfolding** *lits-of-def*
      by (*metis* (*no-types*) *atm-of-uminus imageI*)
  ultimately have ∀ *l* ∈# *A*. −*l* ∈ *lits-of M*
    using *assms*(*2*) **unfolding** *Ball-mset-def* **by** (*metis insertE lits-of-cons uminus-of-uminus-id*)
  thus *?thesis* **by** (*auto simp add*: *true-annots-def*)
qed

type-synonym *'v clauses* = *'v clause multiset*

abbreviation *true-annots-mset* (**infix** ⊨*asm 50*) **where**
I ⊨*asm C* ≡ *I* ⊨*as* (*set-mset C*)

abbreviation *true-clss-clss-m*:: *'a clauses* ⇒ *'a clauses* ⇒ *bool* (**infix** ⊨*psm 50*) **where**
I ⊨*psm C* ≡ *set-mset I* ⊨*ps* (*set-mset C*)

Analog of ⟦*?N* ⊨*ps ?B*; *?A* ⊆ *?B*⟧ ⟹ *?N* ⊨*ps ?A*

lemma *true-clss-clssm-subsetE*: *N* ⊨*psm B* ⟹ *A* ⊆# *B* ⟹ *N* ⊨*psm A*
using *set-mset-mono true-clss-clss-subsetE* **by** *blast*

abbreviation *true-clss-cls-m*:: *'a clauses* ⇒ *'a clause* ⇒ *bool* (**infix** ⊨*pm 50*) **where**
I ⊨*pm C* ≡ *set-mset I* ⊨*p C*

abbreviation *distinct-mset-mset* :: *'a multiset multiset* ⇒ *bool* **where**
distinct-mset-mset Σ ≡ *distinct-mset-set* (*set-mset* Σ)

abbreviation *all-decomposition-implies-m* **where**
all-decomposition-implies-m A B ≡ *all-decomposition-implies* (*set-mset A*) *B*

abbreviation *atms-of-msu* **where**
atms-of-msu U ≡ *atms-of-ms* (*set-mset U*)

abbreviation *true-clss-m*:: *'a interp* ⇒ *'a clauses* ⇒ *bool* (**infix** ⊨*sm 50*) **where**
I ⊨*sm C* ≡ *I* ⊨*s set-mset C*

abbreviation *true-clss-ext-m* (**infix** ⊨*sextm 49*) **where**
I ⊨*sextm C* ≡ *I* ⊨*sext set-mset C*
end
theory *CDCL-NOT*
imports *Partial-Annotated-Clausal-Logic List-More Wellfounded-More Partial-Clausal-Logic*
begin

2 NOT's CDCL

declare *set-mset-minus-replicate-mset*[*simp*]

2.1 Auxiliary Lemmas and Measure

lemma *no-dup-cannot-not-lit-and-uminus*:
no-dup M ⟹ − *lit-of xa* = *lit-of x* ⟹ *x* ∈ *set M* ⟹ *xa* ∉ *set M*
by (*metis atm-of-uminus distinct-map inj-on-eq-iff uminus-not-id'*)

lemma *true-clss-single-iff-incl*:
I ⊨*s single* ‘ *B* ⟷ *B* ⊆ *I*
unfolding *true-clss-def* **by** *auto*

lemma *atms-of-ms-single-atm-of*[*simp*]:
atms-of-ms {{#*lit-of L*#} |*L*. *P L*} = *atm-of* ‘ {*lit-of L* |*L*. *P L*}
unfolding *atms-of-ms-def* **by** *auto*

lemma *atms-of-uminus-lit-atm-of-lit-of*:
atms-of {#− *lit-of x*. *x* ∈# *A*#} = *atm-of* ‘ (*lit-of* ‘ (*set-mset A*))
unfolding *atms-of-def* **by** (*auto simp add*: *Fun.image-comp*)

lemma *atms-of-ms-single-image-atm-of-lit-of*:
atms-of-ms ((λ*x*. {#*lit-of x*#}) ‘ *A*) = *atm-of* ‘ (*lit-of* ‘ *A*)
unfolding *atms-of-ms-def* **by** *auto*

This measure can also be seen as the increasing lexicographic order: it is an order on bounded sequences, when each element is bounded. The proof involves a measure like the one defined here (the same?).

definition μ_C :: *nat* ⇒ *nat* ⇒ *nat list* ⇒ *nat* **where**
μ_C *s b M* ≡ ($\sum$ *i=0..<length M. M!i* ∗ *b^* (*s* +*i* − *length M*))

lemma μ_C-*nil*[*simp*]:
  μ_C *s b* [] = *0*
  unfolding μ_C-*def* **by** *auto*

lemma μ_C-*single*[*simp*]:
  μ_C *s b* [*L*] = *L* ∗ *b* ^ (*s* − *Suc 0*)
  unfolding μ_C-*def* **by** *auto*

lemma *set-sum-atLeastLessThan-add*:
  ($\sum$ *i=k..<k+*(*b::nat*). *f i*) = ($\sum$ *i=0..<b. f* (*k+ i*))
  by (*induction b*) *auto*

lemma *set-sum-atLeastLessThan-Suc*:
  ($\sum$ *i=1..<Suc j. f i*) = ($\sum$ *i=0..<j. f* (*Suc i*))
  using *set-sum-atLeastLessThan-add*[*of - 1 j*] **by** *force*

lemma μ_C-*cons*:
  μ_C *s b* (*L* # *M*) = *L* ∗ *b* ^ (*s* − *1* − *length M*) + μ_C *s b M*
proof −
  have μ_C *s b* (*L* # *M*) = ($\sum$ *i=0..<length* (*L#M*). (*L#M*)!*i* ∗ *b^* (*s* +*i* − *length* (*L#M*)))
    unfolding μ_C-*def* **by** *blast*
  also have . . . = ($\sum$ *i=0..<1.* (*L#M*)!*i* ∗ *b^* (*s* +*i* − *length* (*L#M*)))
                + ($\sum$ *i=1..<length* (*L#M*). (*L#M*)!*i* ∗ *b^* (*s* +*i* − *length* (*L#M*)))
     by (*rule setsum-add-nat-ivl*[*symmetric*]) *simp-all*
  finally have μ_C *s b* (*L* # *M*)= *L* ∗ *b* ^ (*s* − *1* − *length M*)
                + ($\sum$ *i=1..<length* (*L#M*). (*L#M*)!*i* ∗ *b^* (*s* +*i* − *length* (*L#M*)))
     by *auto*
  moreover {
    have ($\sum$ *i=1..<length* (*L#M*). (*L#M*)!*i* ∗ *b^* (*s* +*i* − *length* (*L#M*))) =
           ($\sum$ *i=0..<length* (*M*). (*L#M*)!(*Suc i*) ∗ *b^* (*s* + (*Suc i*) − *length* (*L#M*)))
     unfolding *length-Cons set-sum-atLeastLessThan-Suc* **by** *blast*
    also have . . . = ($\sum$ *i=0..<length* (*M*). *M*!*i* ∗ *b^* (*s* + *i* − *length M*))
      by *auto*
    finally have ($\sum$ *i=1..<length* (*L#M*). (*L#M*)!*i* ∗ *b^* (*s* +*i* − *length* (*L#M*))) = μ_C *s b M*
      unfolding μ_C-*def* **.**
    }
  ultimately show *?thesis* **by** *presburger*
qed

lemma μ_C-*append*:
  assumes *s* ≥ *length* (*M@M'*)
  shows μ_C *s b* (*M@M'*) = μ_C (*s* − *length M'*) *b M* + μ_C *s b M'*
proof −
  have μ_C *s b* (*M@M'*) = ($\sum$ *i=0..<length* (*M@M'*). (*M@M'*)!*i* ∗ *b^* (*s* +*i* − *length* (*M@M'*)))
    unfolding μ_C-*def* **by** *blast*
  moreover then have . . . = ($\sum$ *i=0..< length M.* (*M@M'*)!*i* ∗ *b^* (*s* +*i* − *length* (*M@M'*)))
                + ($\sum$ *i=length M..<length* (*M@M'*). (*M@M'*)!*i* ∗ *b^* (*s* +*i* − *length* (*M@M'*)))
    by (*auto intro*!: *setsum-add-nat-ivl*[*symmetric*])
  moreover

have $\forall\, i \in \{0..< length\ M\}.\ (M@M')!i * b\hat{}\ (s + i - length\ (M@M')) = M\ !\ i * b\ \hat{}\ (s - length\ M' + i - length\ M)$
using ‹$s \geq length\ (M@M')$› **by** (*auto simp add*: *nth-append ac-simps*)
then have $\mu_C\ (s - length\ M')\ b\ M = (\sum i{=}0..< length\ M.\ (M@M')!i * b\hat{}\ (s + i - length\ (M@M')))$
unfolding μ_C*-def* **by** *auto*
ultimately have $\mu_C\ s\ b\ (M@M') = \mu_C\ (s - length\ M')\ b\ M + (\sum i{=}length\ M..<length\ (M@M').\ (M@M')!i * b\hat{}\ (s + i - length\ (M@M')))$
by *auto*
moreover {
have $(\sum i{=}length\ M..<length\ (M@M').\ (M@M')!i * b\hat{}\ (s + i - length\ (M@M'))) = (\sum i{=}0..<length\ M'.\ M'!i * b\hat{}\ (s + i - length\ M'))$
unfolding *length-append set-sum-atLeastLessThan-add* **by** *auto*
then have $(\sum i{=}length\ M..<length\ (M@M').\ (M@M')!i * b\hat{}\ (s + i - length\ (M@M'))) = \mu_C\ s\ b\ M'$
unfolding μ_C*-def* **.**
}
ultimately show *?thesis* **by** *presburger*
qed

lemma μ_C*-cons-non-empty-inf*:
assumes *M-ge-1*: $\forall\, i \in set\ M.\ i \geq 1$ **and** *M*: $M \neq []$
shows $\mu_C\ s\ b\ M \geq b\ \hat{}\ (s - length\ M)$
using *assms* **by** (*cases M*) (*auto simp*: *mult-eq-if* μ_C*-cons*)

Duplicate of " /src/HOL/ex/NatSum.thy" (but generalized to $(0::'a) \leq k$)

lemma *sum-of-powers*: $0 \leq k \Longrightarrow (k - 1) * (\sum i{=}0..<n.\ k\hat{}i) = k\hat{}n - (1::nat)$
apply (*cases* $k = 0$)
apply (*cases n*; *simp*)
by (*induct n*) (*auto simp*: *Nat.nat-distrib*)

In the degenerated cases, we only have the large inequality holds. In the other cases, the following strict inequality holds:

lemma μ_C*-bounded-non-degenerated*:
fixes *b* ::*nat*
assumes
$b > 0$ **and**
$M \neq []$ **and**
M-le: $\forall\, i < length\ M.\ M!i < b$ **and**
$s \geq length\ M$
shows $\mu_C\ s\ b\ M < b\hat{}s$
proof −
consider (*b1*) $b = 1$ | (*b*) $b > 1$ **using** ‹$b>0$› **by** (*cases b*) *auto*
then show *?thesis*
proof *cases*
case *b1*
then have $\forall\, i < length\ M.\ M!i = 0$ **using** *M-le* **by** *auto*
then have $\mu_C\ s\ b\ M = 0$ **unfolding** μ_C*-def* **by** *auto*
then show *?thesis* **using** ‹$b > 0$› **by** *auto*
next
case *b*
have $\forall\ i \in \{0..<length\ M\}.\ M!i * b\hat{}\ (s + i - length\ M) \leq (b-1) * b\hat{}\ (s + i - length\ M)$
using *M-le* ‹$b > 1$› **by** *auto*
then have $\mu_C\ s\ b\ M \leq\ (\sum i{=}0..<length\ M.\ (b-1) * b\hat{}\ (s + i - length\ M))$
using ‹$M \neq []$› ‹$b>0$› **unfolding** μ_C*-def* **by** (*auto intro*: *setsum-mono*)

also
have $\forall\ i \in \{0..<length\ M\}.\ (b-1) * b\hat{\ } (s + i - length\ M) = (b-1) * b\hat{\ } i * b\hat{\ }(s - length\ M)$
by $(metis\ Nat.add\text{-}diff\text{-}assoc2\ add.commute\ assms(4)\ mult.assoc\ power\text{-}add)$
then have $(\sum i=0..<length\ M.\ (b-1) * b\hat{\ } (s + i - length\ M))$
$= (\sum i=0..<length\ M.\ (b-1) * b\hat{\ } i * b\hat{\ }(s - length\ M))$
by $(auto\ simp\ add\colon ac\text{-}simps)$
also have $\ldots = (\sum i=0..<length\ M.\ b\hat{\ } i) * b\hat{\ }(s - length\ M) * (b-1)$
by $(simp\ add\colon setsum\text{-}left\text{-}distrib\ setsum\text{-}right\text{-}distrib\ ac\text{-}simps)$
finally have $\mu_C\ s\ b\ M \leq (\sum i=0..<length\ M.\ b\hat{\ } i) * (b-1) * b\hat{\ }(s - length\ M)$
by $(simp\ add\colon ac\text{-}simps)$

also
have $(\sum i=0..<length\ M.\ b\hat{\ } i) * (b-1) = b\ \hat{\ }\ (length\ M) - 1$
using $sum\text{-}of\text{-}powers[of\ b\ length\ M]\ \langle b>1\rangle$
by $(auto\ simp\ add\colon ac\text{-}simps)$
finally have $\mu_C\ s\ b\ M \leq (b\ \hat{\ }\ (length\ M) - 1) * b\ \hat{\ }\ (s - length\ M)$
by *auto*
also have $\ldots < b\ \hat{\ }\ (length\ M) * b\ \hat{\ }\ (s - length\ M)$
using $\langle b>1\rangle$ **by** *auto*
also have $\ldots = b\ \hat{\ }\ s$
by $(metis\ assms(4)\ le\text{-}add\text{-}diff\text{-}inverse\ power\text{-}add)$
finally show *?thesis* **unfolding** μ_C*-def* **by** $(auto\ simp\ add\colon ac\text{-}simps)$
qed
qed

In the degenerate case $b = (0::'a)$, the list M is empty (since the list cannot contain any element).

lemma μ_C*-bounded*:
fixes $b :: nat$
assumes
$M\text{-}le\colon \forall i < length\ M.\ M!i < b$ **and**
$s \geq length\ M$
$b > 0$
shows $\mu_C\ s\ b\ M < b\ \hat{\ }\ s$
proof –
consider $(M0)\ M = []\ |\ (M)\ b > 0$ **and** $M \neq []$
using *M-le* **by** $(cases\ b,\ cases\ M)\ auto$
then show *?thesis*
proof *cases*
case *M0*
then show *?thesis* **using** *M-le* $\langle b > 0\rangle$ **by** *auto*
next
case *M*
show *?thesis* **using** $\mu_C\text{-}bounded\text{-}non\text{-}degenerated[OF\ M\ assms(1,2)]$ **by** *arith*
qed
qed

When $b = 0$, we cannot show that the measure is empty, since $0^0 = 1$.

lemma μ_C*-base-0*:
assumes $length\ M \leq s$
shows $\mu_C\ s\ 0\ M \leq M!0$
proof –
{
assume $s = length\ M$
moreover {

    fix *n*
    have ($\sum$ *i=0..<n. M ! i* $*$ (*0::nat*) ^ *i*) $\leq$ *M ! 0*
      apply (*induction n rule*: *nat-induct*)
      by *simp* (*rename-tac n*, *case-tac n*, *auto*)
  }
  ultimately have *?thesis* **unfolding** μ_C*-def* **by** *auto*
}
moreover
{
  assume *length M* $<$ *s*
  then have μ_C *s 0 M* = *0* **unfolding** μ_C*-def* **by** *auto***}**
ultimately show *?thesis* **using** *assms* **unfolding** μ_C*-def* **by** *linarith*
qed

2.2 Initial definitions

2.2.1 The state

We define here an abstraction over operation on the state we are manipulating.

locale *dpll-state* =
  fixes
    trail :: *'st* ⇒ (*'v*, *unit*, *unit*) *ann-literals* **and**
    clauses :: *'st* ⇒ *'v clauses* **and**
    prepend-trail :: (*'v*, *unit*, *unit*) *ann-literal* ⇒ *'st* ⇒ *'st* **and**
    tl-trail :: *'st* ⇒*'st* **and**
    add-cls$_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
    remove-cls$_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st*
  assumes
    trail-prepend-trail[*simp*]:
      ⋀*st L*. *undefined-lit* (*trail st*) (*lit-of L*) ⟹ *trail* (*prepend-trail L st*) = *L* # *trail st*
      and
    tl-trail[*simp*]: *trail* (*tl-trail S*) = *tl* (*trail S*) **and**
    trail-add-cls$_{NOT}$[*simp*]: ⋀*st C*. *no-dup* (*trail st*) ⟹ *trail* (*add-cls*$_{NOT}$ *C st*) = *trail st* **and**
    trail-remove-cls$_{NOT}$[*simp*]: ⋀*st C*. *trail* (*remove-cls*$_{NOT}$ *C st*) = *trail st* **and**

    clauses-prepend-trail[*simp*]:
      ⋀*st L*. *undefined-lit* (*trail st*) (*lit-of L*) ⟹ *clauses* (*prepend-trail L st*) = *clauses st*
      and
    clauses-tl-trail[*simp*]: ⋀*st*. *clauses* (*tl-trail st*) = *clauses st* **and**
    clauses-add-cls$_{NOT}$[*simp*]:
      ⋀*st C*. *no-dup* (*trail st*) ⟹ *clauses* (*add-cls*$_{NOT}$ *C st*) = {#*C*#} + *clauses st* **and**
    clauses-remove-cls$_{NOT}$[*simp*]: ⋀*st C*. *clauses* (*remove-cls*$_{NOT}$ *C st*) = *remove-mset C* (*clauses st*)
begin

function *reduce-trail-to*$_{NOT}$ :: *'a list* ⇒ *'st* ⇒ *'st* **where**
reduce-trail-to$_{NOT}$ *F S* =
  (*if length* (*trail S*) = *length F* ∨ *trail S* = [] *then S else reduce-trail-to*$_{NOT}$ *F* (*tl-trail S*))
by *fast+*
termination by (*relation measure* (λ(-, *S*). *length* (*trail S*))) *auto*
declare *reduce-trail-to*$_{NOT}$*.simps*[*simp del*]

lemma
  shows
  reduce-trail-to$_{NOT}$*-nil*[*simp*]: *trail S* = [] ⟹ *reduce-trail-to*$_{NOT}$ *F S* = *S* **and**
  reduce-trail-to$_{NOT}$*-eq-length*[*simp*]: *length* (*trail S*) = *length F* ⟹ *reduce-trail-to*$_{NOT}$ *F S* = *S*

by (*auto simp*: *reduce-trail-to$_{NOT}$.simps*)

lemma *reduce-trail-to$_{NOT}$-length-ne*[*simp*]:
length (*trail S*) $\neq$ *length F* $\Longrightarrow$ *trail S* $\neq$ [] $\Longrightarrow$
reduce-trail-to$_{NOT}$ F S = *reduce-trail-to$_{NOT}$ F* (*tl-trail S*)
by (*auto simp*: *reduce-trail-to$_{NOT}$.simps*)

lemma *trail-reduce-trail-to$_{NOT}$-length-le*:
assumes *length F* > *length* (*trail S*)
shows *trail* (*reduce-trail-to$_{NOT}$ F S*) = []
using *assms* **by** (*induction F S rule*: *reduce-trail-to$_{NOT}$.induct*)
(*simp add*: *less-imp-diff-less reduce-trail-to$_{NOT}$.simps*)

lemma *trail-reduce-trail-to$_{NOT}$-nil*[*simp*]:
trail (*reduce-trail-to$_{NOT}$* [] *S*) = []
by (*induction* [] *S rule*: *reduce-trail-to$_{NOT}$.induct*)
(*simp add*: *less-imp-diff-less reduce-trail-to$_{NOT}$.simps*)

lemma *clauses-reduce-trail-to$_{NOT}$-nil*:
clauses (*reduce-trail-to$_{NOT}$* [] *S*) = *clauses S*
by (*induction* [] *S rule*: *reduce-trail-to$_{NOT}$.induct*)
(*simp add*: *less-imp-diff-less reduce-trail-to$_{NOT}$.simps*)

lemma *trail-reduce-trail-to$_{NOT}$-drop*:
trail (*reduce-trail-to$_{NOT}$ F S*) =
(*if length* (*trail S*) $\geq$ *length F*
then drop (*length* (*trail S*) $-$ *length F*) (*trail S*)
else [])
apply (*induction F S rule*: *reduce-trail-to$_{NOT}$.induct*)
apply (*rename-tac F S*, *case-tac trail S*)
apply *auto*[]
apply (*rename-tac list*, *case-tac Suc* (*length list*) > *length F*)
prefer *2* **apply** *simp*
apply (*subgoal-tac Suc* (*length list*) $-$ *length F* = *Suc* (*length list* $-$ *length F*))
apply *simp*
apply *simp*
done

lemma *reduce-trail-to$_{NOT}$-skip-beginning*:
assumes *trail S* = *F'* @ *F*
shows *trail* (*reduce-trail-to$_{NOT}$ F S*) = *F*
using *assms* **by** (*auto simp*: *trail-reduce-trail-to$_{NOT}$-drop*)

lemma *reduce-trail-to$_{NOT}$-clauses*[*simp*]:
clauses (*reduce-trail-to$_{NOT}$ F S*) = *clauses S*
by (*induction F S rule*: *reduce-trail-to$_{NOT}$.induct*)
(*simp add*: *less-imp-diff-less reduce-trail-to$_{NOT}$.simps*)

abbreviation *trail-weight* **where**
trail-weight S $\equiv$ *map* ((λl. *1* + *length l*) *o snd*) (*get-all-marked-decomposition* (*trail S*))

definition *state-eq$_{NOT}$* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* (**infix** $\sim$ *50*) **where**
S $\sim$ *T* $\longleftrightarrow$ *trail S* = *trail T* $\wedge$ *clauses S* = *clauses T*

lemma *state-eq*$_{NOT}$*-ref*[*simp*]:
  $S \sim S$
  unfolding *state-eq*$_{NOT}$*-def* **by** *auto*

lemma *state-eq*$_{NOT}$*-sym*:
  $S \sim T \longleftrightarrow T \sim S$
  unfolding *state-eq*$_{NOT}$*-def* **by** *auto*

lemma *state-eq*$_{NOT}$*-trans*:
  $S \sim T \Longrightarrow T \sim U \Longrightarrow S \sim U$
  unfolding *state-eq*$_{NOT}$*-def* **by** *auto*

lemma
  shows
    state-eq$_{NOT}$*-trail*: $S \sim T \Longrightarrow$ *trail* S = *trail* T **and**
    state-eq$_{NOT}$*-clauses*: $S \sim T \Longrightarrow$ *clauses* S = *clauses* T
  unfolding *state-eq*$_{NOT}$*-def* **by** *auto*

lemmas *state-simp*$_{NOT}$[*simp*]= *state-eq*$_{NOT}$*-trail* *state-eq*$_{NOT}$*-clauses*

lemma *trail-eq-reduce-trail-to*$_{NOT}$*-eq*:
  trail S = *trail* T $\Longrightarrow$ *trail* (*reduce-trail-to*$_{NOT}$ F S) = *trail* (*reduce-trail-to*$_{NOT}$ F T)
  apply (*induction* F S *arbitrary*: T *rule*: *reduce-trail-to*$_{NOT}$*.induct*)
  by (*metis tl-trail reduce-trail-to*$_{NOT}$*-eq-length reduce-trail-to*$_{NOT}$*-length-ne reduce-trail-to*$_{NOT}$*-nil*)

lemma *reduce-trail-to*$_{NOT}$*-state-eq*$_{NOT}$*-compatible*:
  assumes ST: $S \sim T$
  shows *reduce-trail-to*$_{NOT}$ F S $\sim$ *reduce-trail-to*$_{NOT}$ F T
proof −
  have *clauses* (*reduce-trail-to*$_{NOT}$ F S) = *clauses* (*reduce-trail-to*$_{NOT}$ F T)
    using ST **by** *auto*
  moreover have *trail* (*reduce-trail-to*$_{NOT}$ F S) = *trail* (*reduce-trail-to*$_{NOT}$ F T)
    using *trail-eq-reduce-trail-to*$_{NOT}$*-eq*[*of* S T F] ST **by** *auto*
  ultimately show *?thesis* **by** (*auto simp del*: *state-simp*$_{NOT}$ *simp*: *state-eq*$_{NOT}$*-def*)
qed

lemma *trail-reduce-trail-to*$_{NOT}$*-add-cls*$_{NOT}$[*simp*]:
  no-dup (*trail* S) $\Longrightarrow$
    trail (*reduce-trail-to*$_{NOT}$ F (*add-cls*$_{NOT}$ C S)) = *trail* (*reduce-trail-to*$_{NOT}$ F S)
  by (*rule trail-eq-reduce-trail-to*$_{NOT}$*-eq*) *simp*

lemma *reduce-trail-to*$_{NOT}$*-trail-tl-trail-decomp*[*simp*]:
  trail S = F' @ *Marked* K () # F $\Longrightarrow$
     trail (*reduce-trail-to*$_{NOT}$ F (*tl-trail* S)) = F
  apply (*rule reduce-trail-to*$_{NOT}$*-skip-beginning*[*of - tl* (F' @ *Marked* K () # [])])
  by (*cases* F') (*auto simp add*:*tl-append reduce-trail-to*$_{NOT}$*-skip-beginning*)

end

2.2.2 Definition of the operation

locale *propagate-ops* =
  dpll-state trail clauses prepend-trail tl-trail add-cls$_{NOT}$ *remove-cls*$_{NOT}$ **for**
    trail :: $'st \Rightarrow ('v, unit, unit)$ *ann-literals* **and**
    clauses :: $'st \Rightarrow 'v$ *clauses* **and**
    prepend-trail :: $('v, unit, unit)$ *ann-literal* $\Rightarrow 'st \Rightarrow 'st$ **and**

$tl\text{-}trail :: 'st \Rightarrow 'st$ **and**
$add\text{-}cls_{NOT}\ remove\text{-}cls_{NOT} :: 'v\ clause \Rightarrow 'st \Rightarrow 'st$ **and**
$propagate\text{-}cond :: ('v,\ unit,\ unit)\ ann\text{-}literal \Rightarrow 'st \Rightarrow bool$
begin
inductive $propagate_{NOT} :: 'st \Rightarrow 'st \Rightarrow bool$ **where**
$propagate_{NOT}[intro]\colon C + \{\#L\#\} \in\#\ clauses\ S \Longrightarrow trail\ S \models as\ CNot\ C$
$\Longrightarrow undefined\text{-}lit\ (trail\ S)\ L$
$\Longrightarrow propagate\text{-}cond\ (Propagated\ L\ ())\ S$
$\Longrightarrow T \sim prepend\text{-}trail\ (Propagated\ L\ ())\ S$
$\Longrightarrow propagate_{NOT}\ S\ T$
inductive-cases $propagate_{NOT}E[elim]\colon propagate_{NOT}\ S\ T$

end

locale $decide\text{-}ops =$
$dpll\text{-}state\ trail\ clauses\ prepend\text{-}trail\ tl\text{-}trail\ add\text{-}cls_{NOT}\ remove\text{-}cls_{NOT}$ **for**
$trail :: 'st \Rightarrow ('v,\ unit,\ unit)\ ann\text{-}literals$ **and**
$clauses :: 'st \Rightarrow 'v\ clauses$ **and**
$prepend\text{-}trail :: ('v,\ unit,\ unit)\ ann\text{-}literal \Rightarrow 'st \Rightarrow 'st$ **and**
$tl\text{-}trail :: 'st \Rightarrow 'st$ **and**
$add\text{-}cls_{NOT}\ remove\text{-}cls_{NOT} :: 'v\ clause \Rightarrow 'st \Rightarrow 'st$
begin
inductive $decide_{NOT} :: 'st \Rightarrow 'st \Rightarrow bool$ **where**
$decide_{NOT}[intro]\colon undefined\text{-}lit\ (trail\ S)\ L \Longrightarrow atm\text{-}of\ L \in atms\text{-}of\text{-}msu\ (clauses\ S)$
$\Longrightarrow T \sim prepend\text{-}trail\ (Marked\ L\ ())\ S$
$\Longrightarrow decide_{NOT}\ S\ T$

inductive-cases $decide_{NOT}E[elim]\colon decide_{NOT}\ S\ S'$
end

locale $backjumping\text{-}ops =$
$dpll\text{-}state\ trail\ clauses\ prepend\text{-}trail\ tl\text{-}trail\ add\text{-}cls_{NOT}\ remove\text{-}cls_{NOT}$
for
$trail :: 'st \Rightarrow ('v,\ unit,\ unit)\ ann\text{-}literals$ **and**
$clauses :: 'st \Rightarrow 'v\ clauses$ **and**
$prepend\text{-}trail :: ('v,\ unit,\ unit)\ ann\text{-}literal \Rightarrow 'st \Rightarrow 'st$ **and**
$tl\text{-}trail :: 'st \Rightarrow 'st$ **and**
$add\text{-}cls_{NOT}\ remove\text{-}cls_{NOT} :: 'v\ clause \Rightarrow 'st \Rightarrow 'st$ +
fixes
$backjump\text{-}conds :: 'v\ clause \Rightarrow 'v\ clause \Rightarrow 'v\ literal \Rightarrow 'st \Rightarrow 'st \Rightarrow bool$
begin
inductive $backjump$ **where**
$trail\ S = F'\ @\ Marked\ K\ ()\#\ F$
$\Longrightarrow T \sim prepend\text{-}trail\ (Propagated\ L\ ())\ (reduce\text{-}trail\text{-}to_{NOT}\ F\ S)$
$\Longrightarrow C \in\#\ clauses\ S$
$\Longrightarrow trail\ S \models as\ CNot\ C$
$\Longrightarrow undefined\text{-}lit\ F\ L$
$\Longrightarrow atm\text{-}of\ L \in atms\text{-}of\text{-}msu\ (clauses\ S) \cup atm\text{-}of\ `\ (lits\text{-}of\ (trail\ S))$
$\Longrightarrow clauses\ S \models pm\ C' + \{\#L\#\}$
$\Longrightarrow F \models as\ CNot\ C'$
$\Longrightarrow backjump\text{-}conds\ C\ C'\ L\ S\ T$
$\Longrightarrow backjump\ S\ T$
inductive-cases $backjumpE\colon backjump\ S\ T$
end

2.3 DPLL with backjumping

locale *dpll-with-backjumping-ops* =
  dpll-state trail clauses prepend-trail tl-trail $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ +
  propagate-ops trail clauses prepend-trail tl-trail $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ *propagate-conds* +
  decide-ops trail clauses prepend-trail tl-trail $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ +
  backjumping-ops trail clauses prepend-trail tl-trail $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ *backjump-conds*
  for
    trail :: $'st \Rightarrow ('v, unit, unit)$ *ann-literals* **and**
    clauses :: $'st \Rightarrow 'v$ *clauses* **and**
    prepend-trail :: $('v, unit, unit)$ *ann-literal* $\Rightarrow 'st \Rightarrow 'st$ **and**
    tl-trail :: $'st \Rightarrow 'st$ **and**
    $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$:: $'v$ *clause* $\Rightarrow 'st \Rightarrow 'st$ **and**
    propagate-conds :: $('v, unit, unit)$ *ann-literal* $\Rightarrow 'st \Rightarrow bool$ **and**
    inv :: $'st \Rightarrow bool$ **and**
    backjump-conds :: $'v$ *clause* $\Rightarrow 'v$ *clause* $\Rightarrow 'v$ *literal* $\Rightarrow 'st \Rightarrow 'st \Rightarrow bool$ +
  assumes
      bj-can-jump:
      $\bigwedge S\ C\ F'\ K\ F\ L.$
        $inv\ S \Longrightarrow$
        $no\text{-}dup\ (trail\ S) \Longrightarrow$
        $trail\ S = F'\ @\ Marked\ K\ ()\ \#\ F \Longrightarrow$
        $C \in\#\ clauses\ S \Longrightarrow$
        $trail\ S \models as\ CNot\ C \Longrightarrow$
        $undefined\text{-}lit\ F\ L \Longrightarrow$
        $atm\text{-}of\ L \in atms\text{-}of\text{-}msu\ (clauses\ S) \cup atm\text{-}of\ `\ (lits\text{-}of\ (F'\ @\ Marked\ K\ ()\ \#\ F)) \Longrightarrow$
        $clauses\ S \models pm\ C' + \{\#L\#\} \Longrightarrow$
        $F \models as\ CNot\ C' \Longrightarrow$
        $\neg no\text{-}step\ backjump\ S$
begin

We cannot add a like condition $atms\text{-}of\ C' \subseteq atms\text{-}of\text{-}ms\ N$ because to ensure that we can backjump even if the last decision variable has disappeared.

The part of the condition $atm\text{-}of\ L \in atm\text{-}of\ `\ lits\text{-}of\ (F'\ @\ Marked\ K\ ()\ \#\ F)$ is important, otherwise you are not sure that you can backtrack.

2.3.1 Definition

We define dpll with backjumping:

inductive *dpll-bj* :: $'st \Rightarrow 'st \Rightarrow bool$ **for** $S :: 'st$ **where**
$bj\text{-}decide_{NOT}$: $decide_{NOT}\ S\ S' \Longrightarrow dpll\text{-}bj\ S\ S'$ |
$bj\text{-}propagate_{NOT}$: $propagate_{NOT}\ S\ S' \Longrightarrow dpll\text{-}bj\ S\ S'$ |
bj-backjump: $backjump\ S\ S' \Longrightarrow dpll\text{-}bj\ S\ S'$

lemmas *dpll-bj-induct* = *dpll-bj.induct*[*split-format*(*complete*)]
thm *dpll-bj-induct*[*OF dpll-with-backjumping-ops-axioms*]
lemma *dpll-bj-all-induct*[*consumes 2*, *case-names* $decide_{NOT}$ $propagate_{NOT}$ *backjump*]:
  fixes $S\ T :: 'st$
  assumes
    $dpll\text{-}bj\ S\ T$ **and**
    $inv\ S$
    $\bigwedge L\ T.\ undefined\text{-}lit\ (trail\ S)\ L \Longrightarrow atm\text{-}of\ L \in atms\text{-}of\text{-}msu\ (clauses\ S)$
      $\Longrightarrow T \sim prepend\text{-}trail\ (Marked\ L\ ())\ S$
      $\Longrightarrow P\ S\ T$ **and**
    $\bigwedge C\ L\ T.\ C + \{\#L\#\} \in\#\ clauses\ S \Longrightarrow trail\ S \models as\ CNot\ C \Longrightarrow undefined\text{-}lit\ (trail\ S)\ L$

$\Longrightarrow T \sim$ *prepend-trail* (*Propagated* L ()) S
$\Longrightarrow P\ S\ T$ **and**
$\bigwedge C\ F'\ K\ F\ L\ C'\ T.\ C \in\#$ *clauses* $S \Longrightarrow F'$ @ *Marked* K () # $F \models$*as CNot* C
$\Longrightarrow$ *trail* $S = F'$ @ *Marked* K () # F
$\Longrightarrow$ *undefined-lit* $F\ L$
$\Longrightarrow$ *atm-of* $L \in$ *atms-of-msu* (*clauses* S) $\cup$ *atm-of* ' (*lits-of* (F' @ *Marked* K () # F))
$\Longrightarrow$ *clauses* $S \models$*pm* $C' + \{\#L\#\}$
$\Longrightarrow F \models$*as CNot* C'
$\Longrightarrow T \sim$ *prepend-trail* (*Propagated* L ()) (*reduce-trail-to*$_{NOT}$ $F\ S$)
$\Longrightarrow P\ S\ T$
shows $P\ S\ T$
apply (*induct T rule*: *dpll-bj-induct*[*OF local.dpll-with-backjumping-ops-axioms*])
apply (*rule assms*(*1*))
using *assms*(*3*) **apply** *blast*
apply (*elim propagate*$_{NOT}$*E*) **using** *assms*(*4*) **apply** *blast*
apply (*elim backjumpE*) **using** *assms*(*5*) ‹*inv S*› **by** *simp*

2.3.2 Basic properties

First, some better suited induction principle **lemma** *dpll-bj-clauses*:
assumes *dpll-bj S T* **and** *inv S*
shows *clauses S* = *clauses T*
using *assms* **by** (*induction rule*: *dpll-bj-all-induct*) *auto*

No duplicates in the trail **lemma** *dpll-bj-no-dup*:
assumes *dpll-bj S T* **and** *inv S*
and *no-dup* (*trail S*)
shows *no-dup* (*trail T*)
using *assms* **by** (*induction rule*: *dpll-bj-all-induct*)
(*auto simp add*: *defined-lit-map reduce-trail-to*$_{NOT}$*-skip-beginning*)

Valuations **lemma** *dpll-bj-sat-iff*:
assumes *dpll-bj S T* **and** *inv S*
shows $I \models$*sm clauses* $S \longleftrightarrow I \models$*sm clauses* T
using *assms* **by** (*induction rule*: *dpll-bj-all-induct*) *auto*

Clauses **lemma** *dpll-bj-atms-of-ms-clauses-inv*:
assumes
dpll-bj S T **and**
inv S
shows *atms-of-msu* (*clauses S*) = *atms-of-msu* (*clauses T*)
using *assms* **by** (*induction rule*: *dpll-bj-all-induct*) *auto*

lemma *dpll-bj-atms-in-trail*:
assumes
dpll-bj S T **and**
inv S **and**
atm-of ' (*lits-of* (*trail S*)) $\subseteq$ *atms-of-msu* (*clauses S*)
shows *atm-of* ' (*lits-of* (*trail T*)) $\subseteq$ *atms-of-msu* (*clauses S*)
using *assms* **by** (*induction rule*: *dpll-bj-all-induct*)
(*auto simp*: *in-plus-implies-atm-of-on-atms-of-ms reduce-trail-to*$_{NOT}$*-skip-beginning*)

lemma *dpll-bj-atms-in-trail-in-set*:
assumes *dpll-bj S T***and**
inv S **and**

atms-of-msu (*clauses S*) ⊆ *A* **and**
atm-of ‘ (*lits-of* (*trail S*)) ⊆ *A*
shows *atm-of* ‘ (*lits-of* (*trail T*)) ⊆ *A*
using *assms* **by** (*induction rule*: *dpll-bj-all-induct*)
(*auto simp*: *in-plus-implies-atm-of-on-atms-of-ms*)

lemma *dpll-bj-all-decomposition-implies-inv*:
assumes
dpll-bj S T **and**
inv: *inv S* **and**
decomp: *all-decomposition-implies-m* (*clauses S*) (*get-all-marked-decomposition* (*trail S*))
shows *all-decomposition-implies-m* (*clauses T*) (*get-all-marked-decomposition* (*trail T*))
using *assms*(*1*,*2*)
proof (*induction rule*:*dpll-bj-all-induct*)
case $decide_{NOT}$
then show *?case* **using** *decomp* **by** *auto*
next
case ($propagate_{NOT}$ *C L T*) **note** *propa* = *this*(*1*) **and** *undef* = *this*(*3*) **and** *T* = *this*(*4*)
let *?M′* = *trail* (*prepend-trail* (*Propagated L* ()) *S*)
let *?N* = *clauses S*
obtain *a y l* **where** *ay*: *get-all-marked-decomposition ?M′* = (*a*, *y*) # *l*
by (*cases get-all-marked-decomposition ?M′*) *fastforce*+
then have *M′*: *?M′* = *y* @ *a* **using** *get-all-marked-decomposition-decomp*[*of ?M′*] **by** *auto*
have *M*: *get-all-marked-decomposition* (*trail S*) = (*a*, *tl y*) # *l*
using *ay undef* **by** (*cases get-all-marked-decomposition* (*trail S*)) *auto*
have y_0: *y* = (*Propagated L* ()) # (*tl y*)
using *ay undef* **by** (*auto simp add*: *M*)
from *arg-cong*[*OF this*, *of set*] **have** *y*[*simp*]: *set y* = *insert* (*Propagated L* ()) (*set* (*tl y*))
by *simp*
have *tr-S*: *trail S* = *tl y* @ *a*
using *arg-cong*[*OF M′*, *of tl*] y_0 *M get-all-marked-decomposition-decomp* **by** *force*
have *a-Un-N-M*: *unmark a* ∪ *set-mset ?N* ⊨*ps unmark* (*tl y*)
using *decomp ay* **unfolding** *all-decomposition-implies-def* **by** (*simp add*: *M*)+

moreover have *unmark a* ∪ *set-mset ?N* ⊨*p* {#*L*#} (**is** *?I* ⊨*p* -)
proof (*rule true-clss-cls-plus-CNot*)
show *?I* ⊨*p C* + {#*L*#}
using *propa* $propagate_{NOT}$*.prems* **by** (*auto dest*!: *true-clss-clss-in-imp-true-clss-cls*)
next
have (*λm.* {#*lit-of m*#}) ‘ *set ?M′* ⊨*ps CNot C*
using ‹*trail S* ⊨*as CNot C*› *undef* **by** (*auto simp add*: *true-annots-true-clss-clss*)
have *a1*: (*λm.* {#*lit-of m*#}) ‘ *set a* ∪ (*λm.* {#*lit-of m*#}) ‘ *set* (*tl y*) ⊨*ps CNot C*
using $propagate_{NOT}$*.hyps*(*2*) *tr-S true-annots-true-clss-clss*
by (*force simp add*: *image-Un sup-commute*)
have *a2*: *set-mset* (*clauses S*)∪ *unmark a*
⊨*ps unmark* (*tl y*)
using *calculation* **by** (*auto simp add*: *sup-commute*)
show (*λm.* {#*lit-of m*#}) ‘ *set a* ∪ *set-mset* (*clauses S*)⊨*ps CNot C*
proof −
have *set-mset* (*clauses S*) ∪ (*λm.* {#*lit-of m*#}) ‘ *set a* ⊨*ps*
(*λm.* {#*lit-of m*#}) ‘ *set a* ∪ (*λm.* {#*lit-of m*#}) ‘ *set* (*tl y*)
using *a2 true-clss-clss-def* **by** *blast*
then show (*λm.* {#*lit-of m*#}) ‘ *set a* ∪ *set-mset* (*clauses S*)⊨*ps CNot C*
using *a1* **unfolding** *sup-commute* **by** (*meson true-clss-clss-left-right*
true-clss-clss-union-and true-clss-clss-union-l-r)

qed
qed

ultimately have *unmark a* ∪ *set-mset ?N* $\models$*ps unmark ?M′*
unfolding *M′* **by** (*auto simp add*: *all-in-true-clss-clss image-Un*)

then show *?case*
using *decomp T M undef* **unfolding** *ay all-decomposition-implies-def* **by** (*auto simp add*: *ay*)
next
case (*backjump C F′ K F L D T*) **note** *confl* = *this*(*2*) **and** *tr* = *this*(*3*) **and** *undef* = *this*(*4*)
and *L* = *this*(*5*) **and** *N-C* = *this*(*6*) **and** *vars-D* = *this*(*5*) **and** *T* = *this*(*8*)
have *decomp*: *all-decomposition-implies-m* (*clauses S*) (*get-all-marked-decomposition F*)
using *decomp* **unfolding** *tr all-decomposition-implies-def*
by (*metis* (*no-types*, *lifting*) *get-all-marked-decomposition.simps*(*1*)
get-all-marked-decomposition-never-empty hd-Cons-tl insert-iff list.sel(*3*) *list.set*(*2*)
tl-get-all-marked-decomposition-skip-some)

moreover have *unmark* (*fst* (*hd* (*get-all-marked-decomposition F*)))
∪ *set-mset* (*clauses S*)
$\models$*ps unmark* (*snd* (*hd* (*get-all-marked-decomposition F*)))
by (*metis all-decomposition-implies-cons-single decomp get-all-marked-decomposition-never-empty*
hd-Cons-tl)
moreover
have *vars-of-D*: *atms-of D* ⊆ *atm-of* ‘ *lits-of F*
using ‹*F* $\models$*as CNot D*› **unfolding** *atms-of-def*
by (*meson image-subsetI mem-set-mset-iff true-annots-CNot-all-atms-defined*)

obtain *a b li* **where** *F*: *get-all-marked-decomposition F* = (*a*, *b*) # *li*
by (*cases get-all-marked-decomposition F*) *auto*
have *F* = *b* @ *a*
using *get-all-marked-decomposition-decomp*[*of F a b*] *F* **by** *auto*
have *a-N-b*:*unmark a* ∪ *set-mset* (*clauses S*) $\models$*ps unmark b*
using *decomp* **unfolding** *all-decomposition-implies-def* **by** (*auto simp add*: *F*)

have *F-D*:*unmark F* $\models$*ps CNot D*
using ‹*F* $\models$*as CNot D*› **by** (*simp add*: *true-annots-true-clss-clss*)
then have *unmark a* ∪ *unmark b* $\models$*ps CNot D*
unfolding ‹*F* = *b* @ *a*› **by** (*simp add*: *image-Un sup.commute*)
have *a-N-CNot-D*: *unmark a* ∪ *set-mset* (*clauses S*)
$\models$*ps CNot D* ∪ *unmark b*
apply (*rule true-clss-clss-left-right*)
using *a-N-b F-D* **unfolding** ‹*F* = *b* @ *a*› **by** (*auto simp add*: *image-Un ac-simps*)

have *a-N-D-L*: *unmark a* ∪ *set-mset* (*clauses S*) $\models$*p D*+{#*L*#}
by (*simp add*: *N-C*)
have *unmark a* ∪ *set-mset* (*clauses S*) $\models$*p* {#*L*#}
using *a-N-D-L a-N-CNot-D* **by** (*blast intro*: *true-clss-cls-plus-CNot*)
then show *?case*
using *decomp T tr undef* **unfolding** *all-decomposition-implies-def* **by** (*auto simp add*: *F*)
qed

2.3.3 Termination

Using a proper measure **lemma** *length-get-all-marked-decomposition-append-Marked*:
length (*get-all-marked-decomposition* (*F′* @ *Marked K* () # *F*)) =
length (*get-all-marked-decomposition F′*)

$+$ *length* (*get-all-marked-decomposition* (*Marked K* () # *F*))
$-$ *1*
by (*induction F′ rule*: *ann-literal-list-induct*) *auto*

lemma *take-length-get-all-marked-decomposition-marked-sandwich*:
take (*length* (*get-all-marked-decomposition F*))
(*map* (*f o snd*) (*rev* (*get-all-marked-decomposition* (*F′* @ *Marked K* () # *F*))))
=
map (*f o snd*) (*rev* (*get-all-marked-decomposition F*))

proof (*induction F′ rule*: *ann-literal-list-induct*)
case *nil*
then show *?case* **by** *auto*
next
case (*marked K*)
then show *?case* **by** (*simp add*: *length-get-all-marked-decomposition-append-Marked*)
next
case (*proped L m F′*) **note** *IH* = *this*(*1*)
obtain *a b l* **where** *F′*: *get-all-marked-decomposition* (*F′* @ *Marked K* () # *F*) = (*a*, *b*) # *l*
by (*cases get-all-marked-decomposition* (*F′* @ *Marked K* () # *F*)) *auto*
have *length* (*get-all-marked-decomposition F*) − *length l* = *0*
using *length-get-all-marked-decomposition-append-Marked*[*of F′ K F*]
unfolding *F′* **by** (*cases get-all-marked-decomposition F′*) *auto*
then show *?case*
using *IH* **by** (*simp add*: *F′*)
qed

lemma *length-get-all-marked-decomposition-length*:
length (*get-all-marked-decomposition M*) ≤ *1* + *length M*
by (*induction M rule*: *ann-literal-list-induct*) *auto*

lemma *length-in-get-all-marked-decomposition-bounded*:
assumes *i*:*i* ∈ *set* (*trail-weight S*)
shows *i* ≤ *Suc* (*length* (*trail S*))
proof −
obtain *a b* **where**
(*a*, *b*) ∈ *set* (*get-all-marked-decomposition* (*trail S*)) **and**
ib: *i* = *Suc* (*length b*)
using *i* **by** *auto*
then obtain *c* **where** *trail S* = *c* @ *b* @ *a*
using *get-all-marked-decomposition-exists-prepend′* **by** *metis*
from *arg-cong*[*OF this*, *of length*] **show** *?thesis* **using** *i ib* **by** *auto*
qed

Well-foundedness The bounds are the following:

- *1* + *card* (*atms-of-ms A*): *card* (*atms-of-ms A*) is an upper bound on the length of the list. As *get-all-marked-decomposition* appends an possibly empty couple at the end, adding one is needed.
- *2* + *card* (*atms-of-ms A*): *card* (*atms-of-ms A*) is an upper bound on the number of elements, where adding one is necessary for the same reason as for the bound on the list, and one is needed to have a strict bound.

abbreviation *unassigned-lit* :: *′b literal multiset set* ⇒ *′a list* ⇒ *nat* **where**

unassigned-lit N M ≡ *card* (*atms-of-ms N*) − *length M*

lemma *dpll-bj-trail-mes-increasing-prop*:
fixes *M* :: (*'v*, *unit*, *unit*) *ann-literals* **and** *N* :: *'v clauses*
assumes
dpll-bj S T **and**
inv S **and**
NA: *atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A* **and**
MA: *atm-of* ' *lits-of* (*trail S*) ⊆ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
finite: *finite A*
shows μ_C (*1+card* (*atms-of-ms A*)) (*2+card* (*atms-of-ms A*)) (*trail-weight T*)
> μ_C (*1+card* (*atms-of-ms A*)) (*2+card* (*atms-of-ms A*)) (*trail-weight S*)
using *assms*(*1*,*2*)
proof (*induction rule*: *dpll-bj-all-induct*)
case (*propagate*$_{NOT}$ *C L*) **note** *CLN* = *this*(*1*) **and** *MC* =*this*(*2*) **and** *undef-L* = *this*(*3*) **and** *T* = *this*(*4*)
have *incl*: *atm-of* ' *lits-of* (*Propagated L* () # *trail S*) ⊆ *atms-of-ms A*
using *propagate*$_{NOT}$*.hyps propagate-ops.propagate*$_{NOT}$ *dpll-bj-atms-in-trail-in-set bj-propagate*$_{NOT}$ *NA MA CLN* **by** (*auto simp*: *in-plus-implies-atm-of-on-atms-of-ms*)

have *no-dup*: *no-dup* (*Propagated L* () # *trail S*)
using *defined-lit-map n-d undef-L* **by** *auto*
obtain *a b l* **where** *M*: *get-all-marked-decomposition* (*trail S*) = (*a*, *b*) # *l*
by (*cases get-all-marked-decomposition* (*trail S*)) *auto*
have *b-le-M*: *length b* ≤ *length* (*trail S*)
using *get-all-marked-decomposition-decomp*[*of trail S*] **by** (*simp add*: *M*)
have *finite* (*atms-of-ms A*) **using** *finite* **by** *simp*

then have *length* (*Propagated L* () # *trail S*) ≤ *card* (*atms-of-ms A*)
using *incl finite* **unfolding** *no-dup-length-eq-card-atm-of-lits-of*[*OF no-dup*]
by (*simp add*: *card-mono*)
then have *latm*: *unassigned-lit A b* = *Suc* (*unassigned-lit A* (*Propagated L d* # *b*))
using *b-le-M* **by** *auto*
then show *?case* **using** *T undef-L* **by** (*auto simp*: *latm M* μ_C*-cons*)
next
case (*decide*$_{NOT}$ *L*) **note** *undef-L* = *this*(*1*) **and** *MC* = *this*(*2*) **and** *T* = *this*(*3*)
have *incl*: *atm-of* ' *lits-of* (*Marked L* () # (*trail S*)) ⊆ *atms-of-ms A*
using *dpll-bj-atms-in-trail-in-set bj-decide*$_{NOT}$ *decide*$_{NOT}$*.decide*$_{NOT}$[*OF decide*$_{NOT}$*.hyps*] *NA MA MC*
by *auto*

have *no-dup*: *no-dup* (*Marked L* () # (*trail S*))
using *defined-lit-map n-d undef-L* **by** *auto*
obtain *a b l* **where** *M*: *get-all-marked-decomposition* (*trail S*) = (*a*, *b*) # *l*
by (*cases get-all-marked-decomposition* (*trail S*)) *auto*

then have *length* (*Marked L* () # (*trail S*)) ≤ *card* (*atms-of-ms A*)
using *incl finite* **unfolding** *no-dup-length-eq-card-atm-of-lits-of*[*OF no-dup*]
by (*simp add*: *card-mono*)
then have *latm*: *unassigned-lit A* (*trail S*) = *Suc* (*unassigned-lit A* (*Marked L lv* # (*trail S*)))
by *force*
show *?case* **using** *T undef-L* **by** (*simp add*: *latm* μ_C*-cons*)
next
case (*backjump C F' K F L C' T*) **note** *undef-L* = *this*(*4*) **and** *MC* =*this*(*1*) **and** *tr-S* = *this*(*3*) **and**

$L = this(5)$ **and** $T = this(8)$
have *incl*: *atm-of* ` *lits-of* (*Propagated L* () # *F*) ⊆ *atms-of-ms A*
using *dpll-bj-atms-in-trail-in-set NA MA tr-S L* **by** *auto*

have *no-dup*: *no-dup* (*Propagated L* () # *F*)
using *defined-lit-map n-d undef-L tr-S* **by** *auto*
obtain *a b l* **where** *M*: *get-all-marked-decomposition* (*trail S*) = (*a*, *b*) # *l*
by (*cases get-all-marked-decomposition* (*trail S*)) *auto*
have *b-le-M*: *length b* ≤ *length* (*trail S*)
using *get-all-marked-decomposition-decomp*[*of trail S*] **by** (*simp add*: *M*)
have *fin-atms-A*: *finite* (*atms-of-ms A*) **using** *finite* **by** *simp*

then have *F-le-A*: *length* (*Propagated L* () # *F*) ≤ *card* (*atms-of-ms A*)
using *incl finite* **unfolding** *no-dup-length-eq-card-atm-of-lits-of*[*OF no-dup*]
by (*simp add*: *card-mono*)
have *tr-S-le-A*: *length* (*trail S*) ≤ (*card* (*atms-of-ms A*))
using *n-d MA* **by** (*metis fin-atms-A card-mono no-dup-length-eq-card-atm-of-lits-of*)
obtain *a b l* **where** *F*: *get-all-marked-decomposition F* = (*a*, *b*) # *l*
by (*cases get-all-marked-decomposition F*) *auto*
then have *F* = *b* @ *a*
using *get-all-marked-decomposition-decomp*[*of Propagated L* () # *F a*
Propagated L () # *b*] **by** *simp*
then have *latm*: *unassigned-lit A b* = *Suc* (*unassigned-lit A* (*Propagated L* () # *b*))
using *F-le-A* **by** *simp*
obtain *rem* **where**
rem:*map* (λa. *Suc* (*length* (*snd a*))) (*rev* (*get-all-marked-decomposition* (F' @ *Marked K* () # *F*)))
= *map* (λa. *Suc* (*length* (*snd a*))) (*rev* (*get-all-marked-decomposition F*)) @ *rem*
using *take-length-get-all-marked-decomposition-marked-sandwich*[*of F* λa. *Suc* (*length a*) F' *K*]
unfolding *o-def* **by** (*metis append-take-drop-id*)
then have *rem*: *map* (λa. *Suc* (*length* (*snd a*)))
(*get-all-marked-decomposition* (F' @ *Marked K* () # *F*))
= *rev rem* @ *map* (λa. *Suc* (*length* (*snd a*))) ((*get-all-marked-decomposition F*))
by (*simp add*: *rev-map*[*symmetric*] *rev-swap*)
have *length* (*rev rem* @ *map* (λa. *Suc* (*length* (*snd a*))) (*get-all-marked-decomposition F*))
≤ *Suc* (*card* (*atms-of-ms A*))
using *arg-cong*[*OF rem*, *of length*] *tr-S-le-A*
length-get-all-marked-decomposition-length[*of* F' @ *Marked K* () # *F*] *tr-S* **by** *auto*
moreover
{ **fix** *i* :: *nat* **and** *xs* :: *'a list*
have *i* < *length xs* ⟹ *length xs* − *Suc i* < *length xs*
by *auto*
then have *H*: *i*<*length xs* ⟹ *rev xs* ! *i* ∈ *set xs*
using *rev-nth*[*of i xs*] **unfolding** *in-set-conv-nth* **by** (*force simp add*: *in-set-conv-nth*)
} **note** *H* = *this*
have ∀ *i*<*length rem*. *rev rem* ! *i* < *card* (*atms-of-ms A*) + *2*
using *tr-S-le-A length-in-get-all-marked-decomposition-bounded*[*of - S*] **unfolding** *tr-S*
by (*force simp add*: *o-def rem dest*!: *H intro*: *length-get-all-marked-decomposition-length*)
ultimately show *?case*
using μ_C*-bounded*[*of rev rem card* (*atms-of-ms A*)+*2 unassigned-lit A l*] *T undef-L*
by (*simp add*: *rem* μ_C*-append* μ_C*-cons F tr-S*)
qed

lemma *dpll-bj-trail-mes-decreasing-prop*:
assumes *dpll*: *dpll-bj S T* **and** *inv*: *inv S* **and**
N-A: *atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A* **and**

```
  M-A: atm-of ` lits-of (trail S) ⊆ atms-of-ms A and
  nd: no-dup (trail S) and
  fin-A: finite A
  shows (2+card (atms-of-ms A)) ^ (1+card (atms-of-ms A))
              − μC (1+card (atms-of-ms A)) (2+card (atms-of-ms A)) (trail-weight T)
            < (2+card (atms-of-ms A)) ^ (1+card (atms-of-ms A))
              − μC (1+card (atms-of-ms A)) (2+card (atms-of-ms A)) (trail-weight S)
proof −
  let ?b = 2+card (atms-of-ms A)
  let ?s = 1+card (atms-of-ms A)
  let ?μ = μC ?s ?b
  have M'-A: atm-of ` lits-of (trail T) ⊆ atms-of-ms A
    by (meson M-A N-A dpll dpll-bj-atms-in-trail-in-set inv)
  have nd': no-dup (trail T)
    using ‹dpll-bj S T› dpll-bj-no-dup nd inv by blast
  { fix i :: nat and xs :: 'a list
    have i < length xs ⟹ length xs − Suc i < length xs
      by auto
    then have H: i<length xs ⟹  xs ! i ∈ set xs
      using rev-nth[of i xs] unfolding in-set-conv-nth by (force simp add: in-set-conv-nth)
  } note H = this

  have l-M-A: length (trail S) ≤ card (atms-of-ms A)
    by (simp add: fin-A M-A card-mono no-dup-length-eq-card-atm-of-lits-of nd)
  have l-M'-A: length (trail T) ≤ card (atms-of-ms A)
    by (simp add: fin-A M'-A card-mono no-dup-length-eq-card-atm-of-lits-of nd')
  have l-trail-weight-M: length (trail-weight T) ≤ 1+card (atms-of-ms A)
     using l-M'-A length-get-all-marked-decomposition-length[of trail T] by auto
  have bounded-M: ∀ i<length (trail-weight T). (trail-weight T)! i < card (atms-of-ms A) + 2
    using length-in-get-all-marked-decomposition-bounded[of - T] l-M'-A
    by (metis (no-types, lifting) Nat.le-trans One-nat-def Suc-1 add.right-neutral add-Suc-right
      le-imp-less-Suc less-eq-Suc-le nth-mem)

  from dpll-bj-trail-mes-increasing-prop[OF dpll inv N-A M-A nd fin-A]
  have μC ?s ?b (trail-weight S) < μC ?s ?b (trail-weight T) by simp
  moreover from μC-bounded[OF bounded-M l-trail-weight-M]
    have μC ?s ?b (trail-weight T) ≤ ?b ^ ?s by auto
  ultimately show ?thesis by linarith
qed

lemma wf-dpll-bj:
  assumes fin: finite A
  shows wf {(T, S). dpll-bj S T
    ∧ atms-of-msu (clauses S) ⊆ atms-of-ms A ∧ atm-of ` lits-of (trail S) ⊆ atms-of-ms A
    ∧ no-dup (trail S) ∧ inv S}
  (is wf ?A)
proof (rule wf-bounded-measure[of -
        λ-. (2 + card (atms-of-ms A))^(1 + card (atms-of-ms A))
        λS. μC (1+card (atms-of-ms A)) (2+card (atms-of-ms A)) (trail-weight S)])
  fix a b :: 'st
  let ?b = 2+card (atms-of-ms A)
  let ?s = 1+card (atms-of-ms A)
  let ?μ = μC ?s ?b
  assume ab: (b, a) ∈ {(T, S). dpll-bj S T
    ∧ atms-of-msu (clauses S) ⊆ atms-of-ms A ∧ atm-of ` lits-of (trail S) ⊆ atms-of-ms A
```

$\wedge$ *no-dup* (*trail* S) $\wedge$ *inv* S}

have *fin-A*: *finite* (*atms-of-ms* A)
  using *fin* **by** *auto*
have
  dpll-bj: *dpll-bj a b* **and**
  N-A: *atms-of-msu* (*clauses a*) $\subseteq$ *atms-of-ms* A **and**
  M-A: *atm-of* ‘ *lits-of* (*trail a*) $\subseteq$ *atms-of-ms* A **and**
  nd: *no-dup* (*trail a*) **and**
  inv: *inv a*
  using *ab* **by** *auto*

have *M′-A*: *atm-of* ‘ *lits-of* (*trail b*) $\subseteq$ *atms-of-ms* A
  by (*meson M-A N-A* ‹*dpll-bj a b*› *dpll-bj-atms-in-trail-in-set inv*)
have *nd′*: *no-dup* (*trail b*)
  using ‹*dpll-bj a b*› *dpll-bj-no-dup nd inv* **by** *blast*
{ fix *i* :: *nat* **and** *xs* :: *′a list*
  have $i <$ *length xs* $\Longrightarrow$ *length xs* $-$ *Suc i* $<$ *length xs*
    by *auto*
  then have *H*: *i*$<$*length xs* $\Longrightarrow$ *xs* ! *i* $\in$ *set xs*
    using *rev-nth*[*of i xs*] **unfolding** *in-set-conv-nth* **by** (*force simp add*: *in-set-conv-nth*)
} **note** H = *this*

have *l-M-A*: *length* (*trail a*) $\leq$ *card* (*atms-of-ms* A)
  by (*simp add*: *fin-A M-A card-mono no-dup-length-eq-card-atm-of-lits-of nd*)
have *l-M′-A*: *length* (*trail b*) $\leq$ *card* (*atms-of-ms* A)
  by (*simp add*: *fin-A M′-A card-mono no-dup-length-eq-card-atm-of-lits-of nd′*)
have *l-trail-weight-M*: *length* (*trail-weight b*) $\leq$ *1*+*card* (*atms-of-ms* A)
   using *l-M′-A length-get-all-marked-decomposition-length*[*of trail b*] **by** *auto*
have *bounded-M*: $\forall$ *i*$<$*length* (*trail-weight b*). (*trail-weight b*)! *i* $<$ *card* (*atms-of-ms* A) + *2*
  using *length-in-get-all-marked-decomposition-bounded*[*of - b*] *l-M′-A*
  by (*metis* (*no-types*, *lifting*) *Nat.le-trans One-nat-def Suc-1 add.right-neutral add-Suc-right*
    le-imp-less-Suc less-eq-Suc-le nth-mem)

from *dpll-bj-trail-mes-increasing-prop*[*OF dpll-bj inv N-A M-A nd fin*]
have μ_C *?s ?b* (*trail-weight a*) $<$ μ_C *?s ?b* (*trail-weight b*) **by** *simp*
moreover from μ_C*-bounded*[*OF bounded-M l-trail-weight-M*]
  have μ_C *?s ?b* (*trail-weight b*) $\leq$ *?b* ^ *?s* **by** *auto*
ultimately show *?b* ^ *?s* $\leq$ *?b* ^ *?s* $\wedge$
      μ_C *?s ?b* (*trail-weight b*) $\leq$ *?b* ^ *?s* $\wedge$
      μ_C *?s ?b* (*trail-weight a*) $<$ μ_C *?s ?b* (*trail-weight b*)
  by *blast*
qed

2.3.4 Normal Forms

We prove that given a normal form of DPLL, with some invariants, the either N is satisfiable and the built valuation M is a model; or N is unsatisfiable.

Idea of the proof: We have to prove tat *satisfiable* N, $\neg$ M $\models$*as* N and there is no remaining step is incompatible.

1. The *decide* rules tells us that every variable in N has a value.

2. $\neg$ M $\models$*as* N tells us that there is conflict.

3. There is at least one decision in the trail (otherwise, M is a model of N).
4. Now if we build the clause with all the decision literals of the trail, we can apply the *backjump* rule.

The assumption are saying that we have a finite upper bound A for the literals, that we cannot do any step *no-step dpll-bj S*

theorem *dpll-backjump-final-state*:
  fixes *A* :: *'v literal multiset set* **and** *S T* :: *'st*
  assumes
    atms-of-msu (*clauses S*) ⊆ *atms-of-ms A* **and**
    atm-of ‘ *lits-of* (*trail S*) ⊆ *atms-of-ms A* **and**
    no-dup (*trail S*) **and**
    finite A **and**
    inv: *inv S* **and**
    n-s: *no-step dpll-bj S* **and**
    decomp: *all-decomposition-implies-m* (*clauses S*) (*get-all-marked-decomposition* (*trail S*))
  shows *unsatisfiable* (*set-mset* (*clauses S*))
    ∨ (*trail S* ⊨*asm clauses S* ∧ *satisfiable* (*set-mset* (*clauses S*)))
proof −
  let *?N* = *set-mset* (*clauses S*)
  let *?M* = *trail S*
  consider
      (*sat*) *satisfiable ?N* **and** *?M* ⊨*as ?N*
    | (*sat'*) *satisfiable ?N* **and** ¬ *?M* ⊨*as ?N*
    | (*unsat*) *unsatisfiable ?N*
    by *auto*
  then show *?thesis*
    proof *cases*
      case *sat'* **note** *sat* = *this*(*1*) **and** *M* = *this*(*2*)
      obtain *C* **where** *C* ∈ *?N* **and** ¬*?M* ⊨*a C* **using** *M* **unfolding** *true-annots-def* **by** *auto*
      obtain *I* :: *'v literal set* **where**
        I ⊨*s ?N* **and**
        cons: *consistent-interp I* **and**
        tot: *total-over-m I ?N* **and**
        atm-I-N: *atm-of* ‘*I* ⊆ *atms-of-ms ?N*
        using *sat* **unfolding** *satisfiable-def-min* **by** *auto*
      let *?I* = *I* ∪ {*P*| *P*. *P* ∈ *lits-of ?M* ∧ *atm-of P* ∉ *atm-of* ‘ *I*}
      let *?O* = {{#*lit-of L*#} |*L*. *is-marked L* ∧ *L* ∈ *set ?M* ∧ *atm-of* (*lit-of L*) ∉ *atms-of-ms ?N*}
      have *cons-I'*: *consistent-interp ?I*
        using *cons* **using** ‹*no-dup ?M*› **unfolding** *consistent-interp-def*
        by (*auto simp add*: *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set lits-of-def*
          dest!: *no-dup-cannot-not-lit-and-uminus*)
      have *tot-I'*: *total-over-m ?I* (*?N* ∪ *unmark ?M*)
        using *tot atms-of-s-def* **unfolding** *total-over-m-def total-over-set-def*
        by *fastforce*
      have {*P* |*P*. *P* ∈ *lits-of ?M* ∧ *atm-of P* ∉ *atm-of* ‘ *I*} ⊨*s ?O*
        using ‹*I*⊨*s ?N*› *atm-I-N* **by** (*auto simp add*: *atm-of-eq-atm-of true-clss-def lits-of-def*)
      then have *I'-N*: *?I* ⊨*s ?N* ∪ *?O*
        using ‹*I*⊨*s ?N*› *true-clss-union-increase* **by** *force*
      have *tot'*: *total-over-m ?I* (*?N*∪*?O*)
        using *atm-I-N tot* **unfolding** *total-over-m-def total-over-set-def*
        by (*force simp*: *image-iff lits-of-def dest*!: *is-marked-ex-Marked*)

      have *atms-N-M*: *atms-of-ms ?N* ⊆ *atm-of* ‘ *lits-of ?M*

proof (*rule ccontr*)
assume ¬ *?thesis*
then obtain *l* :: *'v* **where**
l-N: *l* ∈ *atms-of-ms ?N* **and**
l-M: *l* ∉ *atm-of ‘ lits-of ?M*
by *auto*
have *undefined-lit ?M* (*Pos l*)
using *l-M* **by** (*metis Marked-Propagated-in-iff-in-lits-of*
atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set literal.sel(*1*))
from $\mathit{bj\text{-}decide}_{NOT}$[*OF* decide_{NOT}[*OF this*]] **show** *False*
using *l-N n-s* **by** (*metis literal.sel*(*1*) $\mathit{state\text{-}eq}_{NOT}$*-ref*)
qed

have *?M* ⊨*as CNot C*
by (*metis* ‹*C* ∈ *set-mset* (*clauses S*)› ‹¬ *trail S* ⊨*a C*› *all-variables-defined-not-imply-cnot*
atms-N-M atms-of-atms-of-ms-mono atms-of-ms-CNot-atms-of atms-of-ms-CNot-atms-of-ms
subset-eq)
have ∃ *l* ∈ *set ?M*. *is-marked l*
proof (*rule ccontr*)
let *?O* = {{#*lit-of L*#} |*L*. *is-marked L* ∧ *L* ∈ *set ?M* ∧ *atm-of* (*lit-of L*) ∉ *atms-of-ms ?N*}
have *ϑ*[*iff*]: ⋀*I*. *total-over-m I* (*?N* ∪ *?O* ∪ *unmark ?M*)
⟷ *total-over-m I* (*?N* ∪*unmark ?M*)
unfolding *total-over-set-def total-over-m-def atms-of-ms-def* **by** *auto*
assume ¬ *?thesis*
then have [*simp*]:{{#*lit-of L*#} |*L*. *is-marked L* ∧ *L* ∈ *set ?M*}
= {{#*lit-of L*#} |*L*. *is-marked L* ∧ *L* ∈ *set ?M* ∧ *atm-of* (*lit-of L*) ∉ *atms-of-ms ?N*}
by *auto*
then have *?N* ∪ *?O* ⊨*ps unmark ?M*
using *all-decomposition-implies-propagated-lits-are-implied*[*OF decomp*] **by** *auto*

then have *?I* ⊨*s unmark ?M*
using *cons-I' I'-N tot-I'* ‹*?I* ⊨*s ?N* ∪ *?O*› **unfolding** *ϑ true-clss-clss-def* **by** *blast*
then have *lits-of ?M* ⊆ *?I*
unfolding *true-clss-def lits-of-def* **by** *auto*
then have *?M* ⊨*as ?N*
using *I'-N* ‹*C* ∈ *?N*› ‹¬ *?M* ⊨*a C*› *cons-I' atms-N-M*
by (*meson* ‹*trail S* ⊨*as CNot C*› *consistent-CNot-not rev-subsetD sup-ge1 true-annot-def*
true-annots-def true-cls-mono-set-mset-l true-clss-def)
then show *False* **using** *M* **by** *fast*
qed
from *List.split-list-first-propE*[*OF this*] **obtain** *K* :: *'v literal* **and**
F F' :: (*'v*, *unit*, *unit*) *ann-literal list* **where**
M-K: *?M* = *F'* @ *Marked K* () # *F* **and**
nm: ∀*f*∈*set F'*. ¬*is-marked f*
unfolding *is-marked-def* **by** (*metis* (*full-types*) *old.unit.exhaust*)
let *?K* = *Marked K* ()::(*'v*, *unit*, *unit*) *ann-literal*
have *?K* ∈ *set ?M*
unfolding *M-K* **by** *auto*
let *?C* = *image-mset lit-of* {#*L*∈#*mset ?M*. *is-marked L* ∧ *L*≠*?K*#} :: *'v literal multiset*
let *?C'* = *set-mset* (*image-mset* (λ*L*::*'v literal*. {#*L*#}) (*?C*+{#*lit-of ?K*#}))
have *?N* ∪ {{#*lit-of L*#} |*L*. *is-marked L* ∧ *L* ∈ *set ?M*} ⊨*ps unmark ?M*
using *all-decomposition-implies-propagated-lits-are-implied*[*OF decomp*] **.**
moreover have *C'*: *?C'* = {{#*lit-of L*#} |*L*. *is-marked L* ∧ *L* ∈ *set ?M*}
unfolding *M-K* **apply** *standard*
apply *force*

using *IntI* **by** *auto*
ultimately have *N-C-M*: *?N* ∪ *?C′* ⊨*ps unmark ?M*
by *auto*
have *N-M-False*: *?N* ∪ (λ*L*. {#*lit-of L*#}) ‘ (*set ?M*) ⊨*ps* {{#}}
using *M* ‹*?M* ⊨*as CNot C*› ‹*C*∈*?N*› **unfolding** *true-clss-clss-def true-annots-def Ball-def true-annot-def* **by** (*metis consistent-CNot-not sup.orderE sup-commute true-clss-def true-clss-singleton-lit-of-implies-incl true-clss-union true-clss-union-increase*)

have *undefined-lit F K* **using** ‹*no-dup ?M*› **unfolding** *M-K* **by** (*simp add*: *defined-lit-map*)
moreover
have *?N* ∪ *?C′* ⊨*ps* {{#}}
proof −
have *A*: *?N* ∪ *?C′* ∪ *unmark ?M* =
?N ∪ *unmark ?M*
unfolding *M-K* **by** *auto*
show *?thesis*
using *true-clss-clss-left-right*[*OF N-C-M*, *of* {{#}}] *N-M-False* **unfolding** *A* **by** *auto*
qed
have *?N* ⊨*p image-mset uminus ?C* + {#−*K*#}
unfolding *true-clss-cls-def true-clss-clss-def total-over-m-def*
proof (*intro allI impI*)
fix *I*
assume
tot: *total-over-set I* (*atms-of-ms* (*?N* ∪ {*image-mset uminus ?C*+ {#− *K*#}})) **and**
cons: *consistent-interp I* **and**
I ⊨*s ?N*
have (*K* ∈ *I* ∧ −*K* ∉ *I*) ∨ (−*K* ∈ *I* ∧ *K* ∉ *I*)
using *cons tot* **unfolding** *consistent-interp-def* **by** (*cases K*) *auto*
have *tot′*: *total-over-set I*
(*atm-of* ‘ *lit-of* ‘ (*set ?M* ∩ {*L*. *is-marked L* ∧ *L* ≠ *Marked K* ()}))
using *tot* **by** (*auto simp add*: *atms-of-uminus-lit-atm-of-lit-of*)
{ **fix** *x* :: (′*v*, *unit*, *unit*) *ann-literal*
assume
a3: *lit-of x* ∉ *I* **and**
a1: *x* ∈ *set ?M* **and**
a4: *is-marked x* **and**
a5: *x* ≠ *Marked K* ()
then have *Pos* (*atm-of* (*lit-of x*)) ∈ *I* ∨ *Neg* (*atm-of* (*lit-of x*)) ∈ *I*
using *a5 a4 tot′ a1* **unfolding** *total-over-set-def atms-of-s-def* **by** *blast*
moreover have *f6*: *Neg* (*atm-of* (*lit-of x*)) = − *Pos* (*atm-of* (*lit-of x*))
by *simp*
ultimately have − *lit-of x* ∈ *I*
using *f6 a3* **by** (*metis* (*no-types*) *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set literal.sel*(*1*))
} **note** *H* = *this*

have ¬*I* ⊨*s ?C′*
using ‹*?N* ∪ *?C′* ⊨*ps* {{#}}› *tot cons* ‹*I* ⊨*s ?N*›
unfolding *true-clss-clss-def total-over-m-def*
by (*simp add*: *atms-of-uminus-lit-atm-of-lit-of atms-of-ms-single-image-atm-of-lit-of*)
then show *I* ⊨ *image-mset uminus ?C* + {#− *K*#}
unfolding *true-clss-def true-cls-def Bex-mset-def*
using ‹(*K* ∈ *I* ∧ −*K* ∉ *I*) ∨ (−*K* ∈ *I* ∧ *K* ∉ *I*)›
by (*auto dest*!: *H*)
qed

moreover have *F* $\models$*as CNot* (*image-mset uminus ?C*)
using *nm* **unfolding** *true-annots-def CNot-def M-K* **by** (*auto simp add*: *lits-of-def*)
ultimately have *False*
using *bj-can-jump*[*of S F′ K F C −K*
image-mset uminus (*image-mset lit-of* {# *L* :# *mset ?M*. *is-marked L* ∧ *L* ≠ *Marked K* ()#})]
‹*C*∈*?N*› *n-s* ‹*?M* $\models$*as CNot C*› *bj-backjump inv* ‹*no-dup* (*trail S*)› **unfolding** *M-K* **by** *auto*
then show *?thesis* **by** *fast*
qed *auto*
qed

end

locale *dpll-with-backjumping* =
dpll-with-backjumping-ops trail clauses prepend-trail tl-trail add-cls$_{NOT}$ *remove-cls*$_{NOT}$
propagate-conds inv backjump-conds
for
trail :: ′*st* ⇒ (′*v*, *unit*, *unit*) *ann-literals* **and**
clauses :: ′*st* ⇒ ′*v clauses* **and**
prepend-trail :: (′*v*, *unit*, *unit*) *ann-literal* ⇒ ′*st* ⇒ ′*st* **and** *tl-trail* :: ′*st* ⇒ ′*st* **and**
add-cls$_{NOT}$ *remove-cls*$_{NOT}$:: ′*v clause* ⇒ ′*st* ⇒ ′*st* **and**
propagate-conds :: (′*v*, *unit*, *unit*) *ann-literal* ⇒ ′*st* ⇒ *bool* **and**
inv :: ′*st* ⇒ *bool* **and**
backjump-conds :: ′*v clause* ⇒ ′*v clause* ⇒ ′*v literal* ⇒ ′*st* ⇒ ′*st* ⇒ *bool*
+
assumes *dpll-bj-inv*:⋀*S T*. *dpll-bj S T* ⟹ *inv S* ⟹ *inv T*
begin

lemma *rtranclp-dpll-bj-inv*:
assumes *dpll-bj*** *S T* **and** *inv S*
shows *inv T*
using *assms* **by** (*induction rule*: *rtranclp-induct*)
(*auto simp add*: *dpll-bj-no-dup intro*: *dpll-bj-inv*)

lemma *rtranclp-dpll-bj-no-dup*:
assumes *dpll-bj*** *S T* **and** *inv S*
and *no-dup* (*trail S*)
shows *no-dup* (*trail T*)
using *assms* **by** (*induction rule*: *rtranclp-induct*)
(*auto simp add*: *dpll-bj-no-dup dest*: *rtranclp-dpll-bj-inv dpll-bj-inv*)

lemma *rtranclp-dpll-bj-atms-of-ms-clauses-inv*:
assumes
dpll-bj** *S T* **and** *inv S*
shows *atms-of-msu* (*clauses S*) = *atms-of-msu* (*clauses T*)
using *assms* **by** (*induction rule*: *rtranclp-induct*)
(*auto dest*: *rtranclp-dpll-bj-inv dpll-bj-atms-of-ms-clauses-inv*)

lemma *rtranclp-dpll-bj-atms-in-trail*:
assumes
dpll-bj** *S T* **and**
inv S **and**
atm-of ‘ (*lits-of* (*trail S*)) ⊆ *atms-of-msu* (*clauses S*)
shows *atm-of* ‘ (*lits-of* (*trail T*)) ⊆ *atms-of-msu* (*clauses T*)
using *assms* **apply** (*induction rule*: *rtranclp-induct*)
using *dpll-bj-atms-in-trail dpll-bj-atms-of-ms-clauses-inv rtranclp-dpll-bj-inv* **by** *auto*

```
lemma rtranclp-dpll-bj-sat-iff:
  assumes dpll-bj** S T and inv S
  shows I ⊨sm clauses S ⟷ I ⊨sm clauses T
  using assms by (induction rule: rtranclp-induct)
    (auto dest!: dpll-bj-sat-iff simp: rtranclp-dpll-bj-inv)

lemma rtranclp-dpll-bj-atms-in-trail-in-set:
  assumes
    dpll-bj** S T and
    inv S
    atms-of-msu (clauses S) ⊆ A and
    atm-of ` (lits-of (trail S)) ⊆ A
  shows atm-of ` (lits-of (trail T)) ⊆ A
  using assms
    by (induction rule: rtranclp-induct)
      (auto dest: rtranclp-dpll-bj-inv
        simp add: dpll-bj-atms-in-trail-in-set rtranclp-dpll-bj-atms-of-ms-clauses-inv
          rtranclp-dpll-bj-inv)

lemma rtranclp-dpll-bj-all-decomposition-implies-inv:
  assumes
    dpll-bj** S T and
    inv S
    all-decomposition-implies-m (clauses S) (get-all-marked-decomposition (trail S))
  shows all-decomposition-implies-m (clauses T) (get-all-marked-decomposition (trail T))
  using assms by (induction rule: rtranclp-induct)
    (auto intro: dpll-bj-all-decomposition-implies-inv simp: rtranclp-dpll-bj-inv)

lemma rtranclp-dpll-bj-inv-incl-dpll-bj-inv-trancl:
  {(T, S). dpll-bj++ S T
    ∧ atms-of-msu (clauses S) ⊆ atms-of-ms A ∧ atm-of ` lits-of (trail S) ⊆ atms-of-ms A
    ∧ no-dup (trail S) ∧ inv S}
     ⊆ {(T, S). dpll-bj S T ∧ atms-of-msu (clauses S) ⊆ atms-of-ms A
        ∧ atm-of ` lits-of (trail S) ⊆ atms-of-ms A ∧ no-dup (trail S) ∧ inv S}+
    (is ?A ⊆ ?B+)
proof standard
  fix x
  assume x-A: x ∈ ?A
  obtain S T::'st where
    x[simp]: x = (T, S) by (cases x) auto
  have
    dpll-bj++ S T and
    atms-of-msu (clauses S) ⊆ atms-of-ms A and
    atm-of ` lits-of (trail S) ⊆ atms-of-ms A and
    no-dup (trail S) and
     inv S
    using x-A by auto
  then show x ∈ ?B+ unfolding x
    proof (induction rule: tranclp-induct)
      case base
      then show ?case by auto
    next
      case (step T U) note step = this(1) and ST = this(2) and IH = this(3)[OF this(4−7)]
        and N-A = this(4) and M-A = this(5) and nd = this(6) and inv = this(7)
```

have [*simp*]: *atms-of-msu* (*clauses S*) = *atms-of-msu* (*clauses T*)
using *step rtranclp-dpll-bj-atms-of-ms-clauses-inv tranclp-into-rtranclp inv* **by** *fastforce*
have *no-dup* (*trail T*)
using *local.step nd rtranclp-dpll-bj-no-dup tranclp-into-rtranclp inv* **by** *fastforce*
moreover have *atm-of* ‘ (*lits-of* (*trail T*)) ⊆ *atms-of-ms A*
by (*metis inv M-A N-A local.step rtranclp-dpll-bj-atms-in-trail-in-set tranclp-into-rtranclp*)
moreover have *inv T*
using *inv local.step rtranclp-dpll-bj-inv tranclp-into-rtranclp* **by** *fastforce*
ultimately have (*U*, *T*) ∈ *?B* **using** *ST N-A M-A inv* **by** *auto*
then show *?case* **using** *IH* **by** (*rule trancl-into-trancl2*)
qed
qed

lemma *wf-tranclp-dpll-bj*:
assumes *fin*: *finite A*
shows *wf* {(*T*, *S*). *dpll-bj*$^{++}$ *S T*
∧ *atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A* ∧ *atm-of* ‘ *lits-of* (*trail S*) ⊆ *atms-of-ms A*
∧ *no-dup* (*trail S*) ∧ *inv S*}
using *wf-trancl*[*OF wf-dpll-bj*[*OF fin*]] *rtranclp-dpll-bj-inv-incl-dpll-bj-inv-trancl*
by (*rule wf-subset*)

lemma *dpll-bj-sat-ext-iff*:
dpll-bj S T ⟹ *inv S* ⟹ *I*⊨*sextm clauses S* ⟷ *I*⊨*sextm clauses T*
by (*simp add*: *dpll-bj-clauses*)

lemma *rtranclp-dpll-bj-sat-ext-iff*:
dpll-bj** *S T* ⟹ *inv S* ⟹ *I*⊨*sextm clauses S* ⟷ *I*⊨*sextm clauses T*
by (*induction rule*: *rtranclp-induct*) (*simp-all add*: *rtranclp-dpll-bj-inv dpll-bj-sat-ext-iff*)

theorem *full-dpll-backjump-final-state*:
fixes *A* :: *'v literal multiset set* **and** *S T* :: *'st*
assumes
full: *full dpll-bj S T* **and**
atms-S: *atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A* **and**
atms-trail: *atm-of* ‘ *lits-of* (*trail S*) ⊆ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
finite A **and**
inv: *inv S* **and**
decomp: *all-decomposition-implies-m* (*clauses S*) (*get-all-marked-decomposition* (*trail S*))
shows *unsatisfiable* (*set-mset* (*clauses S*))
∨ (*trail T* ⊨*asm clauses S* ∧ *satisfiable* (*set-mset* (*clauses S*)))
proof −
have *st*: *dpll-bj*** *S T* **and** *no-step dpll-bj T*
using *full* **unfolding** *full-def* **by** *fast+*
moreover have *atms-of-msu* (*clauses T*) ⊆ *atms-of-ms A*
using *atms-S inv rtranclp-dpll-bj-atms-of-ms-clauses-inv st* **by** *blast*
moreover have *atm-of* ‘ *lits-of* (*trail T*) ⊆ *atms-of-ms A*
using *atms-S atms-trail inv rtranclp-dpll-bj-atms-in-trail-in-set st* **by** *auto*
moreover have *no-dup* (*trail T*)
using *n-d inv rtranclp-dpll-bj-no-dup st* **by** *blast*
moreover have *inv*: *inv T*
using *inv rtranclp-dpll-bj-inv st* **by** *blast*
moreover

have *decomp*: *all-decomposition-implies-m* (*clauses T*) (*get-all-marked-decomposition* (*trail T*))
  using ⟨*inv S*⟩ *decomp rtranclp-dpll-bj-all-decomposition-implies-inv st* **by** *blast*
ultimately have *unsatisfiable* (*set-mset* (*clauses T*))
  ∨ (*trail T* ⊨*asm clauses T* ∧ *satisfiable* (*set-mset* (*clauses T*)))
  using ⟨*finite A*⟩ *dpll-backjump-final-state* **by** *force*
then show *?thesis*
  by (*meson* ⟨*inv S*⟩ *rtranclp-dpll-bj-sat-iff satisfiable-carac st true-annots-true-cls*)
qed

corollary *full-dpll-backjump-final-state-from-init-state*:
  fixes *A* :: *'v literal multiset set* **and** *S T* :: *'st*
  assumes
    full: *full dpll-bj S T* **and**
    trail S = [] **and**
    clauses S = *N* **and**
    inv S
  shows *unsatisfiable* (*set-mset N*) ∨ (*trail T* ⊨*asm N* ∧ *satisfiable* (*set-mset N*))
  using *assms full-dpll-backjump-final-state*[*of S T set-mset N*] **by** *auto*

lemma *tranclp-dpll-bj-trail-mes-decreasing-prop*:
  assumes *dpll*: $dpll\text{-}bj^{++}$ *S T* **and** *inv*: *inv S* **and**
  N-A: *atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A* **and**
  M-A: *atm-of* ‘ *lits-of* (*trail S*) ⊆ *atms-of-ms A* **and**
  n-d: *no-dup* (*trail S*) **and**
  fin-A: *finite A*
  shows (*2+card* (*atms-of-ms A*)) ^ (*1+card* (*atms-of-ms A*))
        − μ_C (*1+card* (*atms-of-ms A*)) (*2+card* (*atms-of-ms A*)) (*trail-weight T*)
      < (*2+card* (*atms-of-ms A*)) ^ (*1+card* (*atms-of-ms A*))
        − μ_C (*1+card* (*atms-of-ms A*)) (*2+card* (*atms-of-ms A*)) (*trail-weight S*)
  using *dpll*
proof (*induction*)
  case *base*
  then show *?case*
    using *N-A M-A n-d dpll-bj-trail-mes-decreasing-prop fin-A inv* **by** *blast*
next
  case (*step T U*) **note** *st* = *this*(*1*) **and** *dpll* = *this*(*2*) **and** *IH* = *this*(*3*)
  have *atms-of-msu* (*clauses S*) = *atms-of-msu* (*clauses T*)
    using *rtranclp-dpll-bj-atms-of-ms-clauses-inv* **by** (*metis dpll-bj-clauses dpll-bj-inv inv st tranclpD*)
  then have *N-A'*: *atms-of-msu* (*clauses T*) ⊆ *atms-of-ms A*
    using *N-A* **by** *auto*
  moreover have *M-A'*: *atm-of* ‘ *lits-of* (*trail T*) ⊆ *atms-of-ms A*
    by (*meson M-A N-A inv rtranclp-dpll-bj-atms-in-trail-in-set st dpll tranclp.r-into-trancl tranclp-into-rtranclp tranclp-trans*)
  moreover have *nd*: *no-dup* (*trail T*)
    by (*metis inv n-d rtranclp-dpll-bj-no-dup st tranclp-into-rtranclp*)
  moreover have *inv T*
    by (*meson dpll dpll-bj-inv inv rtranclp-dpll-bj-inv st tranclp-into-rtranclp*)
  ultimately show *?case*
    using *IH dpll-bj-trail-mes-decreasing-prop*[*of T U A*] *dpll fin-A* **by** *linarith*
qed

end

2.4 CDCL

2.4.1 Learn and Forget

locale *learn-ops* =
  dpll-state trail clauses prepend-trail tl-trail $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
  for
    trail :: *'st* ⇒ (*'v*, *unit*, *unit*) *ann-literals* **and**
    clauses :: *'st* ⇒ *'v clauses* **and**
    prepend-trail :: (*'v*, *unit*, *unit*) *ann-literal* ⇒ *'st* ⇒ *'st* **and** *tl-trail* :: *'st* ⇒ *'st* **and**
    $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$:: *'v clause* ⇒ *'st* ⇒ *'st* +
  fixes
    learn-cond :: *'v clause* ⇒ *'st* ⇒ *bool*

begin
inductive *learn* :: *'st* ⇒ *'st* ⇒ *bool* **where**
clauses S ⊨*pm C* ⟹ *atms-of C* ⊆ *atms-of-msu* (*clauses S*) ∪ *atm-of* ‘ (*lits-of* (*trail S*))
  ⟹ *learn-cond C S*
  ⟹ *T* ∼ $add\text{-}cls_{NOT}$ *C S*
  ⟹ *learn S T*
inductive-cases $learn_{NOT}E$: *learn S T*

lemma *learn*-μ_C-*stable*:
  assumes *learn S T* **and** *no-dup* (*trail S*)
  shows μ_C *A B* (*trail-weight S*) = μ_C *A B* (*trail-weight T*)
  using *assms* **by** (*auto elim*: $learn_{NOT}E$)
end

locale *forget-ops* =
  dpll-state trail clauses prepend-trail tl-trail $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
  for
    trail :: *'st* ⇒ (*'v*, *unit*, *unit*) *ann-literals* **and**
    clauses :: *'st* ⇒ *'v clauses* **and**
    prepend-trail :: (*'v*, *unit*, *unit*) *ann-literal* ⇒ *'st* ⇒ *'st* **and** *tl-trail* :: *'st* ⇒ *'st* **and**
    $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$:: *'v clause* ⇒ *'st* ⇒ *'st* +
  fixes
    forget-cond :: *'v clause* ⇒ *'st* ⇒ *bool*
begin
inductive $forget_{NOT}$ :: *'st* ⇒ *'st* ⇒ *bool* **where**
$forget_{NOT}$:*clauses S* − *replicate-mset* (*count* (*clauses S*) *C*) *C* ⊨*pm C*
  ⟹ *forget-cond C S*
  ⟹ *C* ∈# *clauses S*
  ⟹ *T* ∼ $remove\text{-}cls_{NOT}$ *C S*
  ⟹ $forget_{NOT}$ *S T*
inductive-cases $forget_{NOT}E$: $forget_{NOT}$ *S T*

lemma *forget*-μ_C-*stable*:
  assumes $forget_{NOT}$ *S T*
  shows μ_C *A B* (*trail-weight S*) = μ_C *A B* (*trail-weight T*)
  using *assms* **by** (*auto elim*!: $forget_{NOT}E$)
end

locale *learn-and-forget*$_{NOT}$ =
  learn-ops trail clauses prepend-trail tl-trail $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ *learn-cond* +
  forget-ops trail clauses prepend-trail tl-trail $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ *forget-cond*
  for

trail :: *'st* ⇒ (*'v*, *unit*, *unit*) *ann-literals* **and**
clauses :: *'st* ⇒ *'v clauses* **and**
prepend-trail :: (*'v*, *unit*, *unit*) *ann-literal* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒ *'st* **and**
add-cls$_{NOT}$ *remove-cls*$_{NOT}$:: *'v clause* ⇒ *'st* ⇒ *'st* **and**
learn-cond forget-cond :: *'v clause* ⇒ *'st* ⇒ *bool*
begin
inductive *learn-and-forget*$_{NOT}$ :: *'st* ⇒ *'st* ⇒ *bool*
where
lf-learn: *learn S T* ⟹ *learn-and-forget*$_{NOT}$ *S T* |
lf-forget: *forget*$_{NOT}$ *S T* ⟹ *learn-and-forget*$_{NOT}$ *S T*
end

2.4.2 Definition of CDCL

locale *conflict-driven-clause-learning-ops* =
dpll-with-backjumping-ops trail clauses prepend-trail tl-trail add-cls$_{NOT}$ *remove-cls*$_{NOT}$
propagate-conds inv backjump-conds +
learn-and-forget$_{NOT}$ *trail clauses prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$ *learn-cond*
forget-cond
for
trail :: *'st* ⇒ (*'v*, *unit*, *unit*) *ann-literals* **and**
clauses :: *'st* ⇒ *'v clauses* **and**
prepend-trail :: (*'v*, *unit*, *unit*) *ann-literal* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒ *'st* **and**
add-cls$_{NOT}$ *remove-cls*$_{NOT}$:: *'v clause* ⇒ *'st* ⇒ *'st* **and**
propagate-conds :: (*'v*, *unit*, *unit*) *ann-literal* ⇒*'st* ⇒ *bool* **and**
inv :: *'st* ⇒ *bool* **and**
backjump-conds :: *'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool* **and**
learn-cond forget-cond :: *'v clause* ⇒ *'st* ⇒ *bool*
begin

inductive *cdcl*$_{NOT}$ :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
c-dpll-bj: *dpll-bj S S'* ⟹ *cdcl*$_{NOT}$ *S S'* |
c-learn: *learn S S'* ⟹ *cdcl*$_{NOT}$ *S S'* |
c-forget$_{NOT}$: *forget*$_{NOT}$ *S S'* ⟹ *cdcl*$_{NOT}$ *S S'*

lemma *cdcl*$_{NOT}$*-all-induct*[*consumes 1*, *case-names dpll-bj learn forget*$_{NOT}$]:
fixes *S T* :: *'st*
assumes *cdcl*$_{NOT}$ *S T* **and**
dpll: ⋀*T*. *dpll-bj S T* ⟹ *P S T* **and**
learning:
⋀*C T*. *clauses S* ⊨*pm C* ⟹
atms-of C ⊆ *atms-of-msu* (*clauses S*) ∪ *atm-of* ‘ (*lits-of* (*trail S*)) ⟹
T ∼ *add-cls*$_{NOT}$ *C S* ⟹
P S T **and**
forgetting: ⋀*C T*. *clauses S* − *replicate-mset* (*count* (*clauses S*) *C*) *C* ⊨*pm C* ⟹
C ∈# *clauses S* ⟹
T ∼ *remove-cls*$_{NOT}$ *C S* ⟹
P S T
shows *P S T*
using *assms*(*1*) **by** (*induction rule*: *cdcl*$_{NOT}$*.induct*)
(*auto intro*: *assms*(*2*, *3*, *4*) *elim*!: *learn*$_{NOT}$*E forget*$_{NOT}$*E*)+

lemma *cdcl*$_{NOT}$*-no-dup*:
assumes

cdcl$_{NOT}$ *S T* **and**
inv S **and**
no-dup (*trail S*)
shows *no-dup* (*trail T*)
using *assms* **by** (*induction rule*: *cdcl*$_{NOT}$*-all-induct*) (*auto intro*: *dpll-bj-no-dup*)

Consistency of the trail **lemma** *cdcl*$_{NOT}$*-consistent*:
assumes
cdcl$_{NOT}$ *S T* **and**
inv S **and**
no-dup (*trail S*)
shows *consistent-interp* (*lits-of* (*trail T*))
using *cdcl*$_{NOT}$*-no-dup*[*OF assms*] *distinctconsistent-interp* **by** *fast*

The subtle problem here is that tautologies can be removed, meaning that some variable can disappear of the problem. It is also possible that some variable of the trail are not in the clauses anymore.

lemma *cdcl*$_{NOT}$*-atms-of-ms-clauses-decreasing*:
assumes *cdcl*$_{NOT}$ *S T***and** *inv S* **and** *no-dup* (*trail S*)
shows *atms-of-msu* (*clauses T*) ⊆ *atms-of-msu* (*clauses S*) ∪ *atm-of* ‘ (*lits-of* (*trail S*))
using *assms* **by** (*induction rule*: *cdcl*$_{NOT}$*-all-induct*)
(*auto dest*!: *dpll-bj-atms-of-ms-clauses-inv set-mp simp add*: *atms-of-ms-def Union-eq*)

lemma *cdcl*$_{NOT}$*-atms-in-trail*:
assumes *cdcl*$_{NOT}$ *S T***and** *inv S* **and** *no-dup* (*trail S*)
and *atm-of* ‘ (*lits-of* (*trail S*)) ⊆ *atms-of-msu* (*clauses S*)
shows *atm-of* ‘ (*lits-of* (*trail T*)) ⊆ *atms-of-msu* (*clauses S*)
using *assms* **by** (*induction rule*: *cdcl*$_{NOT}$*-all-induct*) (*auto simp add*: *dpll-bj-atms-in-trail*)

lemma *cdcl*$_{NOT}$*-atms-in-trail-in-set*:
assumes
cdcl$_{NOT}$ *S T* **and** *inv S* **and** *no-dup* (*trail S*) **and**
atms-of-msu (*clauses S*) ⊆ *A* **and**
atm-of ‘ (*lits-of* (*trail S*)) ⊆ *A*
shows *atm-of* ‘ (*lits-of* (*trail T*)) ⊆ *A*
using *assms*
by (*induction rule*: *cdcl*$_{NOT}$*-all-induct*)
(*simp-all add*: *dpll-bj-atms-in-trail-in-set dpll-bj-atms-of-ms-clauses-inv*)

lemma *cdcl*$_{NOT}$*-all-decomposition-implies*:
assumes *cdcl*$_{NOT}$ *S T* **and** *inv S* **and** *n-d*[*simp*]: *no-dup* (*trail S*) **and**
all-decomposition-implies-m (*clauses S*) (*get-all-marked-decomposition* (*trail S*))
shows
all-decomposition-implies-m (*clauses T*) (*get-all-marked-decomposition* (*trail T*))
using *assms*(*1*,*2*,*4*)
proof (*induction rule*: *cdcl*$_{NOT}$*-all-induct*)
case *dpll-bj*
then show *?case*
using *dpll-bj-all-decomposition-implies-inv n-d* **by** *blast*
next
case *learn*
then show *?case* **by** (*auto simp add*: *all-decomposition-implies-def*)
next
case (*forget*$_{NOT}$ *C T*) **note** *cls-C* = *this*(*1*) **and** *C* = *this*(*2*) **and** *T* = *this*(*3*) **and** *iniv* = *this*(*4*)
and

decomp = *this*(*5*)
show *?case*
unfolding *all-decomposition-implies-def Ball-def*
proof (*intro allI*, *clarify*)
fix *a b*
assume (*a*, *b*) ∈ *set* (*get-all-marked-decomposition* (*trail T*))
then have *unmark a* ∪ *set-mset* (*clauses S*) ⊨*ps unmark b*
using *decomp T* **by** (*auto simp add*: *all-decomposition-implies-def*)
moreover
have *C* ∈ *set-mset* (*clauses S*)
by (*simp add*: *C*)
then have *set-mset* (*clauses T*) ⊨*ps set-mset* (*clauses S*)
by (*metis* (*no-types*) *T clauses-remove-cls*$_{NOT}$ *cls-C insert-Diff order-refl set-mset-minus-replicate-mset*(*1*) *state-eq*$_{NOT}$*-clauses true-clss-clss-def true-clss-clss-insert*)
ultimately show *unmark a* ∪ *set-mset* (*clauses T*) ⊨*ps unmark b*
using *true-clss-clss-generalise-true-clss-clss* **by** *blast*
qed
qed

Extension of models **lemma** *cdcl*$_{NOT}$*-bj-sat-ext-iff*:
assumes *cdcl*$_{NOT}$ *S T***and** *inv S* **and** *n-d*: *no-dup* (*trail S*)
shows *I*⊨*sextm clauses S* ⟷ *I*⊨*sextm clauses T*
using *assms*
proof (*induction rule*:*cdcl*$_{NOT}$*-all-induct*)
case *dpll-bj*
then show *?case* **by** (*simp add*: *dpll-bj-clauses*)
next
case (*learn C T*) **note** *T* = *this*(*3*)
{ **fix** *J*
assume
I ⊨*sextm clauses S* **and**
I ⊆ *J* **and**
tot: *total-over-m J* (*set-mset* ({#*C*#} + (*clauses S*))) **and**
cons: *consistent-interp J*
then have *J* ⊨*sm clauses S* **unfolding** *true-clss-ext-def* **by** *auto*

moreover
with ‹*clauses S*⊨*pm C*› **have** *J* ⊨ *C*
using *tot cons* **unfolding** *true-clss-cls-def* **by** *auto*
ultimately have *J* ⊨*sm* {#*C*#} + *clauses S* **by** *auto*
}
then have *H*: *I* ⊨*sextm* (*clauses S*) ⟹ *I* ⊨*sext insert C* (*set-mset* (*clauses S*))
unfolding *true-clss-ext-def* **by** *auto*
show *?case*
apply *standard*
using *T n-d* **apply** (*auto simp add*: *H*)[]
using *T n-d* **apply** *simp*
by (*metis Diff-insert-absorb insert-subset subsetI subset-antisym true-clss-ext-decrease-right-remove-r*)
next
case (*forget*$_{NOT}$ *C T*) **note** *cls-C* = *this*(*1*) **and** *T* = *this*(*3*)
{ **fix** *J*
assume

$I \models sext\ set\text{-}mset\ (clauses\ S) - \{C\}$ **and**
$I \subseteq J$ **and**
tot: *total-over-m* J (*set-mset* (*clauses* S)) **and**
cons: *consistent-interp* J
then have $J \models s\ set\text{-}mset\ (clauses\ S) - \{C\}$
unfolding *true-clss-ext-def* **by** (*meson Diff-subset total-over-m-subset*)

moreover
with *cls-C* **have** $J \models C$
using *tot cons* **unfolding** *true-clss-cls-def*
by (*metis Un-commute* $forget_{NOT}$*.hyps*(*2*) *insert-Diff insert-is-Un mem-set-mset-iff order-refl set-mset-minus-replicate-mset*(*1*))
ultimately have $J \models sm\ (clauses\ S)$ **by** (*metis insert-Diff-single true-clss-insert*)
}
then have H: $I \models sext\ set\text{-}mset\ (clauses\ S) - \{C\} \Longrightarrow I \models sextm\ (clauses\ S)$
unfolding *true-clss-ext-def* **by** *blast*
show *?case* **using** T **by** (*auto simp*: *true-clss-ext-decrease-right-remove-r* H)
qed

end — end of *conflict-driven-clause-learning-ops*

2.5 CDCL with invariant

locale *conflict-driven-clause-learning* =
conflict-driven-clause-learning-ops +
assumes $cdcl_{NOT}$*-inv*: $\bigwedge S\ T.\ cdcl_{NOT}\ S\ T \Longrightarrow inv\ S \Longrightarrow inv\ T$
begin
sublocale *dpll-with-backjumping*
apply *unfold-locales*
using $cdcl_{NOT}$*.simps* $cdcl_{NOT}$*-inv* **by** *auto*

lemma *rtranclp-*$cdcl_{NOT}$*-inv*:
${cdcl_{NOT}}^{**}\ S\ T \Longrightarrow inv\ S \Longrightarrow inv\ T$
by (*induction rule*: *rtranclp-induct*) (*auto simp add*: $cdcl_{NOT}$*-inv*)

lemma *rtranclp-*$cdcl_{NOT}$*-no-dup*:
assumes ${cdcl_{NOT}}^{**}\ S\ T$ **and** *inv* S
and *no-dup* (*trail* S)
shows *no-dup* (*trail* T)
using *assms* **by** (*induction rule*: *rtranclp-induct*) (*auto intro*: $cdcl_{NOT}$*-no-dup rtranclp-*$cdcl_{NOT}$*-inv*)

lemma *rtranclp-*$cdcl_{NOT}$*-trail-clauses-bound*:
assumes
cdcl: ${cdcl_{NOT}}^{**}\ S\ T$ **and**
inv: *inv* S **and**
n-d: *no-dup* (*trail* S) **and**
atms-clauses-S: *atms-of-msu* (*clauses* S) $\subseteq A$ **and**
atms-trail-S: *atm-of* ‘(*lits-of* (*trail* S)) $\subseteq A$
shows *atm-of* ‘(*lits-of* (*trail* T)) $\subseteq A \wedge$ *atms-of-msu* (*clauses* T) $\subseteq A$
using *cdcl*
proof (*induction rule*: *rtranclp-induct*)
case *base*
then show *?case* **using** *atms-clauses-S atms-trail-S* **by** *simp*
next
case (*step* T U) **note** $st = this(1)$ **and** $cdcl_{NOT} = this(2)$ **and** $IH = this(3)$
have *inv* T **using** *inv st rtranclp-*$cdcl_{NOT}$*-inv* **by** *blast*

have *no-dup* (*trail* *T*)
using *rtranclp-cdcl*$_{NOT}$*-no-dup*[*of* *S* *T*] *st* *cdcl*$_{NOT}$ *inv* *n-d* **by** *blast*
then have *atms-of-msu* (*clauses* *U*) ⊆ *A*
using *cdcl*$_{NOT}$*-atms-of-ms-clauses-decreasing*[*OF* *cdcl*$_{NOT}$] *IH* *n-d* ‹*inv* *T*› **by** *auto*
moreover
have *atm-of* ‘(*lits-of* (*trail* *U*)) ⊆ *A*
using *cdcl*$_{NOT}$*-atms-in-trail-in-set*[*OF* *cdcl*$_{NOT}$, *of* *A*] ‹*no-dup* (*trail* *T*)›
by (*meson* *atms-trail-S* *atms-clauses-S* *IH* ‹*inv* *T*› *cdcl*$_{NOT}$)
ultimately show *?case* **by** *fast*
qed

lemma *rtranclp-cdcl*$_{NOT}$*-all-decomposition-implies*:
assumes *cdcl*$_{NOT}$** *S* *T* **and** *inv* *S* **and** *no-dup* (*trail* *S*) **and**
all-decomposition-implies-m (*clauses* *S*) (*get-all-marked-decomposition* (*trail* *S*))
shows
all-decomposition-implies-m (*clauses* *T*) (*get-all-marked-decomposition* (*trail* *T*))
using *assms* **by** (*induction*)
(*auto* *intro*: *rtranclp-cdcl*$_{NOT}$*-inv* *cdcl*$_{NOT}$*-all-decomposition-implies* *rtranclp-cdcl*$_{NOT}$*-no-dup*)

lemma *rtranclp-cdcl*$_{NOT}$*-bj-sat-ext-iff*:
assumes *cdcl*$_{NOT}$** *S* *T***and** *inv* *S* **and** *no-dup* (*trail* *S*)
shows *I*⊨*sextm* *clauses* *S* ⟷ *I*⊨*sextm* *clauses* *T*
using *assms* **apply** (*induction* *rule*: *rtranclp-induct*)
using *cdcl*$_{NOT}$*-bj-sat-ext-iff* **by** (*auto* *intro*: *rtranclp-cdcl*$_{NOT}$*-inv* *rtranclp-cdcl*$_{NOT}$*-no-dup*)

definition *cdcl*$_{NOT}$*-NOT-all-inv* **where**
cdcl$_{NOT}$*-NOT-all-inv* *A* *S* ⟷ (*finite* *A* ∧ *inv* *S* ∧ *atms-of-msu* (*clauses* *S*) ⊆ *atms-of-ms* *A*
∧ *atm-of* ‘ *lits-of* (*trail* *S*) ⊆ *atms-of-ms* *A* ∧ *no-dup* (*trail* *S*))

lemma *cdcl*$_{NOT}$*-NOT-all-inv*:
assumes *cdcl*$_{NOT}$** *S* *T* **and** *cdcl*$_{NOT}$*-NOT-all-inv* *A* *S*
shows *cdcl*$_{NOT}$*-NOT-all-inv* *A* *T*
using *assms* **unfolding** *cdcl*$_{NOT}$*-NOT-all-inv-def*
by (*simp* *add*: *rtranclp-cdcl*$_{NOT}$*-inv* *rtranclp-cdcl*$_{NOT}$*-no-dup* *rtranclp-cdcl*$_{NOT}$*-trail-clauses-bound*)

abbreviation *learn-or-forget* **where**
learn-or-forget *S* *T* ≡ (λ*S* *T*. *learn* *S* *T* ∨ *forget*$_{NOT}$ *S* *T*) *S* *T*

lemma *rtranclp-learn-or-forget-cdcl*$_{NOT}$:
learn-or-forget** *S* *T* ⟹ *cdcl*$_{NOT}$** *S* *T*
using *rtranclp-mono*[*of* *learn-or-forget* *cdcl*$_{NOT}$] *cdcl*$_{NOT}$*.c-learn* *cdcl*$_{NOT}$*.c-forget*$_{NOT}$ **by** *blast*

lemma *learn-or-forget-dpll-*μ_C:
assumes
l-f: *learn-or-forget*** *S* *T* **and**
dpll: *dpll-bj* *T* *U* **and**
inv: *cdcl*$_{NOT}$*-NOT-all-inv* *A* *S*
shows (*2*+*card* (*atms-of-ms* *A*)) ^ (*1*+*card* (*atms-of-ms* *A*))
− μ_C (*1*+*card* (*atms-of-ms* *A*)) (*2*+*card* (*atms-of-ms* *A*)) (*trail-weight* *U*)
< (*2*+*card* (*atms-of-ms* *A*)) ^ (*1*+*card* (*atms-of-ms* *A*))
− μ_C (*1*+*card* (*atms-of-ms* *A*)) (*2*+*card* (*atms-of-ms* *A*)) (*trail-weight* *S*)
(**is** *?μ* *U* < *?μ* *S*)
proof −
have *?μ* *S* = *?μ* *T*

    using *l-f*
    proof (*induction*)
      case *base*
      then show *?case* **by** *simp*
    next
      case (*step T U*)
      moreover then have *no-dup* (*trail T*)
        using *rtranclp-cdcl*$_{NOT}$*-no-dup*[*of S T*] *cdcl*$_{NOT}$*-NOT-all-inv-def inv*
        rtranclp-learn-or-forget-cdcl$_{NOT}$ **by** *auto*
      ultimately show *?case*
        using *forget-*μ_C*-stable learn-*μ_C*-stable inv* **unfolding** *cdcl*$_{NOT}$*-NOT-all-inv-def* **by** *presburger*
    qed
  moreover have *cdcl*$_{NOT}$*-NOT-all-inv A T*
     using *rtranclp-learn-or-forget-cdcl*$_{NOT}$ *cdcl*$_{NOT}$*-NOT-all-inv l-f inv* **by** *blast*
  ultimately show *?thesis*
    using *dpll-bj-trail-mes-decreasing-prop*[*of T U A*, *OF dpll*] *finite*
    unfolding *cdcl*$_{NOT}$*-NOT-all-inv-def* **by** *linarith*
qed

lemma *infinite-cdcl*$_{NOT}$*-exists-learn-and-forget-infinite-chain*:
  assumes
    $\bigwedge$*i. cdcl*$_{NOT}$ (*f i*) (*f*(*Suc i*)) **and**
    inv: *cdcl*$_{NOT}$*-NOT-all-inv A* (*f 0*)
  shows ∃*j*. ∀*i*≥*j*. *learn-or-forget* (*f i*) (*f* (*Suc i*))
  using *assms*
proof (*induction* (*2*+*card* (*atms-of-ms A*)) ^ (*1*+*card* (*atms-of-ms A*))
    − μ_C (*1*+*card* (*atms-of-ms A*)) (*2*+*card* (*atms-of-ms A*)) (*trail-weight* (*f 0*))
    arbitrary: *f*
    rule: *nat-less-induct-case*)
  case (*Suc n*) **note** *IH* = *this*(*1*) **and** *μ* = *this*(*2*) **and** *cdcl*$_{NOT}$ = *this*(*3*) **and** *inv* = *this*(*4*)
  consider
      (*dpll-end*) ∃*j*. ∀*i*≥*j*. *learn-or-forget* (*f i*) (*f* (*Suc i*))
    | (*dpll-more*) ¬(∃*j*. ∀*i*≥*j*. *learn-or-forget* (*f i*) (*f* (*Suc i*)))
    by *blast*
  then show *?case*
    proof *cases*
      case *dpll-end*
      then show *?thesis* **by** *auto*
    next
      case *dpll-more*
      then have *j*: ∃*i*. ¬ *learn* (*f i*) (*f* (*Suc i*)) ∧ ¬*forget*$_{NOT}$ (*f i*) (*f* (*Suc i*))
        by *blast*
      obtain *i* **where**
        ¬*learn* (*f i*) (*f* (*Suc i*)) ∧ ¬*forget*$_{NOT}$ (*f i*) (*f* (*Suc i*)) **and**
        ∀*k*<*i*. *learn-or-forget* (*f k*) (*f* (*Suc k*))
        proof −
          obtain i_0 **where** ¬ *learn* (*f* i_0) (*f* (*Suc* i_0)) ∧ ¬*forget*$_{NOT}$ (*f* i_0) (*f* (*Suc* i_0))
            using *j* **by** *auto*
          then have {*i*. *i*≤i_0 ∧ ¬ *learn* (*f i*) (*f* (*Suc i*)) ∧ ¬*forget*$_{NOT}$ (*f i*) (*f* (*Suc i*))} ≠ {}
            by *auto*
          let *?I* = {*i*. *i*≤i_0 ∧ ¬ *learn* (*f i*) (*f* (*Suc i*)) ∧ ¬*forget*$_{NOT}$ (*f i*) (*f* (*Suc i*))}
          let *?i* = *Min ?I*
          have *finite ?I*
            by *auto*
          have ¬ *learn* (*f ?i*) (*f* (*Suc ?i*)) ∧ ¬*forget*$_{NOT}$ (*f ?i*) (*f* (*Suc ?i*))

```
        using Min-in[OF ‹finite ?I› ‹?I ≠ {}›] by auto
      moreover have ∀k<?i. learn-or-forget (f k) (f (Suc k))
        using Min.coboundedI[of {i. i ≤ i₀ ∧ ¬ learn (f i) (f (Suc i)) ∧ ¬ forget_NOT (f i)
          (f (Suc i))}, simplified]
        by (meson ‹¬ learn (f i₀) (f (Suc i₀)) ∧ ¬ forget_NOT (f i₀) (f (Suc i₀))› less-imp-le
          dual-order.trans not-le)
      ultimately show ?thesis using that by blast
    qed
  def g ≡ λn. f (n + Suc i)
  have dpll-bj (f i) (g 0)
    using ‹¬ learn (f i) (f (Suc i)) ∧ ¬ forget_NOT (f i) (f (Suc i))› cdcl_NOT cdcl_NOT.cases
    g-def by auto
  {
    fix j
    assume j ≤ i
    then have learn-or-forget** (f 0) (f j)
      apply (induction j)
       apply simp
      by (metis (no-types, lifting) Suc-leD Suc-le-lessD rtranclp.simps
        ‹∀k<i. learn (f k) (f (Suc k)) ∨ forget_NOT (f k) (f (Suc k))›)
  }
  then have learn-or-forget** (f 0) (f i) by blast
  then have (2 + card (atms-of-ms A)) ^ (1 + card (atms-of-ms A))
      − μ_C (1 + card (atms-of-ms A)) (2 + card (atms-of-ms A)) (trail-weight (g 0))
    < (2 + card (atms-of-ms A)) ^ (1 + card (atms-of-ms A))
     − μ_C (1 + card (atms-of-ms A)) (2 + card (atms-of-ms A)) (trail-weight (f 0))
    using learn-or-forget-dpll-μ_C[of f 0 f i g 0 A] inv ‹dpll-bj (f i) (g 0)›
    unfolding cdcl_NOT-NOT-all-inv-def by linarith

  moreover have cdcl_NOT-i: cdcl_NOT** (f 0) (g 0)
    using rtranclp-learn-or-forget-cdcl_NOT[of f 0 f i] ‹learn-or-forget** (f 0) (f i)›
    cdcl_NOT[of i] unfolding g-def by auto
  moreover have ⋀i. cdcl_NOT (g i) (g (Suc i))
    using cdcl_NOT g-def by auto
  moreover have cdcl_NOT-NOT-all-inv A (g 0)
    using inv cdcl_NOT-i rtranclp-cdcl_NOT-trail-clauses-bound g-def cdcl_NOT-NOT-all-inv by auto
  ultimately obtain j where j: ⋀i. i≥j ⟹ learn-or-forget (g i) (g (Suc i))
    using IH unfolding μ[symmetric] by presburger
  show ?thesis
    proof
      {
        fix k
        assume k ≥ j + Suc i
        then have learn-or-forget (f k) (f (Suc k))
          using j[of k−Suc i] unfolding g-def by auto
      }
      then show ∀k≥j+Suc i. learn-or-forget (f k) (f (Suc k))
        by auto
    qed
  qed
next
  case 0 note H = this(1) and cdcl_NOT = this(2) and inv = this(3)
  show ?case
    proof (rule ccontr)
      assume ¬ ?case
```

then have j: $\exists\, i.\ \neg\ learn\ (f\ i)\ (f\ (Suc\ i)) \wedge \neg forget_{NOT}\ (f\ i)\ (f\ (Suc\ i))$
by *blast*
obtain i **where**
$\neg learn\ (f\ i)\ (f\ (Suc\ i)) \wedge \neg forget_{NOT}\ (f\ i)\ (f\ (Suc\ i))$ **and**
$\forall\, k{<}i.\ learn\text{-}or\text{-}forget\ (f\ k)\ (f\ (Suc\ k))$
proof −
obtain i_0 **where** $\neg\ learn\ (f\ i_0)\ (f\ (Suc\ i_0)) \wedge \neg forget_{NOT}\ (f\ i_0)\ (f\ (Suc\ i_0))$
using j **by** *auto*
then have $\{i.\ i{\le}i_0 \wedge\ \neg\ learn\ (f\ i)\ (f\ (Suc\ i)) \wedge \neg forget_{NOT}\ (f\ i)\ (f\ (Suc\ i))\} \neq \{\}$
by *auto*
let $?I = \{i.\ i{\le}i_0 \wedge\ \neg\ learn\ (f\ i)\ (f\ (Suc\ i)) \wedge \neg forget_{NOT}\ (f\ i)\ (f\ (Suc\ i))\}$
let $?i = Min\ ?I$
have *finite ?I*
by *auto*
have $\neg\ learn\ (f\ ?i)\ (f\ (Suc\ ?i)) \wedge \neg forget_{NOT}\ (f\ ?i)\ (f\ (Suc\ ?i))$
using *Min-in*[*OF* ‹*finite ?I*› ‹$?I \neq \{\}$›] **by** *auto*
moreover have $\forall\, k{<}?i.\ learn\text{-}or\text{-}forget\ (f\ k)\ (f\ (Suc\ k))$
using *Min.coboundedI*[*of* $\{i.\ i \le i_0 \wedge \neg\ learn\ (f\ i)\ (f\ (Suc\ i)) \wedge \neg\ forget_{NOT}\ (f\ i)$
$(f\ (Suc\ i))\}$, *simplified*]
by (*meson* ‹$\neg\ learn\ (f\ i_0)\ (f\ (Suc\ i_0)) \wedge \neg\ forget_{NOT}\ (f\ i_0)\ (f\ (Suc\ i_0))$› *less-imp-le*
dual-order.trans not-le)
ultimately show *?thesis* **using** *that* **by** *blast*
qed
have *dpll-bj* $(f\ i)\ (f\ (Suc\ i))$
using ‹$\neg\ learn\ (f\ i)\ (f\ (Suc\ i)) \wedge \neg\ forget_{NOT}\ (f\ i)\ (f\ (Suc\ i))$› $cdcl_{NOT}$ $cdcl_{NOT}$*.cases*
by *blast*
{
fix j
assume $j \le i$
then have $learn\text{-}or\text{-}forget^{**}\ (f\ 0)\ (f\ j)$
apply (*induction* j)
apply *simp*
by (*metis* (*no-types*, *lifting*) *Suc-leD Suc-le-lessD rtranclp.simps*
‹$\forall\, k{<}i.\ learn\ (f\ k)\ (f\ (Suc\ k)) \vee forget_{NOT}\ (f\ k)\ (f\ (Suc\ k))$›)
}
then have $learn\text{-}or\text{-}forget^{**}\ (f\ 0)\ (f\ i)$ **by** *blast*

then show *False*
using *learn-or-forget-dpll-*μ_C[*of f 0 f i f* (*Suc i*) *A*] *inv 0*
‹*dpll-bj* $(f\ i)\ (f\ (Suc\ i))$› **unfolding** $cdcl_{NOT}$*-NOT-all-inv-def* **by** *linarith*
qed
qed

lemma *wf-*$cdcl_{NOT}$*-no-learn-and-forget-infinite-chain*:
assumes
no-infinite-lf: $\bigwedge f\ j.\ \neg\ (\forall\, i{\ge}j.\ learn\text{-}or\text{-}forget\ (f\ i)\ (f\ (Suc\ i)))$
shows *wf* $\{(T, S).\ cdcl_{NOT}\ S\ T \wedge cdcl_{NOT}\text{-}NOT\text{-}all\text{-}inv\ A\ S\}$ (**is** *wf* $\{(T, S).\ cdcl_{NOT}\ S\ T$
$\wedge\ ?inv\ S\}$)
unfolding *wf-iff-no-infinite-down-chain*
proof (*rule ccontr*)
assume $\neg\ \neg\ (\exists f.\ \forall\, i.\ (f\ (Suc\ i), f\ i) \in \{(T, S).\ cdcl_{NOT}\ S\ T \wedge\ ?inv\ S\})$
then obtain f **where**
$\forall\, i.\ cdcl_{NOT}\ (f\ i)\ (f\ (Suc\ i)) \wedge\ ?inv\ (f\ i)$
by *fast*
then have $\exists\, j.\ \forall\, i{\ge}j.\ learn\text{-}or\text{-}forget\ (f\ i)\ (f\ (Suc\ i))$

using *infinite-cdcl*$_{NOT}$*-exists-learn-and-forget-infinite-chain*[*of f*] **by** *meson*
then show *False* **using** *no-infinite-lf* **by** *blast*
qed

lemma *inv-and-tranclp-cdcl-*$_{NOT}$*-tranclp-cdcl*$_{NOT}$*-and-inv*:
$cdcl_{NOT}^{++}$ S T ∧ *cdcl*$_{NOT}$*-NOT-all-inv* A S ⟷ (λS T. *cdcl*$_{NOT}$ S T ∧ *cdcl*$_{NOT}$*-NOT-all-inv* A S)$^{++}$ S T
(**is** *?A* ∧ *?I* ⟷ *?B*)
proof
assume *?A* ∧ *?I*
then have *?A* **and** *?I* **by** *blast+*
then show *?B*
apply *induction*
apply (*simp add*: *tranclp.r-into-trancl*)
by (*metis* (*no-types*, *lifting*) *cdcl*$_{NOT}$*-NOT-all-inv tranclp.simps tranclp-into-rtranclp*)
next
assume *?B*
then have *?A* **by** *induction auto*
moreover have *?I* **using** ‹*?B*› *tranclpD* **by** *fastforce*
ultimately show *?A* ∧ *?I* **by** *blast*
qed

lemma *wf-tranclp-cdcl*$_{NOT}$*-no-learn-and-forget-infinite-chain*:
assumes
no-infinite-lf: ⋀*f j*. ¬ (∀ *i*≥*j*. *learn-or-forget* (*f i*) (*f* (*Suc i*)))
shows *wf* {(T, S). $cdcl_{NOT}^{++}$ S T ∧ *cdcl*$_{NOT}$*-NOT-all-inv* A S}
using *wf-trancl*[*OF wf-cdcl*$_{NOT}$*-no-learn-and-forget-infinite-chain*[*OF no-infinite-lf*]]
apply (*rule wf-subset*)
by (*auto simp*: *trancl-set-tranclp inv-and-tranclp-cdcl-*$_{NOT}$*-tranclp-cdcl*$_{NOT}$*-and-inv*)

lemma *cdcl*$_{NOT}$*-final-state*:
assumes
n-s: *no-step cdcl*$_{NOT}$ S **and**
inv: *cdcl*$_{NOT}$*-NOT-all-inv* A S **and**
decomp: *all-decomposition-implies-m* (*clauses* S) (*get-all-marked-decomposition* (*trail* S))
shows *unsatisfiable* (*set-mset* (*clauses* S))
∨ (*trail* S ⊨*asm clauses* S ∧ *satisfiable* (*set-mset* (*clauses* S)))
proof −
have *n-s'*: *no-step dpll-bj* S
using *n-s* **by** (*auto simp*: *cdcl*$_{NOT}$*.simps*)
show *?thesis*
apply (*rule dpll-backjump-final-state*[*of S A*])
using *inv decomp n-s'* **unfolding** *cdcl*$_{NOT}$*-NOT-all-inv-def* **by** *auto*
qed

lemma *full-cdcl*$_{NOT}$*-final-state*:
assumes
full: *full cdcl*$_{NOT}$ S T **and**
inv: *cdcl*$_{NOT}$*-NOT-all-inv* A S **and**
n-d: *no-dup* (*trail* S) **and**
decomp: *all-decomposition-implies-m* (*clauses* S) (*get-all-marked-decomposition* (*trail* S))
shows *unsatisfiable* (*set-mset* (*clauses* T))
∨ (*trail* T ⊨*asm clauses* T ∧ *satisfiable* (*set-mset* (*clauses* T)))
proof −
have *st*: $cdcl_{NOT}^{**}$ S T **and** *n-s*: *no-step cdcl*$_{NOT}$ T

using *full* **unfolding** *full-def* **by** *blast+*
have *n-s'*: $cdcl_{NOT}$*-NOT-all-inv A T*
using $cdcl_{NOT}$*-NOT-all-inv inv st* **by** *blast*
moreover have *all-decomposition-implies-m* (*clauses T*) (*get-all-marked-decomposition* (*trail T*))
using $cdcl_{NOT}$*-NOT-all-inv-def decomp inv rtranclp-*$cdcl_{NOT}$*-all-decomposition-implies st* **by** *auto*
ultimately show *?thesis*
using $cdcl_{NOT}$*-final-state n-s* **by** *blast*
qed

end — end of *conflict-driven-clause-learning*

2.6 Termination

2.6.1 Restricting learn and forget

locale *conflict-driven-clause-learning-learning-before-backjump-only-distinct-learnt* =
conflict-driven-clause-learning trail clauses prepend-trail tl-trail add-cls$_{NOT}$ *remove-cls*$_{NOT}$
propagate-conds inv backjump-conds
$\lambda C\ S.\ $ *distinct-mset* $C \wedge \neg$*tautology* $C \wedge$ *learn-restrictions* $C\ S \wedge$
$(\exists F\ K\ d\ F'\ C'\ L.\ $ *trail* $S = F'$ @ *Marked* K () # $F \wedge C = C' + \{\#L\#\} \wedge F \models$*as CNot* C'
$\wedge\ C' + \{\#L\#\} \notin\#$ *clauses* S)
$\lambda C\ S.\ \neg(\exists F'\ F\ K\ d\ L.$ *trail* $S = F'$ @ *Marked* K () # $F \wedge F \models$*as CNot* $(C - \{\#L\#\}))$
$\wedge$ *forget-restrictions* $C\ S$
for
trail :: *'st* ⇒ (*'v*, *unit*, *unit*) *ann-literals* **and**
clauses :: *'st* ⇒ *'v clauses* **and**
prepend-trail :: (*'v*, *unit*, *unit*) *ann-literal* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒ *'st* **and**
add-cls$_{NOT}$ *remove-cls*$_{NOT}$:: *'v clause* ⇒ *'st* ⇒ *'st* **and**
propagate-conds :: (*'v*, *unit*, *unit*) *ann-literal* ⇒ *'st* ⇒ *bool* **and**
inv :: *'st* ⇒ *bool* **and**
backjump-conds :: *'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool* **and**
learn-restrictions forget-restrictions :: *'v clause* ⇒ *'st* ⇒ *bool*
begin

lemma $cdcl_{NOT}$*-learn-all-induct*[*consumes 1*, *case-names dpll-bj learn forget*$_{NOT}$]:
fixes $S\ T$ :: *'st*
assumes $cdcl_{NOT}$ $S\ T$ **and**
dpll: $\bigwedge T.$ *dpll-bj* $S\ T \Longrightarrow P\ S\ T$ **and**
learning:
$\bigwedge C\ F\ K\ F'\ C'\ L\ T.$ *clauses* $S \models$*pm* C
$\Longrightarrow$ *atms-of* $C \subseteq$ *atms-of-msu* (*clauses* S) $\cup$ *atm-of* ‘ (*lits-of* (*trail* S))
$\Longrightarrow$ *distinct-mset* $C \Longrightarrow \neg$ *tautology* $C \Longrightarrow$ *learn-restrictions* $C\ S$
$\Longrightarrow$ *trail* $S = F'$ @ *Marked* K () # $F \Longrightarrow C = C' + \{\#L\#\} \Longrightarrow F \models$*as CNot* C'
$\Longrightarrow C' + \{\#L\#\} \notin\#$ *clauses* $S \Longrightarrow T \sim$ *add-cls*$_{NOT}$ $C\ S$
$\Longrightarrow P\ S\ T$ **and**
forgetting: $\bigwedge C\ T.$ *clauses* $S -$ *replicate-mset* (*count* (*clauses* S) C) $C \models$*pm* C
$\Longrightarrow C \in\#$ *clauses* S
$\Longrightarrow \neg(\exists F'\ F\ K\ L.$ *trail* $S = F'$ @ *Marked* K () # $F \wedge F \models$*as CNot* $(C - \{\#L\#\}))$
$\Longrightarrow T \sim$ *remove-cls*$_{NOT}$ $C\ S$
$\Longrightarrow$ *forget-restrictions* $C\ S \Longrightarrow P\ S\ T$
shows $P\ S\ T$
using *assms*(*1*)
apply (*induction rule*: $cdcl_{NOT}$*.induct*)
apply (*auto dest*: *assms*(*2*) *simp add*: *learn-ops-axioms*)[]
apply (*auto elim*!: *learn-ops.learn.cases*[*OF learn-ops-axioms*] *dest*: *assms*(*3*))[]

apply (*auto elim*!: *forget-ops.forget*$_{NOT}$*.cases*[*OF forget-ops-axioms*] *dest*!: *assms*(*4*))
done

lemma *rtranclp-cdcl*$_{NOT}$*-inv*:
cdcl$_{NOT}$** *S T* $\Longrightarrow$ *inv S* $\Longrightarrow$ *inv T*
apply (*induction rule*: *rtranclp-induct*)
apply *simp*
using *cdcl*$_{NOT}$*-inv* **unfolding** *conflict-driven-clause-learning-def*
conflict-driven-clause-learning-axioms-def **by** *blast*

lemma *learn-always-simple-clauses*:
assumes
learn: *learn S T* **and**
n-d: *no-dup* (*trail S*)
shows *set-mset* (*clauses T* $-$ *clauses S*)
$\subseteq$ *simple-clss* (*atms-of-msu* (*clauses S*) $\cup$ *atm-of* ' *lits-of* (*trail S*))
proof
fix *C* **assume** *C*: *C* $\in$ *set-mset* (*clauses T* $-$ *clauses S*)
have *distinct-mset C* $\neg$*tautology C* **using** *learn C n-d* **by** (*elim learn*$_{NOT}$*E*; *auto*)+
then have *C* $\in$ *simple-clss* (*atms-of C*)
using *distinct-mset-not-tautology-implies-in-simple-clss* **by** *blast*
moreover have *atms-of C* $\subseteq$ *atms-of-msu* (*clauses S*) $\cup$ *atm-of* ' *lits-of* (*trail S*)
using *learn C n-d* **by** (*elim learn*$_{NOT}$*E*) (*auto simp*: *atms-of-ms-def atms-of-def image-Un*
true-annots-CNot-all-atms-defined)
moreover have *finite* (*atms-of-msu* (*clauses S*) $\cup$ *atm-of* ' *lits-of* (*trail S*))
by *auto*
ultimately show *C* $\in$ *simple-clss* (*atms-of-msu* (*clauses S*) $\cup$ *atm-of* ' *lits-of* (*trail S*))
using *simple-clss-mono* **by** (*metis* (*no-types*) *insert-subset mk-disjoint-insert*)
qed

definition *conflicting-bj-clss S* $\equiv$
{*C*+{#*L*#}|*C L*. *C*+{#*L*#} $\in$# *clauses S* $\wedge$ *distinct-mset* (*C*+{#*L*#}) $\wedge$ $\neg$*tautology* (*C*+{#*L*#})
$\wedge$ ($\exists$ *F' K F*. *trail S* = *F'* @ *Marked K* () # *F* $\wedge$ *F* $\models$*as CNot C*)}

lemma *conflicting-bj-clss-remove-cls*$_{NOT}$[*simp*]:
conflicting-bj-clss (*remove-cls*$_{NOT}$ *C S*) = *conflicting-bj-clss S* $-$ {*C*}
unfolding *conflicting-bj-clss-def* **by** *fastforce*

lemma *conflicting-bj-clss-add-cls*$_{NOT}$*-state-eq*:
T $\sim$ *add-cls*$_{NOT}$ *C' S* $\Longrightarrow$ *no-dup* (*trail S*) $\Longrightarrow$ *conflicting-bj-clss T*
= *conflicting-bj-clss S*
$\cup$ (*if* $\exists$ *C L*. *C'* = *C* +{#*L*#} $\wedge$ *distinct-mset* (*C*+{#*L*#}) $\wedge$ $\neg$*tautology* (*C*+{#*L*#})
$\wedge$ ($\exists$ *F' K d F*. *trail S* = *F'* @ *Marked K* () # *F* $\wedge$ *F* $\models$*as CNot C*)
then {*C'*} *else* {})
unfolding *conflicting-bj-clss-def* **by** *auto metis*+

lemma *conflicting-bj-clss-add-cls*$_{NOT}$:
no-dup (*trail S*) $\Longrightarrow$
conflicting-bj-clss (*add-cls*$_{NOT}$ *C' S*)
= *conflicting-bj-clss S*
$\cup$ (*if* $\exists$ *C L*. *C'* = *C* +{#*L*#}$\wedge$ *distinct-mset* (*C*+{#*L*#}) $\wedge$ $\neg$*tautology* (*C*+{#*L*#})
$\wedge$ ($\exists$ *F' K d F*. *trail S* = *F'* @ *Marked K* () # *F* $\wedge$ *F* $\models$*as CNot C*)
then {*C'*} *else* {})
using *conflicting-bj-clss-add-cls*$_{NOT}$*-state-eq* **by** *auto*

lemma *conflicting-bj-clss-incl-clauses*:
  conflicting-bj-clss $S \subseteq$ *set-mset* (*clauses* S)
  unfolding *conflicting-bj-clss-def* **by** *auto*

lemma *finite-conflicting-bj-clss*[*simp*]:
  finite (*conflicting-bj-clss* S)
  using *conflicting-bj-clss-incl-clauses*[*of* S] *rev-finite-subset* **by** *blast*

lemma *learn-conflicting-increasing*:
  no-dup (*trail* S) $\Longrightarrow$ *learn* S T $\Longrightarrow$ *conflicting-bj-clss* $S \subseteq$ *conflicting-bj-clss* T
  apply (*elim* $learn_{NOT}E$)
  by (*subst* *conflicting-bj-clss-add-cls*$_{NOT}$*-state-eq*[*of* T]) *auto*

abbreviation *conflicting-bj-clss-yet* b $S \equiv$
  3 ^ b $-$ *card* (*conflicting-bj-clss* S)

abbreviation μ_L :: *nat* $\Rightarrow$ *'st* $\Rightarrow$ *nat* $\times$ *nat* **where**
  μ_L b $S \equiv$ (*conflicting-bj-clss-yet* b S, *card* (*set-mset* (*clauses* S)))

lemma *do-not-forget-before-backtrack-rule-clause-learned-clause-untouched*:
  assumes $forget_{NOT}$ S T
  shows *conflicting-bj-clss* S = *conflicting-bj-clss* T
  using *assms* **apply** *induction*
  unfolding *conflicting-bj-clss-def*
  by (*metis* (*no-types*, *lifting*) *Diff-insert-absorb Set.set-insert clauses-remove-cls*$_{NOT}$
    diff-union-cancelR insert-iff mem-set-mset-iff order-refl set-mset-minus-replicate-mset(*1*)
    state-eq$_{NOT}$*-clauses state-eq*$_{NOT}$*-trail trail-remove-cls*$_{NOT}$)

lemma *forget-*μ_L*-decrease*:
  assumes $forget_{NOT}$: $forget_{NOT}$ S T
  shows (μ_L b T, μ_L b S) $\in$ *less-than* <*∗lex∗*> *less-than*
proof −
  have *card* (*set-mset* (*clauses* T)) < *card* (*set-mset* (*clauses* S))
    using $forget_{NOT}$ **apply** *induction*
    by (*metis card-Diff1-less clauses-remove-cls*$_{NOT}$ *finite-set-mset mem-set-mset-iff order-refl*
      set-mset-minus-replicate-mset(*1*) *state-eq*$_{NOT}$*-clauses*)
  then show *?thesis*
    unfolding *do-not-forget-before-backtrack-rule-clause-learned-clause-untouched*[*OF* $forget_{NOT}$]
    by *auto*
qed

lemma *set-condition-or-split*:
  $\{a.\ (a = b \lor Q\ a) \land S\ a\}$ = (*if* S b *then* $\{b\}$ *else* $\{\}$) $\cup$ $\{a.\ Q\ a \land S\ a\}$
  by *auto*

lemma *set-insert-neq*:
  $A \neq$ *insert* a $A \longleftrightarrow a \notin A$
  by *auto*

lemma *learn-*μ_L*-decrease*:
  assumes *learnST*: *learn* S T **and** *n-d*: *no-dup* (*trail* S) **and**
  A: *atms-of-msu* (*clauses* S) $\cup$ *atm-of* ‘ *lits-of* (*trail* S) $\subseteq A$ **and**
  fin-A: *finite* A
  shows (μ_L (*card* A) T, μ_L (*card* A) S) $\in$ *less-than* <*∗lex∗*> *less-than*
proof −

have [*simp*]: (*atms-of-msu* (*clauses* T) ∪ *atm-of* ' *lits-of* (*trail* T))
  = (*atms-of-msu* (*clauses* S) ∪ *atm-of* ' *lits-of* (*trail* S))
  using *learnST n-d* **by** (*elim* $learn_{NOT}E$) *auto*

then have *card* (*atms-of-msu* (*clauses* T) ∪ *atm-of* ' *lits-of* (*trail* T))
  = *card* (*atms-of-msu* (*clauses* S) ∪ *atm-of* ' *lits-of* (*trail* S))
  by (*auto intro*!: *card-mono*)
then have *3*: (*3*::*nat*) ^ *card* (*atms-of-msu* (*clauses* T) ∪ *atm-of* ' *lits-of* (*trail* T))
  = *3* ^ *card* (*atms-of-msu* (*clauses* S) ∪ *atm-of* ' *lits-of* (*trail* S))
  by (*auto intro*: *power-mono*)
moreover have *conflicting-bj-clss* S ⊆ *conflicting-bj-clss* T
  using *learnST n-d* **by** (*simp add*: *learn-conflicting-increasing*)
moreover have *conflicting-bj-clss* S ≠ *conflicting-bj-clss* T
  using *learnST*
  proof (*elim* $learn_{NOT}E$, *goal-cases*)
    case (*1* C) **note** *clss-S* = *this*(*1*) **and** *atms-C* = *this*(*2*) **and** *inv* = *this*(*3*) **and** T = *this*(*4*)
    then obtain F K F' C' L **where**
      tr-S: *trail* S = F' @ *Marked* K () # F **and**
      C: C = C' + {#L#} **and**
      F: F ⊨*as CNot* C' **and**
      C-S:C' + {#L#} ∉# *clauses* S
      by *blast*
    moreover have *distinct-mset* C ¬ *tautology* C **using** *inv* **by** *blast*+
    ultimately have C' + {#L#} ∈ *conflicting-bj-clss* T
      using T *n-d* **unfolding** *conflicting-bj-clss-def* **by** *fastforce*
    moreover have C' + {#L#} ∉ *conflicting-bj-clss* S
      using *C-S* **unfolding** *conflicting-bj-clss-def* **by** *auto*
    ultimately show *?case* **by** *blast*
  qed
moreover have *fin-T*: *finite* (*conflicting-bj-clss* T)
  using *learnST* **by** *induction* (*auto simp add*: $conflicting\text{-}bj\text{-}clss\text{-}add\text{-}cls_{NOT}$)
ultimately have *card* (*conflicting-bj-clss* T) ≥ *card* (*conflicting-bj-clss* S)
  using *card-mono* **by** *blast*

moreover
  have *fin'*: *finite* (*atms-of-msu* (*clauses* T) ∪ *atm-of* ' *lits-of* (*trail* T))
    by *auto*
  have *1*:*atms-of-ms* (*conflicting-bj-clss* T) ⊆ *atms-of-msu* (*clauses* T)
    unfolding *conflicting-bj-clss-def atms-of-ms-def* **by** *auto*
  have *2*: ⋀x. x∈ *conflicting-bj-clss* T ⟹ ¬ *tautology* x ∧ *distinct-mset* x
    unfolding *conflicting-bj-clss-def* **by** *auto*
  have T: *conflicting-bj-clss* T
  ⊆ *simple-clss* (*atms-of-msu* (*clauses* T) ∪ *atm-of* ' *lits-of* (*trail* T))
    by *standard* (*meson 1 2 fin'* ‹*finite* (*conflicting-bj-clss* T)› *simple-clss-mono*
      distinct-mset-set-def simplified-in-simple-clss subsetCE sup.coboundedI1)
moreover
  then have #: *3* ^ *card* (*atms-of-msu* (*clauses* T) ∪ *atm-of* ' *lits-of* (*trail* T))
      ≥ *card* (*conflicting-bj-clss* T)
    by (*meson Nat.le-trans simple-clss-card simple-clss-finite card-mono fin'*)
  have *atms-of-msu* (*clauses* T) ∪ *atm-of* ' *lits-of* (*trail* T) ⊆ A
    using $learn_{NOT}E$[*OF learnST*] A **by** *simp*
  then have *3* ^ (*card* A) ≥ *card* (*conflicting-bj-clss* T)
    using # *fin-A* **by** (*meson simple-clss-card simple-clss-finite*
      simple-clss-mono calculation(*2*) *card-mono dual-order.trans*)
ultimately show *?thesis*

using *psubset-card-mono*[*OF fin-T*]
unfolding *less-than-iff lex-prod-def* **by** *clarify*
(*meson* ‹*conflicting-bj-clss S* ≠ *conflicting-bj-clss T*›
‹*conflicting-bj-clss S* ⊆ *conflicting-bj-clss T*›
diff-less-mono2 le-less-trans not-le psubsetI)
qed

We have to assume the following:

- *inv S*: the invariant holds in the inital state.
- *A* is a (finite *finite A*) superset of the literals in the trail *atm-of* ' *lits-of* (*trail S*) ⊆ *atms-of-ms A* and in the clauses *atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A*. This can the the set of all the literals in the starting set of clauses.
- *no-dup* (*trail S*): no duplicate in the trail. This is invariant along the path.

definition μ_{CDCL} **where**
μ_{CDCL} *A T* ≡ ((*2*+*card* (*atms-of-ms A*)) ^ (*1*+*card* (*atms-of-ms A*))
− μ_C (*1*+*card* (*atms-of-ms A*)) (*2*+*card* (*atms-of-ms A*)) (*trail-weight T*),
conflicting-bj-clss-yet (*card* (*atms-of-ms A*)) *T*, *card* (*set-mset* (*clauses T*)))
lemma $cdcl_{NOT}$*-decreasing-measure*:
assumes
$cdcl_{NOT}$ *S T* **and**
inv: *inv S* **and**
atm-clss: *atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A* **and**
atm-lits: *atm-of* ' *lits-of* (*trail S*) ⊆ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
fin-A: *finite A*
shows (μ_{CDCL} *A T*, μ_{CDCL} *A S*)
∈ *less-than* <∗*lex*∗> (*less-than* <∗*lex*∗> *less-than*)
using *assms*(*1*)
proof *induction*
case (*c-dpll-bj T*)
from *dpll-bj-trail-mes-decreasing-prop*[*OF this*(*1*) *inv atm-clss atm-lits n-d fin-A*]
show *?case* **unfolding** μ_{CDCL}*-def*
by (*meson in-lex-prod less-than-iff*)
next
case (*c-learn T*) **note** *learn* = *this*(*1*)
then have *S*: *trail S* = *trail T*
using *inv atm-clss atm-lits n-d fin-A*
by (*elim* $learn_{NOT}E$) *auto*
show *?case*
using *learn*-μ_L*-decrease*[*OF learn* -] *atm-clss atm-lits fin-A n-d* **unfolding** *S* μ_{CDCL}*-def* **by** *auto*
next
case (*c-*$forget_{NOT}$ *T*) **note** $forget_{NOT}$ = *this*(*1*)
have *trail S* = *trail T* **using** $forget_{NOT}$ **by** *induction auto*
then show *?case*
using *forget*-μ_L*-decrease*[*OF* $forget_{NOT}$] **unfolding** μ_{CDCL}*-def* **by** *auto*
qed

lemma *wf-*$cdcl_{NOT}$*-restricted-learning*:
assumes *finite A*
shows *wf* {(*T*, *S*).
(*atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A* ∧ *atm-of* ' *lits-of* (*trail S*) ⊆ *atms-of-ms A*

∧ *no-dup* (*trail S*)
∧ *inv S*)
∧ $cdcl_{NOT}$ *S T* }
by (*rule wf-wf-if-measure'*[*of less-than* <*lex*> (*less-than* <*lex*> *less-than*)])
(*auto intro*: $cdcl_{NOT}$*-decreasing-measure*[*OF - - - - - assms*])

definition $\mu_C{}'$ :: *'v literal multiset set* ⇒ *'st* ⇒ *nat* **where**
$\mu_C{}'$ *A T* ≡ μ_C (*1+card* (*atms-of-ms A*)) (*2+card* (*atms-of-ms A*)) (*trail-weight T*)

definition $\mu_{CDCL}{}'$ :: *'v literal multiset set* ⇒ *'st* ⇒ *nat* **where**
$\mu_{CDCL}{}'$ *A T* ≡
((*2+card* (*atms-of-ms A*)) ^ (*1+card* (*atms-of-ms A*)) − $\mu_C{}'$ *A T*) ∗ (*1+ 3^card* (*atms-of-ms A*)) ∗ *2*
+ *conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) *T* ∗ *2*
+ *card* (*set-mset* (*clauses T*))

lemma $cdcl_{NOT}$*-decreasing-measure'*:
assumes
$cdcl_{NOT}$ *S T* **and**
inv: *inv S* **and**
atms-clss: *atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A* **and**
atms-trail: *atm-of* ‘ *lits-of* (*trail S*) ⊆ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
fin-A: *finite A*
shows $\mu_{CDCL}{}'$ *A T* < $\mu_{CDCL}{}'$ *A S*
using *assms*(*1*)
proof (*induction rule*: $cdcl_{NOT}$*-learn-all-induct*)
case (*dpll-bj T*)
then have (*2+card* (*atms-of-ms A*)) ^ (*1+card* (*atms-of-ms A*)) − $\mu_C{}'$ *A T*
< (*2+card* (*atms-of-ms A*)) ^ (*1+card* (*atms-of-ms A*)) − $\mu_C{}'$ *A S*
using *dpll-bj-trail-mes-decreasing-prop fin-A inv n-d atms-clss atms-trail*
unfolding $\mu_C{}'$*-def* **by** *blast*
then have *XX*: ((*2+card* (*atms-of-ms A*)) ^ (*1+card* (*atms-of-ms A*)) − $\mu_C{}'$ *A T*) + *1*
≤ (*2+card* (*atms-of-ms A*)) ^ (*1+card* (*atms-of-ms A*)) − $\mu_C{}'$ *A S*
by *auto*
from *mult-le-mono1*[*OF this*, *of* (*1* + *3* ^ *card* (*atms-of-ms A*))]
have ((*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*)) − $\mu_C{}'$ *A T*) ∗
(*1* + *3* ^ *card* (*atms-of-ms A*)) + (*1* + *3* ^ *card* (*atms-of-ms A*))
≤ ((*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*)) − $\mu_C{}'$ *A S*)
∗ (*1* + *3* ^ *card* (*atms-of-ms A*))
unfolding *Nat.add-mult-distrib*
by *presburger*
moreover
have *cl-T-S*: *clauses T* = *clauses S*
using *dpll-bj.hyps inv dpll-bj-clauses* **by** *auto*
have *conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) *S* < *1+ 3* ^ *card* (*atms-of-ms A*)
by *simp*
ultimately have ((*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*)) − $\mu_C{}'$ *A T*)
∗ (*1* + *3* ^ *card* (*atms-of-ms A*)) + *conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) *T*
< ((*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*)) − $\mu_C{}'$ *A S*) ∗(*1* + *3* ^ *card* (*atms-of-ms A*))
by *linarith*
then have ((*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*)) − $\mu_C{}'$ *A T*)
∗ (*1* + *3* ^ *card* (*atms-of-ms A*))
+ *conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) *T*

$< ((2 + card\ (atms\text{-}of\text{-}ms\ A))\ \widehat{}\ (1 + card\ (atms\text{-}of\text{-}ms\ A)) - \mu_C{}'\ A\ S)$
$* (1 + 3\ \widehat{}\ card\ (atms\text{-}of\text{-}ms\ A))$
$+ conflicting\text{-}bj\text{-}clss\text{-}yet\ (card\ (atms\text{-}of\text{-}ms\ A))\ S$
by *linarith*
then have $((2 + card\ (atms\text{-}of\text{-}ms\ A))\ \widehat{}\ (1 + card\ (atms\text{-}of\text{-}ms\ A)) - \mu_C{}'\ A\ T)$
$* (1 + 3\ \widehat{}\ card\ (atms\text{-}of\text{-}ms\ A)) * 2$
$+ conflicting\text{-}bj\text{-}clss\text{-}yet\ (card\ (atms\text{-}of\text{-}ms\ A))\ T * 2$
$< ((2 + card\ (atms\text{-}of\text{-}ms\ A))\ \widehat{}\ (1 + card\ (atms\text{-}of\text{-}ms\ A)) - \mu_C{}'\ A\ S)$
$* (1 + 3\ \widehat{}\ card\ (atms\text{-}of\text{-}ms\ A)) * 2$
$+ conflicting\text{-}bj\text{-}clss\text{-}yet\ (card\ (atms\text{-}of\text{-}ms\ A))\ S * 2$
by *linarith*
then show *?case* **unfolding** $\mu_{CDCL}{}'$*-def cl-T-S* **by** *presburger*
next
case (*learn C F′ K F C′ L T*) **note** *clss-S-C* = *this*(*1*) **and** *atms-C* = *this*(*2*) **and** *dist* = *this*(*3*)
and *tauto* = *this*(*4*) **and** *learn-restr* = *this*(*5*) **and** *tr-S* = *this*(*6*) **and** *C′* = *this*(*7*) **and**
F-C = *this*(*8*) **and** *C-new* = *this*(*9*) **and** *T* =*this*(*10*)
have *insert C* (*conflicting-bj-clss S*) ⊆ *simple-clss* (*atms-of-ms A*)
proof −
have *C* ∈ *simple-clss* (*atms-of-ms A*)
by (*metis* (*no-types*, *hide-lams*) *Un-subset-iff atms-of-ms-finite simple-clss-mono*
contra-subsetD dist distinct-mset-not-tautology-implies-in-simple-clss
dual-order.trans fin-A atms-C atms-clss atms-trail tauto)
moreover have *conflicting-bj-clss S* ⊆ *simple-clss* (*atms-of-ms A*)
unfolding *conflicting-bj-clss-def*
proof
fix *x* :: *′v literal multiset*
assume *x* ∈ {*C* + {#*L*#} |*C L. C* + {#*L*#} ∈# *clauses S*
∧ *distinct-mset* (*C* + {#*L*#}) ∧ ¬ *tautology* (*C* + {#*L*#})
∧ (∃ *F′ K F. trail S* = *F′* @ *Marked K* () # *F* ∧ *F* ⊨*as CNot C*)}
then have ∃ *m l. x* = *m* + {#*l*#} ∧ *m* + {#*l*#} ∈# *clauses S*
∧ *distinct-mset* (*m* + {#*l*#}) ∧ ¬ *tautology* (*m* + {#*l*#})
∧ (∃ *ms l msa. trail S* = *ms* @ *Marked l* () # *msa* ∧ *msa* ⊨*as CNot m*)
by *blast*
then show *x* ∈ *simple-clss* (*atms-of-ms A*)
by (*meson atms-clss atms-of-atms-of-ms-mono atms-of-ms-finite simple-clss-mono*
distinct-mset-not-tautology-implies-in-simple-clss fin-A finite-subset
mem-set-mset-iff set-rev-mp)
qed
ultimately show *?thesis*
by *auto*
qed
then have $card\ (insert\ C\ (conflicting\text{-}bj\text{-}clss\ S)) \le 3\ \widehat{}\ (card\ (atms\text{-}of\text{-}ms\ A))$
by (*meson Nat.le-trans atms-of-ms-finite simple-clss-card simple-clss-finite*
card-mono fin-A)
moreover have [*simp*]: *card* (*insert C* (*conflicting-bj-clss S*))
= *Suc* (*card* ((*conflicting-bj-clss S*)))
by (*metis* (*no-types*) *C′ C-new card-insert-if conflicting-bj-clss-incl-clauses contra-subsetD*
finite-conflicting-bj-clss mem-set-mset-iff)
moreover have [*simp*]: *conflicting-bj-clss* (*add-cls*$_{NOT}$ *C S*) = *conflicting-bj-clss S* ∪ {*C*}
using *dist tauto F-C n-d* **by** (*subst conflicting-bj-clss-add-cls*$_{NOT}$)
(*force simp add*: *ac-simps C′ tr-S*)+
ultimately have [*simp*]: *conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) *S*
= *Suc* (*conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) (*add-cls*$_{NOT}$ *C S*))
by *simp*
have *1*: *clauses T* = *clauses* (*add-cls*$_{NOT}$ *C S*) **using** *T* **by** *auto*

have *2*: *conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) *T*
= *conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) (*add-cls*$_{NOT}$ *C S*)
using *T* **unfolding** *conflicting-bj-clss-def* **by** *auto*
have *3*: $\mu_C{}'$ *A T* = $\mu_C{}'$ *A* (*add-cls*$_{NOT}$ *C S*)
using *T* **unfolding** $\mu_C{}'$*-def* **by** *auto*
have ((*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*)) − $\mu_C{}'$ *A* (*add-cls*$_{NOT}$ *C S*))
∗ (*1* + *3* ^ *card* (*atms-of-ms A*)) ∗ *2*
= ((*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*)) − $\mu_C{}'$ *A S*)
∗ (*1* + *3* ^ *card* (*atms-of-ms A*)) ∗ *2*
using *n-d* **unfolding** $\mu_C{}'$*-def* **by** *auto*
moreover
have *conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) (*add-cls*$_{NOT}$ *C S*)
∗ *2*
+ *card* (*set-mset* (*clauses* (*add-cls*$_{NOT}$ *C S*)))
< *conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) *S* ∗ *2*
+ *card* (*set-mset* (*clauses S*))
by (*simp add*: *C′ C-new n-d*)
ultimately show *?case* **unfolding** $\mu_{CDCL}{}'$*-def 1 2 3* **by** *presburger*
next
case (*forget*$_{NOT}$ *C T*) **note** *T* = *this*(*4*)
have [*simp*]: $\mu_C{}'$ *A* (*remove-cls*$_{NOT}$ *C S*) = $\mu_C{}'$ *A S*
unfolding $\mu_C{}'$*-def* **by** *auto*
have *forget*$_{NOT}$ *S T*
apply (*rule forget*$_{NOT}$*.intros*) **using** *forget*$_{NOT}$ **by** *auto*
then have *conflicting-bj-clss T* = *conflicting-bj-clss S*
using *do-not-forget-before-backtrack-rule-clause-learned-clause-untouched* **by** *blast*
moreover have *card* (*set-mset* (*clauses T*)) < *card* (*set-mset* (*clauses S*))
by (*metis T card-Diff1-less clauses-remove-cls*$_{NOT}$ *finite-set-mset forget*$_{NOT}$*.hyps*(*2*)
mem-set-mset-iff order-refl set-mset-minus-replicate-mset(*1*) *state-eq*$_{NOT}$*-clauses*)
ultimately show *?case* **unfolding** $\mu_{CDCL}{}'$*-def*
by (*metis* (*no-types*) *T* ‹$\mu_C{}'$ *A* (*remove-cls*$_{NOT}$ *C S*) = $\mu_C{}'$ *A S*› *add-le-cancel-left*
$\mu_C{}'$*-def not-le state-eq*$_{NOT}$*-trail*)
qed

lemma *cdcl*$_{NOT}$*-clauses-bound*:
assumes
cdcl$_{NOT}$ *S T* **and**
inv S **and**
atms-of-msu (*clauses S*) ⊆ *A* **and**
atm-of ‘(*lits-of* (*trail S*)) ⊆ *A* **and**
n-d: *no-dup* (*trail S*) **and**
fin-A[*simp*]: *finite A*
shows *set-mset* (*clauses T*) ⊆ *set-mset* (*clauses S*) ∪ *simple-clss A*
using *assms*
proof (*induction rule*: *cdcl*$_{NOT}$*-learn-all-induct*)
case *dpll-bj*
then show *?case* **using** *dpll-bj-clauses* **by** *simp*
next
case *forget*$_{NOT}$
then show *?case* **using** *clauses-remove-cls*$_{NOT}$ **unfolding** *state-eq*$_{NOT}$*-def* **by** *auto*
next
case (*learn C F K d F′ C′ L*) **note** *atms-C* = *this*(*2*) **and** *dist* = *this*(*3*) **and** *tauto* = *this*(*4*) **and**
T = *this*(*10*) **and** *atms-clss-S* = *this*(*12*) **and** *atms-trail-S* = *this*(*13*)
have *atms-of C* ⊆ *A*
using *atms-C atms-clss-S atms-trail-S* **by** *auto*

then have *simple-clss* (*atms-of* C) $\subseteq$ *simple-clss* A
  by (*simp add*: *simple-clss-mono*)
then have $C \in$ *simple-clss* A
  using *finite dist tauto*
  by (*auto dest*: *distinct-mset-not-tautology-implies-in-simple-clss*)
then show *?case* **using** T *n-d* **by** *auto*
qed

lemma *rtranclp-cdcl*$_{NOT}$*-clauses-bound*:
  assumes
    cdcl$_{NOT}$** S T **and**
    inv S **and**
    atms-of-msu (*clauses* S) $\subseteq A$ **and**
    atm-of ‘(*lits-of* (*trail* S)) $\subseteq A$ **and**
    n-d: *no-dup* (*trail* S) **and**
    finite: *finite* A
  shows *set-mset* (*clauses* T) $\subseteq$ *set-mset* (*clauses* S) $\cup$ *simple-clss* A
  using *assms*(*1*−*5*)
proof *induction*
  case *base*
  then show *?case* **by** *simp*
next
  case (*step* T U) **note** *st* = *this*(*1*) **and** *cdcl*$_{NOT}$ = *this*(*2*) **and** *IH* = *this*(*3*)[*OF this*(*4*−*7*)] **and**
    inv = *this*(*4*) **and** *atms-clss-S* = *this*(*5*) **and** *atms-trail-S* = *this*(*6*) **and** *finite-cls-S* = *this*(*7*)
  have *inv* T
    using *rtranclp-cdcl*$_{NOT}$*-inv st inv* **by** *blast*
  moreover have *atms-of-msu* (*clauses* T) $\subseteq A$ **and** *atm-of* ‘ *lits-of* (*trail* T) $\subseteq A$
    using *rtranclp-cdcl*$_{NOT}$*-trail-clauses-bound*[*OF st*] *inv atms-clss-S atms-trail-S n-d* **by** *blast*+
  moreover have *no-dup* (*trail* T)
   using *rtranclp-cdcl*$_{NOT}$*-no-dup*[*OF st* ‹*inv* S› *n-d*] **by** *simp*
  ultimately have *set-mset* (*clauses* U) $\subseteq$ *set-mset* (*clauses* T) $\cup$ *simple-clss* A
    using *cdcl*$_{NOT}$ *finite n-d* **by** (*auto simp*: *cdcl*$_{NOT}$*-clauses-bound*)
  then show *?case* **using** *IH* **by** *auto*
qed

lemma *rtranclp-cdcl*$_{NOT}$*-card-clauses-bound*:
  assumes
    cdcl$_{NOT}$** S T **and**
    inv S **and**
    atms-of-msu (*clauses* S) $\subseteq A$ **and**
    atm-of ‘(*lits-of* (*trail* S)) $\subseteq A$ **and**
    n-d: *no-dup* (*trail* S) **and**
    finite: *finite* A
  shows *card* (*set-mset* (*clauses* T)) $\leq$ *card* (*set-mset* (*clauses* S)) + *3* ^ (*card* A)
  using *rtranclp-cdcl*$_{NOT}$*-clauses-bound*[*OF assms*] *finite* **by** (*meson Nat.le-trans*
    simple-clss-card simple-clss-finite card-Un-le card-mono finite-UnI
    finite-set-mset nat-add-left-cancel-le)

lemma *rtranclp-cdcl*$_{NOT}$*-card-clauses-bound*$'$:
  assumes
    cdcl$_{NOT}$** S T **and**
    inv S **and**
    atms-of-msu (*clauses* S) $\subseteq A$ **and**
    atm-of ‘(*lits-of* (*trail* S)) $\subseteq A$ **and**

n-d: *no-dup* (*trail S*) **and**
finite: *finite A*
shows *card* {*C*|*C*. *C* ∈# *clauses T* ∧ (*tautology C* ∨ ¬*distinct-mset C*)}
≤ *card* {*C*|*C*. *C*∈# *clauses S* ∧ (*tautology C* ∨ ¬*distinct-mset C*)} + *3* ^ (*card A*)
(**is** *card ?T* ≤ *card ?S* + -)
using *rtranclp-cdcl*$_{NOT}$*-clauses-bound*[*OF assms*] *finite*
proof −
have *?T* ⊆ *?S* ∪ *simple-clss A*
using *rtranclp-cdcl*$_{NOT}$*-clauses-bound*[*OF assms*] **by** *force*
then have *card ?T* ≤ *card* (*?S* ∪ *simple-clss A*)
using *finite* **by** (*simp add*: *assms*(*5*) *simple-clss-finite card-mono*)
then show *?thesis*
by (*meson le-trans simple-clss-card card-Un-le local.finite nat-add-left-cancel-le*)
qed

lemma *rtranclp-cdcl*$_{NOT}$*-card-simple-clauses-bound*:
assumes
cdcl$_{NOT}$** *S T* **and**
inv S **and**
atms-of-msu (*clauses S*) ⊆ *A* **and**
atm-of ‘(*lits-of* (*trail S*)) ⊆ *A* **and**
n-d: *no-dup* (*trail S*) **and**
finite: *finite A*
shows *card* (*set-mset* (*clauses T*))
≤ *card* {*C*. *C* ∈# *clauses S* ∧ (*tautology C* ∨ ¬*distinct-mset C*)} + *3* ^ (*card A*)
(**is** *card ?T* ≤ *card ?S* + -)
using *rtranclp-cdcl*$_{NOT}$*-clauses-bound*[*OF assms*] *finite*
proof −
have $\bigwedge$*x*. *x* ∈# *clauses T* ⟹¬ *tautology x* ⟹ *distinct-mset x* ⟹ *x* ∈ *simple-clss A*
using *rtranclp-cdcl*$_{NOT}$*-clauses-bound*[*OF assms*] **by** (*metis* (*no-types*, *hide-lams*) *Un-iff assms*(*3*)
atms-of-atms-of-ms-mono simple-clss-mono contra-subsetD
distinct-mset-not-tautology-implies-in-simple-clss local.finite mem-set-mset-iff
subset-trans)
then have *set-mset* (*clauses T*) ⊆ *?S* ∪ *simple-clss A*
using *rtranclp-cdcl*$_{NOT}$*-clauses-bound*[*OF assms*] **by** *auto*
then have *card*(*set-mset* (*clauses T*)) ≤ *card* (*?S* ∪ *simple-clss A*)
using *finite* **by** (*simp add*: *assms*(*5*) *simple-clss-finite card-mono*)
then show *?thesis*
by (*meson le-trans simple-clss-card card-Un-le local.finite nat-add-left-cancel-le*)
qed

definition μ_{CDCL}*'-bound* :: *'v literal multiset set* ⇒ *'st* ⇒ *nat* **where**
μ_{CDCL}*'-bound A S* =
((*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*))) ∗ (*1* + *3* ^ *card* (*atms-of-ms A*)) ∗ *2*
+ *2*∗*3* ^ (*card* (*atms-of-ms A*))
+ *card* {*C*. *C* ∈# *clauses S* ∧ (*tautology C* ∨ ¬*distinct-mset C*)} + *3* ^ (*card* (*atms-of-ms A*))

lemma μ_{CDCL}*'-bound-reduce-trail-to*$_{NOT}$[*simp*]:
μ_{CDCL}*'-bound A* (*reduce-trail-to*$_{NOT}$ *M S*) = μ_{CDCL}*'-bound A S*
unfolding μ_{CDCL}*'-bound-def* **by** *auto*

lemma *rtranclp-cdcl*$_{NOT}$*-*μ_{CDCL}*'-bound-reduce-trail-to*$_{NOT}$:
assumes
cdcl$_{NOT}$** *S T* **and**
inv S **and**

atms-of-msu (*clauses S*) ⊆ *atms-of-ms A* **and**
atm-of ‘(*lits-of* (*trail S*)) ⊆ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
finite: *finite* (*atms-of-ms A*) **and**
U: $U \sim$ *reduce-trail-to*$_{NOT}$ *M T*
shows $\mu_{CDCL}{}'$ *A U* ≤ $\mu_{CDCL}{}'$*-bound A S*
proof −
have ((*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*)) − $\mu_C{}'$ *A U*)
≤ (*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*))
by *auto*
then have ((*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*)) − $\mu_C{}'$ *A U*)
∗ (*1* + *3* ^ *card* (*atms-of-ms A*)) ∗ *2*
≤ (*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*)) ∗ (*1* + *3* ^ *card* (*atms-of-ms A*)) ∗ *2*
using *mult-le-mono1* **by** *blast*
moreover
have *conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) *T* ∗ *2* ≤ *2* ∗ *3* ^ *card* (*atms-of-ms A*)
by *linarith*
moreover have *card* (*set-mset* (*clauses U*))
≤ *card* {*C*. *C* ∈# *clauses S* ∧ (*tautology C* ∨ ¬*distinct-mset C*)} + *3* ^ *card* (*atms-of-ms A*)
using *rtranclp-cdcl*$_{NOT}$*-card-simple-clauses-bound*[*OF assms*(*1*−*6*)] *U* **by** *auto*
ultimately show *?thesis*
unfolding $\mu_{CDCL}{}'$*-def* $\mu_{CDCL}{}'$*-bound-def* **by** *linarith*
qed

lemma *rtranclp-cdcl*$_{NOT}$-$\mu_{CDCL}{}'$*-bound*:
assumes
cdcl$_{NOT}{}^{**}$ *S T* **and**
inv S **and**
atms-of-msu (*clauses S*) ⊆ *atms-of-ms A* **and**
atm-of ‘(*lits-of* (*trail S*)) ⊆ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
finite: *finite* (*atms-of-ms A*)
shows $\mu_{CDCL}{}'$ *A T* ≤ $\mu_{CDCL}{}'$*-bound A S*
proof −
have $\mu_{CDCL}{}'$ *A* (*reduce-trail-to*$_{NOT}$ (*trail T*) *T*) = $\mu_{CDCL}{}'$ *A T*
unfolding $\mu_{CDCL}{}'$*-def* $\mu_C{}'$*-def conflicting-bj-clss-def* **by** *auto*
then show *?thesis* **using** *rtranclp-cdcl*$_{NOT}$-$\mu_{CDCL}{}'$*-bound-reduce-trail-to*$_{NOT}$[*OF assms*, *of - trail T*]
state-eq$_{NOT}$*-ref* **by** *fastforce*
qed

lemma *rtranclp*-$\mu_{CDCL}{}'$*-bound-decreasing*:
assumes
cdcl$_{NOT}{}^{**}$ *S T* **and**
inv S **and**
atms-of-msu (*clauses S*) ⊆ *atms-of-ms A* **and**
atm-of ‘(*lits-of* (*trail S*)) ⊆ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
finite[*simp*]: *finite* (*atms-of-ms A*)
shows $\mu_{CDCL}{}'$*-bound A T* ≤ $\mu_{CDCL}{}'$*-bound A S*
proof −
have {*C*. *C* ∈# *clauses T* ∧ (*tautology C* ∨ ¬ *distinct-mset C*)}
⊆ {*C*. *C* ∈# *clauses S* ∧ (*tautology C* ∨ ¬ *distinct-mset C*)} (**is** *?T* ⊆ *?S*)
proof (*rule Set.subsetI*)
fix *C* **assume** *C* ∈ *?T*
then have *C-T*: *C* ∈# *clauses T* **and** *t-d*: *tautology C* ∨ ¬ *distinct-mset C*

by *auto*
then have $C \notin$ *simple-clss* (*atms-of-ms* A)
by (*auto dest*: *simple-clssE*)
then show $C \in$ *?S*
using *C-T rtranclp-cdcl*$_{NOT}$*-clauses-bound*[*OF assms*] *t-d* **by** *force*
qed
then have *card* $\{C.\ C \in\#$ *clauses* $T \wedge$ (*tautology* $C \vee \neg$ *distinct-mset* C)$\} \leq$
card $\{C.\ C \in\#$ *clauses* $S \wedge$ (*tautology* $C \vee \neg$ *distinct-mset* C)$\}$
by (*simp add*: *card-mono*)
then show *?thesis*
unfolding μ_{CDCL}*'-bound-def* **by** *auto*
qed

end — end of *conflict-driven-clause-learning-learning-before-backjump-only-distinct-learnt*

2.7 CDCL with restarts

2.7.1 Definition

locale *restart-ops* =
fixes
cdcl$_{NOT}$ :: *'st* ⇒ *'st* ⇒ *bool* **and**
restart :: *'st* ⇒ *'st* ⇒ *bool*
begin
inductive *cdcl*$_{NOT}$*-raw-restart* :: *'st* ⇒ *'st* ⇒ *bool* **where**
cdcl$_{NOT}$ S T ⟹ *cdcl*$_{NOT}$*-raw-restart* S T |
restart S T ⟹ *cdcl*$_{NOT}$*-raw-restart* S T

end

locale *conflict-driven-clause-learning-with-restarts* =
conflict-driven-clause-learning trail clauses prepend-trail tl-trail add-cls$_{NOT}$ *remove-cls*$_{NOT}$
propagate-conds inv backjump-conds learn-cond forget-cond
for
trail :: *'st* ⇒ (*'v*, *unit*, *unit*) *ann-literals* **and**
clauses :: *'st* ⇒ *'v clauses* **and**
prepend-trail :: (*'v*, *unit*, *unit*) *ann-literal* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒ *'st* **and**
add-cls$_{NOT}$ *remove-cls*$_{NOT}$:: *'v clause* ⇒ *'st* ⇒ *'st* **and**
propagate-conds :: (*'v*, *unit*, *unit*) *ann-literal* ⇒ *'st* ⇒ *bool* **and**
inv :: *'st* ⇒ *bool* **and**
backjump-conds :: *'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool* **and**
learn-cond forget-cond :: *'v clause* ⇒ *'st* ⇒ *bool*
begin

lemma *cdcl*$_{NOT}$*-iff-cdcl*$_{NOT}$*-raw-restart-no-restarts*:
cdcl$_{NOT}$ S T ⟷ *restart-ops.cdcl*$_{NOT}$*-raw-restart cdcl*$_{NOT}$ (λ- -. *False*) S T
(**is** *?C* S T ⟷ *?R* S T)
proof
fix S T
assume *?C* S T
then show *?R* S T **by** (*simp add*: *restart-ops.cdcl*$_{NOT}$*-raw-restart.intros*(*1*))
next
fix S T
assume *?R* S T
then show *?C* S T

apply (*cases rule*: *restart-ops.cdcl*$_{NOT}$*-raw-restart.cases*)
using ‹*?R S T*› **by** *fast+*
qed

lemma *cdcl*$_{NOT}$*-cdcl*$_{NOT}$*-raw-restart*:
cdcl$_{NOT}$ *S T* $\Longrightarrow$ *restart-ops.cdcl*$_{NOT}$*-raw-restart cdcl*$_{NOT}$ *restart S T*
by (*simp add*: *restart-ops.cdcl*$_{NOT}$*-raw-restart.intros*(*1*))
end

2.7.2 Increasing restarts

To add restarts we needs some assumptions on the predicate (called *cdcl*$_{NOT}$ here):

- a function *f* that is strictly monotonic. The first step is actually only used as a restart to clean the state (e.g. to ensure that the trail is empty). Then we assume that $(1::'a) \leq f\ n$ for $(1::'a) \leq n$: it means that between two consecutive restarts, at least one step will be done. This is necessary to avoid sequence. like: full – restart – full – ...
- a measure μ: it should decrease under the assumptions *bound-inv*, whenever a *cdcl*$_{NOT}$ or a *restart* is done. A parameter is given to μ: for conflict- driven clause learning, it is an upper-bound of the clauses. We are assuming that such a bound can be found after a restart whenever the invariant holds.
- we also assume that the measure decrease after any *cdcl*$_{NOT}$ step.
- an invariant on the states *cdcl*$_{NOT}$*-inv* that also holds after restarts.
- it is *not required* that the measure decrease with respect to restarts, but the measure has to be bound by some function *μ-bound* taking the same parameter as μ and the initial state of the considered *cdcl*$_{NOT}$ chain.

locale *cdcl*$_{NOT}$*-increasing-restarts-ops* =
restart-ops cdcl$_{NOT}$ *restart* **for**
restart :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **and**
cdcl$_{NOT}$ :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* +
fixes
f :: *nat* $\Rightarrow$ *nat* **and**
bound-inv :: *'bound* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **and**
μ :: *'bound* $\Rightarrow$ *'st* $\Rightarrow$ *nat* **and**
cdcl$_{NOT}$*-inv* :: *'st* $\Rightarrow$ *bool* **and**
μ-bound :: *'bound* $\Rightarrow$ *'st* $\Rightarrow$ *nat*
assumes
f: *unbounded f* **and**
f-ge-1: $\bigwedge n.\ n \geq 1 \Longrightarrow f\ n \neq 0$ **and**
bound-inv: $\bigwedge A\ S\ T.$ *cdcl*$_{NOT}$*-inv S* $\Longrightarrow$ *bound-inv A S* $\Longrightarrow$ *cdcl*$_{NOT}$ *S T* $\Longrightarrow$ *bound-inv A T* **and**
cdcl$_{NOT}$*-measure*: $\bigwedge A\ S\ T.$ *cdcl*$_{NOT}$*-inv S* $\Longrightarrow$ *bound-inv A S* $\Longrightarrow$ *cdcl*$_{NOT}$ *S T* $\Longrightarrow \mu\ A\ T < \mu\ A\ S$ **and**
measure-bound2: $\bigwedge A\ T\ U.$ *cdcl*$_{NOT}$*-inv T* $\Longrightarrow$ *bound-inv A T* $\Longrightarrow$ *cdcl*$_{NOT}{}^{**}$ *T U*
$\Longrightarrow \mu\ A\ U \leq$ *μ-bound A T* **and**
measure-bound4: $\bigwedge A\ T\ U.$ *cdcl*$_{NOT}$*-inv T* $\Longrightarrow$ *bound-inv A T* $\Longrightarrow$ *cdcl*$_{NOT}{}^{**}$ *T U*
$\Longrightarrow$ *μ-bound A U* $\leq$ *μ-bound A T* **and**
cdcl$_{NOT}$*-restart-inv*: $\bigwedge A\ U\ V.$ *cdcl*$_{NOT}$*-inv U* $\Longrightarrow$ *restart U V* $\Longrightarrow$ *bound-inv A U* $\Longrightarrow$ *bound-inv A V*
and

exists-bound: $\bigwedge R\ S.\ cdcl_{NOT}\text{-}inv\ R \Longrightarrow restart\ R\ S \Longrightarrow \exists A.\ bound\text{-}inv\ A\ S$ **and**
$cdcl_{NOT}$-inv: $\bigwedge S\ T.\ cdcl_{NOT}\text{-}inv\ S \Longrightarrow cdcl_{NOT}\ S\ T \Longrightarrow cdcl_{NOT}\text{-}inv\ T$ **and**
$cdcl_{NOT}$-inv-restart: $\bigwedge S\ T.\ cdcl_{NOT}\text{-}inv\ S \Longrightarrow restart\ S\ T \Longrightarrow cdcl_{NOT}\text{-}inv\ T$
begin

lemma *$cdcl_{NOT}$-$cdcl_{NOT}$-inv*:
assumes
$(cdcl_{NOT}\widehat{\ }\widehat{\ }n)\ S\ T$ **and**
$cdcl_{NOT}$-inv S
shows *$cdcl_{NOT}$-inv T*
using *assms* **by** (*induction n arbitrary*: *T*) (*auto intro*:*bound-inv $cdcl_{NOT}$-inv*)

lemma *$cdcl_{NOT}$-bound-inv*:
assumes
$(cdcl_{NOT}\widehat{\ }\widehat{\ }n)\ S\ T$ **and**
$cdcl_{NOT}$-inv S
bound-inv A S
shows *bound-inv A T*
using *assms* **by** (*induction n arbitrary*: *T*) (*auto intro*:*bound-inv $cdcl_{NOT}$-$cdcl_{NOT}$-inv*)

lemma *rtranclp-$cdcl_{NOT}$-$cdcl_{NOT}$-inv*:
assumes
$cdcl_{NOT}^{**}\ S\ T$ **and**
$cdcl_{NOT}$-inv S
shows *$cdcl_{NOT}$-inv T*
using *assms* **by** *induction* (*auto intro*: *$cdcl_{NOT}$-inv*)

lemma *rtranclp-$cdcl_{NOT}$-bound-inv*:
assumes
$cdcl_{NOT}^{**}\ S\ T$ **and**
bound-inv A S **and**
$cdcl_{NOT}$-inv S
shows *bound-inv A T*
using *assms* **by** *induction* (*auto intro*:*bound-inv rtranclp-$cdcl_{NOT}$-$cdcl_{NOT}$-inv*)

lemma *$cdcl_{NOT}$-comp-n-le*:
assumes
$(cdcl_{NOT}\widehat{\ }\widehat{\ }(Suc\ n))\ S\ T$ **and**
bound-inv A S
$cdcl_{NOT}$-inv S
shows $\mu\ A\ T < \mu\ A\ S - n$
using *assms*
proof (*induction n arbitrary*: *T*)
case *0*
then show *?case* **using** *$cdcl_{NOT}$-measure* **by** *auto*
next
case (*Suc n*) **note** *IH* =*this*(*1*)[*OF - this*(*3*) *this*(*4*)] **and** *S-T* =*this*(*2*) **and** *b-inv* = *this*(*3*) **and** *c-inv* = *this*(*4*)
obtain *U* :: *'st* **where** *S-U*: $(cdcl_{NOT}\widehat{\ }\widehat{\ }(Suc\ n))\ S\ U$ **and** *U-T*: $cdcl_{NOT}\ U\ T$ **using** *S-T* **by** *auto*
then have $\mu\ A\ U < \mu\ A\ S - n$ **using** *IH*[*of U*] **by** *simp*
moreover
have *bound-inv A U*
using *S-U b-inv $cdcl_{NOT}$-bound-inv c-inv* **by** *blast*
then have $\mu\ A\ T < \mu\ A\ U$ **using** *$cdcl_{NOT}$-measure*[*OF - - U-T*] *S-U c-inv $cdcl_{NOT}$-$cdcl_{NOT}$-inv* **by** *auto*

ultimately show *?case* **by** *linarith*
qed

lemma *wf-cdcl*$_{NOT}$:
wf {(*T*, *S*). *cdcl*$_{NOT}$ *S T* ∧ *cdcl*$_{NOT}$*-inv S* ∧ *bound-inv A S*} (**is** *wf ?A*)
apply (*rule wfP-if-measure2*[*of - - μ A*])
using *cdcl*$_{NOT}$*-comp-n-le*[*of 0 - - A*] **by** *auto*

lemma *rtranclp-cdcl*$_{NOT}$*-measure*:
assumes
cdcl$_{NOT}$** *S T* **and**
bound-inv A S **and**
cdcl$_{NOT}$*-inv S*
shows $\mu\ A\ T \leq \mu\ A\ S$
using *assms*
proof (*induction rule*: *rtranclp-induct*)
case *base*
then show *?case* **by** *auto*
next
case (*step T U*) **note** *IH* =*this*(*3*)[*OF this*(*4*) *this*(*5*)] **and** *st* =*this*(*1*) **and** *cdcl*$_{NOT}$= *this*(*2*) **and**
b-inv = *this*(*4*) **and** *c-inv* = *this*(*5*)
have *bound-inv A T*
by (*meson cdcl*$_{NOT}$*-bound-inv rtranclp-imp-relpowp st step.prems*)
moreover have *cdcl*$_{NOT}$*-inv T*
using *c-inv rtranclp-cdcl*$_{NOT}$*-cdcl*$_{NOT}$*-inv st* **by** *blast*
ultimately have $\mu\ A\ U < \mu\ A\ T$ **using** *cdcl*$_{NOT}$*-measure*[*OF - - cdcl*$_{NOT}$] **by** *auto*
then show *?case* **using** *IH* **by** *linarith*
qed

lemma *cdcl*$_{NOT}$*-comp-bounded*:
assumes
bound-inv A S **and** *cdcl*$_{NOT}$*-inv S* **and** $m \geq 1+\mu\ A\ S$
shows ¬(*cdcl*$_{NOT}$ ^^ *m*) *S T*
using *assms cdcl*$_{NOT}$*-comp-n-le*[*of m−1 S T A*] **by** *fastforce*

- $f\ n < m$ ensures that at least one step has been done.

inductive *cdcl*$_{NOT}$*-restart* **where**
restart-step: (*cdcl*$_{NOT}$^^*m*) *S T* ⟹ $m \geq f\ n$ ⟹ *restart T U*
⟹ *cdcl*$_{NOT}$*-restart* (*S*, *n*) (*U*, *Suc n*) |
restart-full: *full1 cdcl*$_{NOT}$ *S T* ⟹ *cdcl*$_{NOT}$*-restart* (*S*, *n*) (*T*, *Suc n*)

lemmas *cdcl*$_{NOT}$*-with-restart-induct* = *cdcl*$_{NOT}$*-restart.induct*[*split-format*(*complete*),
OF cdcl$_{NOT}$*-increasing-restarts-ops-axioms*]

lemma *cdcl*$_{NOT}$*-restart-cdcl*$_{NOT}$*-raw-restart*:
cdcl$_{NOT}$*-restart S T* ⟹ *cdcl*$_{NOT}$*-raw-restart*** (*fst S*) (*fst T*)
proof (*induction rule*: *cdcl*$_{NOT}$*-restart.induct*)
case (*restart-step m S T n U*)
then have *cdcl*$_{NOT}$** *S T* **by** (*meson relpowp-imp-rtranclp*)
then have *cdcl*$_{NOT}$*-raw-restart*** *S T* **using** *cdcl*$_{NOT}$*-raw-restart.intros*(*1*)
rtranclp-mono[*of cdcl*$_{NOT}$ *cdcl*$_{NOT}$*-raw-restart*] **by** *blast*
moreover have *cdcl*$_{NOT}$*-raw-restart T U*
using ‹*restart T U*› *cdcl*$_{NOT}$*-raw-restart.intros*(*2*) **by** *blast*
ultimately show *?case* **by** *auto*

next
  case (*restart-full S T*)
  then have $cdcl_{NOT}{}^{**}$ *S T* **unfolding** *full1-def* **by** *auto*
  then show *?case* **using** $cdcl_{NOT}$*-raw-restart.intros*(*1*)
    rtranclp-mono[*of* $cdcl_{NOT}$ $cdcl_{NOT}$*-raw-restart*] **by** *auto*
qed

lemma $cdcl_{NOT}$*-with-restart-bound-inv*:
  assumes
    $cdcl_{NOT}$*-restart S T* **and**
    bound-inv A (*fst S*) **and**
    $cdcl_{NOT}$*-inv* (*fst S*)
  shows *bound-inv A* (*fst T*)
  using *assms* **apply** (*induction rule*: $cdcl_{NOT}$*-restart.induct*)
    prefer *2* **apply** (*metis rtranclp-unfold fstI full1-def rtranclp-*$cdcl_{NOT}$*-bound-inv*)
  by (*metis* $cdcl_{NOT}$*-bound-inv* $cdcl_{NOT}$*-*$cdcl_{NOT}$*-inv* $cdcl_{NOT}$*-restart-inv fst-conv*)

lemma $cdcl_{NOT}$*-with-restart-*$cdcl_{NOT}$*-inv*:
  assumes
    $cdcl_{NOT}$*-restart S T* **and**
    $cdcl_{NOT}$*-inv* (*fst S*)
  shows $cdcl_{NOT}$*-inv* (*fst T*)
  using *assms* **apply** *induction*
    apply (*metis* $cdcl_{NOT}$*-*$cdcl_{NOT}$*-inv* $cdcl_{NOT}$*-inv-restart fst-conv*)
   apply (*metis fstI full-def full-unfold rtranclp-*$cdcl_{NOT}$*-*$cdcl_{NOT}$*-inv*)
  done

lemma *rtranclp-*$cdcl_{NOT}$*-with-restart-*$cdcl_{NOT}$*-inv*:
  assumes
    $cdcl_{NOT}$*-restart*** *S T* **and**
    $cdcl_{NOT}$*-inv* (*fst S*)
  shows $cdcl_{NOT}$*-inv* (*fst T*)
  using *assms* **by** *induction* (*auto intro*: $cdcl_{NOT}$*-with-restart-*$cdcl_{NOT}$*-inv*)

lemma *rtranclp-*$cdcl_{NOT}$*-with-restart-bound-inv*:
  assumes
    $cdcl_{NOT}$*-restart*** *S T* **and**
    $cdcl_{NOT}$*-inv* (*fst S*) **and**
    bound-inv A (*fst S*)
  shows *bound-inv A* (*fst T*)
  using *assms* **apply** *induction*
   apply (*simp add*: $cdcl_{NOT}$*-*$cdcl_{NOT}$*-inv* $cdcl_{NOT}$*-with-restart-bound-inv*)
  using $cdcl_{NOT}$*-with-restart-bound-inv rtranclp-*$cdcl_{NOT}$*-with-restart-*$cdcl_{NOT}$*-inv* **by** *blast*

lemma $cdcl_{NOT}$*-with-restart-increasing-number*:
  $cdcl_{NOT}$*-restart S T* $\Longrightarrow$ *snd T* = *1* + *snd S*
  by (*induction rule*: $cdcl_{NOT}$*-restart.induct*) *auto*
end

locale $cdcl_{NOT}$*-increasing-restarts* =
  $cdcl_{NOT}$*-increasing-restarts-ops restart* $cdcl_{NOT}$ *f bound-inv* μ $cdcl_{NOT}$*-inv* μ*-bound*
  for
    trail :: *'st* $\Rightarrow$ (*'v*, *unit*, *unit*) *ann-literals* **and**
    clauses :: *'st* $\Rightarrow$ *'v clauses* **and**
    prepend-trail :: (*'v*, *unit*, *unit*) *ann-literal* $\Rightarrow$ *'st* $\Rightarrow$ *'st* **and**

tl-trail :: *'st* ⇒ *'st* **and**
add-cls$_{NOT}$ *remove-cls*$_{NOT}$:: *'v clause* ⇒ *'st* ⇒ *'st* **and**
f :: *nat* ⇒ *nat* **and**
restart :: *'st* ⇒ *'st* ⇒ *bool* **and**
bound-inv :: *'bound* ⇒ *'st* ⇒ *bool* **and**
μ :: *'bound* ⇒ *'st* ⇒ *nat* **and**
cdcl$_{NOT}$ :: *'st* ⇒ *'st* ⇒ *bool* **and**
cdcl$_{NOT}$*-inv* :: *'st* ⇒ *bool* **and**
μ*-bound* :: *'bound* ⇒ *'st* ⇒ *nat* +
assumes
measure-bound: $\bigwedge$*A T V n. cdcl*$_{NOT}$*-inv T* ⟹ *bound-inv A T*
⟹ *cdcl*$_{NOT}$*-restart* (*T*, *n*) (*V*, *Suc n*) ⟹ μ *A V* ≤ μ*-bound A T* **and**
cdcl$_{NOT}$*-raw-restart-*μ*-bound*:
cdcl$_{NOT}$*-restart* (*T*, *a*) (*V*, *b*) ⟹ *cdcl*$_{NOT}$*-inv T* ⟹ *bound-inv A T*
⟹ μ*-bound A V* ≤ μ*-bound A T*
begin

lemma *rtranclp-cdcl*$_{NOT}$*-raw-restart-*μ*-bound*:
cdcl$_{NOT}$*-restart*** (*T*, *a*) (*V*, *b*) ⟹ *cdcl*$_{NOT}$*-inv T* ⟹ *bound-inv A T*
⟹ μ*-bound A V* ≤ μ*-bound A T*
apply (*induction rule*: *rtranclp-induct2*)
apply *simp*
by (*metis cdcl*$_{NOT}$*-raw-restart-*μ*-bound dual-order.trans fst-conv*
rtranclp-cdcl$_{NOT}$*-with-restart-bound-inv rtranclp-cdcl*$_{NOT}$*-with-restart-cdcl*$_{NOT}$*-inv*)

lemma *cdcl*$_{NOT}$*-raw-restart-measure-bound*:
cdcl$_{NOT}$*-restart* (*T*, *a*) (*V*, *b*) ⟹ *cdcl*$_{NOT}$*-inv T* ⟹ *bound-inv A T*
⟹ μ *A V* ≤ μ*-bound A T*
apply (*cases rule*: *cdcl*$_{NOT}$*-restart.cases*)
apply *simp*
using *measure-bound relpowp-imp-rtranclp* **apply** *fastforce*
by (*metis full-def full-unfold measure-bound2 prod.inject*)

lemma *rtranclp-cdcl*$_{NOT}$*-raw-restart-measure-bound*:
cdcl$_{NOT}$*-restart*** (*T*, *a*) (*V*, *b*) ⟹ *cdcl*$_{NOT}$*-inv T* ⟹ *bound-inv A T*
⟹ μ *A V* ≤ μ*-bound A T*
apply (*induction rule*: *rtranclp-induct2*)
apply (*simp add*: *measure-bound2*)
by (*metis dual-order.trans fst-conv measure-bound2 r-into-rtranclp rtranclp.rtrancl-refl*
rtranclp-cdcl$_{NOT}$*-with-restart-bound-inv rtranclp-cdcl*$_{NOT}$*-with-restart-cdcl*$_{NOT}$*-inv*
rtranclp-cdcl$_{NOT}$*-raw-restart-*μ*-bound*)

lemma *wf-cdcl*$_{NOT}$*-restart*:
wf {(*T*, *S*). *cdcl*$_{NOT}$*-restart S T* ∧ *cdcl*$_{NOT}$*-inv* (*fst S*)} (**is** *wf ?A*)
proof (*rule ccontr*)
assume ¬ *?thesis*
then obtain *g* **where**
g: $\bigwedge$*i. cdcl*$_{NOT}$*-restart* (*g i*) (*g* (*Suc i*)) **and**
cdcl$_{NOT}$*-inv-g*: $\bigwedge$*i. cdcl*$_{NOT}$*-inv* (*fst* (*g i*))
unfolding *wf-iff-no-infinite-down-chain* **by** *fast*

have *snd-g*: $\bigwedge$*i. snd* (*g i*) = *i* + *snd* (*g 0*)
apply (*induct-tac i*)
apply *simp*
by (*metis Suc-eq-plus1-left add.commute add.left-commute*

cdcl$_{NOT}$*-with-restart-increasing-number g*)
then have *snd-g-0*: $\bigwedge i.\ i > 0 \Longrightarrow snd\ (g\ i) = i + snd\ (g\ 0)$
by *blast*
have *unbounded-f-g*: *unbounded* ($\lambda i.$ *f* (*snd* (*g i*)))
using *f* **unfolding** *bounded-def* **by** (*metis add.commute f less-or-eq-imp-le snd-g not-bounded-nat-exists-larger not-le le-iff-add*)

{ fix *i*
have *H*: $\bigwedge T\ Ta\ m.$ (*cdcl*$_{NOT}$ ^^ *m*) *T Ta* $\Longrightarrow$ *no-step cdcl*$_{NOT}$ *T* $\Longrightarrow m = 0$
apply (*case-tac m*) **by** *simp* (*meson relpowp-E2*)
have $\exists$ *T m*. (*cdcl*$_{NOT}$ ^^ *m*) (*fst* (*g i*)) *T* $\wedge$ *m* $\geq$ *f* (*snd* (*g i*))
using *g*[*of i*] **apply** (*cases rule*: *cdcl*$_{NOT}$*-restart.cases*)
apply *auto*[]
using *g*[*of Suc i*] *f-ge-1* **apply** (*cases rule*: *cdcl*$_{NOT}$*-restart.cases*)
apply (*auto simp add*: *full1-def full-def dest*: *H dest*: *tranclpD*)
using *H Suc-leI leD* **by** *blast*
} note *H* = *this*
obtain *A* **where** *bound-inv A* (*fst* (*g 1*))
using *g*[*of 0*] *cdcl*$_{NOT}$*-inv-g*[*of 0*] **apply** (*cases rule*: *cdcl*$_{NOT}$*-restart.cases*)
apply (*metis One-nat-def cdcl*$_{NOT}$*-inv exists-bound fst-conv relpowp-imp-rtranclp rtranclp-induct*)
using *H*[*of 1*] **unfolding** *full1-def* **by** (*metis One-nat-def Suc-eq-plus1 diff-is-0-eq′ diff-zero f-ge-1 fst-conv le-add2 relpowp-E2 snd-conv*)
let *?j* = μ*-bound A* (*fst* (*g 1*)) + *1*
obtain *j* **where**
j: *f* (*snd* (*g j*)) > *?j* **and** *j* > *1*
using *unbounded-f-g not-bounded-nat-exists-larger* **by** *blast*
{
fix *i j*
have *cdcl*$_{NOT}$*-with-restart*: *j* $\geq$ *i* $\Longrightarrow$ *cdcl*$_{NOT}$*-restart*** (*g i*) (*g j*)
apply (*induction j*)
apply *simp*
by (*metis g le-Suc-eq rtranclp.rtrancl-into-rtrancl rtranclp.rtrancl-refl*)
} note *cdcl*$_{NOT}$*-restart* = *this*
have *cdcl*$_{NOT}$*-inv* (*fst* (*g* (*Suc 0*)))
by (*simp add*: *cdcl*$_{NOT}$*-inv-g*)
have *cdcl*$_{NOT}$*-restart*** (*fst* (*g 1*), *snd* (*g 1*)) (*fst* (*g j*), *snd* (*g j*))
using ‹*j*> *1*› **by** (*simp add*: *cdcl*$_{NOT}$*-restart*)
have μ *A* (*fst* (*g j*)) $\leq$ μ*-bound A* (*fst* (*g 1*))
apply (*rule rtranclp-cdcl*$_{NOT}$*-raw-restart-measure-bound*)
using ‹*cdcl*$_{NOT}$*-restart*** (*fst* (*g 1*), *snd* (*g 1*)) (*fst* (*g j*), *snd* (*g j*))› **apply** *blast*
apply (*simp add*: *cdcl*$_{NOT}$*-inv-g*)
using ‹*bound-inv A* (*fst* (*g 1*))› **apply** *simp*
done
then have μ *A* (*fst* (*g j*)) $\leq$ *?j*
by *auto*
have *inv*: *bound-inv A* (*fst* (*g j*))
using ‹*bound-inv A* (*fst* (*g 1*))› ‹*cdcl*$_{NOT}$*-inv* (*fst* (*g* (*Suc 0*)))›
‹*cdcl*$_{NOT}$*-restart*** (*fst* (*g 1*), *snd* (*g 1*)) (*fst* (*g j*), *snd* (*g j*))›
rtranclp-cdcl$_{NOT}$*-with-restart-bound-inv* **by** *auto*
obtain *T m* **where**
cdcl$_{NOT}$*-m*: (*cdcl*$_{NOT}$ ^^ *m*) (*fst* (*g j*)) *T* **and**
f-m: *f* (*snd* (*g j*)) $\leq$ *m*
using *H*[*of j*] **by** *blast*
have *?j* < *m*

using *f-m j Nat.le-trans* **by** *linarith*

then show *False*
using ‹μ *A* (*fst* (*g j*)) $\leq$ μ*-bound A* (*fst* (*g 1*))›
$cdcl_{NOT}$*-comp-bounded*[*OF inv* $cdcl_{NOT}$*-inv-g*, *of*] $cdcl_{NOT}$*-inv-g* $cdcl_{NOT}$*-m*
‹*?j* < *m*› **by** *auto*
qed

lemma $cdcl_{NOT}$*-restart-steps-bigger-than-bound*:
assumes
$cdcl_{NOT}$*-restart S T* **and**
bound-inv A (*fst S*) **and**
$cdcl_{NOT}$*-inv* (*fst S*) **and**
f (*snd S*) > μ*-bound A* (*fst S*)
shows *full1* $cdcl_{NOT}$ (*fst S*) (*fst T*)
using *assms*
proof (*induction rule*: $cdcl_{NOT}$*-restart.induct*)
case *restart-full*
then show *?case* **by** *auto*
next
case (*restart-step m S T n U*) **note** *st* = *this*(*1*) **and** *f* = *this*(*2*) **and** *bound-inv* = *this*(*4*) **and** $cdcl_{NOT}$*-inv* =*this*(*5*) **and** μ = *this*(*6*)
then obtain *m'* **where** *m*: *m* = *Suc m'* **by** (*cases m*) *auto*
have μ *A S* − *m'* = *0*
using *f bound-inv* $cdcl_{NOT}$*-inv* μ *m rtranclp-*$cdcl_{NOT}$*-raw-restart-measure-bound* **by** *fastforce*
then have *False* **using** $cdcl_{NOT}$*-comp-n-le*[*of m' S T A*] *restart-step* **unfolding** *m* **by** *simp*
then show *?case* **by** *fast*
qed

lemma *rtranclp-*$cdcl_{NOT}$*-with-inv-inv-rtranclp-*$cdcl_{NOT}$:
assumes
inv: $cdcl_{NOT}$*-inv S* **and**
binv: *bound-inv A S*
shows (λ*S T*. $cdcl_{NOT}$ *S T* $\wedge$ $cdcl_{NOT}$*-inv S* $\wedge$ *bound-inv A S*)** *S T* $\longleftrightarrow$ $cdcl_{NOT}{}^{**}$ *S T*
(**is** *?A*** *S T* $\longleftrightarrow$ *?B*** *S T*)
apply (*rule iffI*)
using *rtranclp-mono*[*of ?A ?B*] **apply** *blast*
apply (*induction rule*: *rtranclp-induct*)
using *inv binv* **apply** *simp*
by (*metis* (*mono-tags*, *lifting*) *binv inv rtranclp.simps rtranclp-*$cdcl_{NOT}$*-bound-inv rtranclp-*$cdcl_{NOT}$*-*$cdcl_{NOT}$*-inv*)

lemma *no-step-*$cdcl_{NOT}$*-restart-no-step-*$cdcl_{NOT}$:
assumes
n-s: *no-step* $cdcl_{NOT}$*-restart S* **and**
inv: $cdcl_{NOT}$*-inv* (*fst S*) **and**
binv: *bound-inv A* (*fst S*)
shows *no-step* $cdcl_{NOT}$ (*fst S*)
proof (*rule ccontr*)
assume ¬ *?thesis*
then obtain *T* **where** *T*: $cdcl_{NOT}$ (*fst S*) *T*
by *blast*
then obtain *U* **where** *U*: *full* (λ*S T*. $cdcl_{NOT}$ *S T* $\wedge$ $cdcl_{NOT}$*-inv S* $\wedge$ *bound-inv A S*) *T U*
using *wf-exists-normal-form-full*[*OF wf-*$cdcl_{NOT}$, *of A T*] **by** *auto*
moreover have *inv-T*: $cdcl_{NOT}$*-inv T*

using ‹*cdcl*$_{NOT}$ (*fst S*) *T*› *cdcl*$_{NOT}$*-inv inv* **by** *blast*
moreover have *b-inv-T*: *bound-inv A T*
using ‹*cdcl*$_{NOT}$ (*fst S*) *T*› *binv bound-inv inv* **by** *blast*
ultimately have *full cdcl*$_{NOT}$ *T U*
using *rtranclp-cdcl*$_{NOT}$*-with-inv-inv-rtranclp-cdcl*$_{NOT}$ *rtranclp-cdcl*$_{NOT}$*-bound-inv* *rtranclp-cdcl*$_{NOT}$*-cdcl*$_{NOT}$*-inv* **unfolding** *full-def* **by** *blast*
then have *full1 cdcl*$_{NOT}$ (*fst S*) *U*
using *T full-fullI* **by** *metis*
then show *False* **by** (*metis n-s prod.collapse restart-full*)
qed

end

2.8 Merging backjump and learning

locale *cdcl*$_{NOT}$*-merge-bj-learn-ops* =
dpll-state trail clauses prepend-trail tl-trail add-cls$_{NOT}$ *remove-cls*$_{NOT}$ +
decide-ops trail clauses prepend-trail tl-trail add-cls$_{NOT}$ *remove-cls*$_{NOT}$ +
forget-ops trail clauses prepend-trail tl-trail add-cls$_{NOT}$ *remove-cls*$_{NOT}$ *forget-cond* +
propagate-ops trail clauses prepend-trail tl-trail add-cls$_{NOT}$ *remove-cls*$_{NOT}$ *propagate-conds*
for
trail :: *'st* ⇒ (*'v*, *unit*, *unit*) *ann-literals* **and**
clauses :: *'st* ⇒ *'v clauses* **and**
prepend-trail :: (*'v*, *unit*, *unit*) *ann-literal* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒ *'st* **and**
add-cls$_{NOT}$ *remove-cls*$_{NOT}$:: *'v clause* ⇒ *'st* ⇒ *'st* **and**
propagate-conds :: (*'v*, *unit*, *unit*) *ann-literal* ⇒ *'st* ⇒ *bool* **and**
forget-cond :: *'v clause* ⇒ *'st* ⇒ *bool* +
fixes *backjump-l-cond* :: *'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *bool*
begin
inductive *backjump-l* **where**
backjump-l: *trail S* = *F'* @ *Marked K* () # *F*
⟹ *no-dup* (*trail S*)
⟹ *T* ∼ *prepend-trail* (*Propagated L* ()) (*reduce-trail-to*$_{NOT}$ *F* (*add-cls*$_{NOT}$ (*C'* + {#*L*#}) *S*))
⟹ *C* ∈# *clauses S*
⟹ *trail S* ⊨*as CNot C*
⟹ *undefined-lit F L*
⟹ *atm-of L* ∈ *atms-of-msu* (*clauses S*) ∪ *atm-of* ‘ (*lits-of* (*trail S*))
⟹ *clauses S* ⊨*pm C'* + {#*L*#}
⟹ *F* ⊨*as CNot C'*
⟹ *backjump-l-cond C C' L T*
⟹ *backjump-l S T*
inductive-cases *backjump-lE*: *backjump-l S T*

inductive *cdcl*$_{NOT}$*-merged-bj-learn* :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
cdcl$_{NOT}$*-merged-bj-learn-decide*$_{NOT}$: *decide*$_{NOT}$ *S S'* ⟹ *cdcl*$_{NOT}$*-merged-bj-learn S S'* |
cdcl$_{NOT}$*-merged-bj-learn-propagate*$_{NOT}$: *propagate*$_{NOT}$ *S S'* ⟹ *cdcl*$_{NOT}$*-merged-bj-learn S S'* |
cdcl$_{NOT}$*-merged-bj-learn-backjump-l*: *backjump-l S S'* ⟹ *cdcl*$_{NOT}$*-merged-bj-learn S S'* |
cdcl$_{NOT}$*-merged-bj-learn-forget*$_{NOT}$: *forget*$_{NOT}$ *S S'* ⟹ *cdcl*$_{NOT}$*-merged-bj-learn S S'*

lemma *cdcl*$_{NOT}$*-merged-bj-learn-no-dup-inv*:
cdcl$_{NOT}$*-merged-bj-learn S T* ⟹ *no-dup* (*trail S*) ⟹ *no-dup* (*trail T*)
apply (*induction rule*: *cdcl*$_{NOT}$*-merged-bj-learn.induct*)
using *defined-lit-map* **apply** *fastforce*
using *defined-lit-map* **apply** *fastforce*
apply (*force simp*: *defined-lit-map elim*!: *backjump-lE*)[]

using *forget*$_{NOT}$*.simps* **apply** *auto[1]*
done
end

locale *cdcl*$_{NOT}$*-merge-bj-learn-proxy* =
cdcl$_{NOT}$*-merge-bj-learn-ops trail clauses prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$
propagate-conds forget-conds $\lambda C\ C'\ L'\ S.$ *backjump-l-cond* $C\ C'\ L'\ S$
$\wedge$ *distinct-mset* $(C' + \{\#L'\#\}) \wedge \neg$*tautology* $(C' + \{\#L'\#\})$
for
trail :: $'st \Rightarrow ('v,\ unit,\ unit)$ *ann-literals* **and**
clauses :: $'st \Rightarrow\ 'v$ *clauses* **and**
prepend-trail :: $('v,\ unit,\ unit)$ *ann-literal* $\Rightarrow\ 'st \Rightarrow\ 'st$ **and**
tl-trail :: $'st \Rightarrow\ 'st$ **and**
add-cls$_{NOT}$ *remove-cls*$_{NOT}$:: $'v$ *clause* $\Rightarrow\ 'st \Rightarrow\ 'st$ **and**
propagate-conds :: $('v,\ unit,\ unit)$ *ann-literal* $\Rightarrow\ 'st \Rightarrow$ *bool* **and**
forget-conds :: $'v$ *clause* $\Rightarrow\ 'st \Rightarrow$ *bool* **and**
backjump-l-cond :: $'v$ *clause* $\Rightarrow\ 'v$ *clause* $\Rightarrow\ 'v$ *literal* $\Rightarrow\ 'st \Rightarrow$ *bool* +
fixes
inv :: $'st \Rightarrow$ *bool*
assumes
bj-merge-can-jump:
$\bigwedge S\ C\ F'\ K\ F\ L.$
inv S
$\Longrightarrow$ *trail* $S = F'$ @ *Marked* K () # F
$\Longrightarrow$ $C \in\#$ *clauses* S
$\Longrightarrow$ *trail* $S \models as$ *CNot* C
$\Longrightarrow$ *undefined-lit* $F\ L$
$\Longrightarrow$ *atm-of* $L \in$ *atms-of-msu* (*clauses* S) $\cup$ *atm-of* ‘ (*lits-of* (F' @ *Marked* K () # F))
$\Longrightarrow$ *clauses* $S \models pm\ C' + \{\#L\#\}$
$\Longrightarrow$ $F \models as$ *CNot* C'
$\Longrightarrow$ $\neg$*no-step backjump-l* S **and**
cdcl-merged-inv: $\bigwedge S\ T.$ *cdcl*$_{NOT}$*-merged-bj-learn* $S\ T \Longrightarrow$ *inv* $S \Longrightarrow$ *inv* T
begin
abbreviation *backjump-conds* **where**
backjump-conds $\equiv \lambda$- $C\ L$ - -. *distinct-mset* $(C + \{\#L\#\}) \wedge \neg$*tautology* $(C + \{\#L\#\})$

sublocale *dpll-with-backjumping-ops trail clauses prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$
propagate-conds inv backjump-conds
proof (*unfold-locales*, *goal-cases*)
case *1*
{ fix $S\ S'$
assume *bj*: *backjump-l* $S\ S'$ **and** *no-dup* (*trail* S)
then obtain $F'\ K\ F\ L\ C'\ C$ **where**
S': $S' \sim$ *prepend-trail* (*Propagated* L ()) (*reduce-trail-to*$_{NOT}$ F
(*tl-trail*(*add-cls*$_{NOT}$ $(C' + \{\#L\#\})$ S)))
and
tr-S: *trail* $S = F'$ @ *Marked* K () # F **and**
C: $C \in\#$ *clauses* S **and**
tr-S-C: *trail* $S \models as$ *CNot* C **and**
undef-L: *undefined-lit* $F\ L$ **and**
atm-L: *atm-of* $L \in$ *atms-of-msu* (*clauses* S) $\cup$ *atm-of* ‘ *lits-of* (*trail* S) **and**
cls-S-C': *clauses* $S \models pm\ C' + \{\#L\#\}$ **and**
F-C': $F \models as$ *CNot* C' **and**
dist: *distinct-mset* $(C' + \{\#L\#\})$ **and**
not-tauto: $\neg$ *tautology* $(C' + \{\#L\#\})$

by (*elim backjump-lE*) *simp*

have $\exists S'$. *backjumping-ops.backjump trail clauses prepend-trail tl-trail backjump-conds S* S'
apply *rule*
apply (*rule backjumping-ops.backjump.intros*)
apply *unfold-locales*
using *tr-S* **apply** *simp*
apply (*rule state-eq*$_{NOT}$*-ref*)
using *C* **apply** *simp*
using *tr-S-C* **apply** *simp*
using *undef-L* **apply** *simp*
using *atm-L* **apply** *simp*
using *cls-S-C'* **apply** *simp*
using *F-C'* **apply** *simp*
using *dist not-tauto* **apply** *simp*
done
} **note** $H = this(1)$
then show *?case* **using** *1 bj-merge-can-jump* **by** *meson*
qed

end

locale *cdcl*$_{NOT}$*-merge-bj-learn-proxy2* =
cdcl$_{NOT}$*-merge-bj-learn-proxy trail clauses prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$
propagate-conds forget-conds backjump-l-cond inv
for
trail :: $'st \Rightarrow ('v, unit, unit)$ *ann-literals* **and**
clauses :: $'st \Rightarrow 'v$ *clauses* **and**
prepend-trail :: $('v, unit, unit)$ *ann-literal* $\Rightarrow 'st \Rightarrow 'st$ **and**
tl-trail :: $'st \Rightarrow 'st$ **and**
add-cls$_{NOT}$ *remove-cls*$_{NOT}$:: $'v$ *clause* $\Rightarrow 'st \Rightarrow 'st$ **and**
propagate-conds :: $('v, unit, unit)$ *ann-literal* $\Rightarrow 'st \Rightarrow bool$ **and**
inv :: $'st \Rightarrow bool$ **and**
forget-conds :: $'v$ *clause* $\Rightarrow 'st \Rightarrow bool$ **and**
backjump-l-cond :: $'v$ *clause* $\Rightarrow 'v$ *clause* $\Rightarrow 'v$ *literal* $\Rightarrow 'st \Rightarrow bool$
begin

sublocale *conflict-driven-clause-learning-ops trail clauses prepend-trail tl-trail add-cls*$_{NOT}$
remove-cls$_{NOT}$ *propagate-conds inv backjump-conds* λC -. *distinct-mset* $C \wedge \neg$*tautology* C
forget-conds
by *unfold-locales*
end

locale *cdcl*$_{NOT}$*-merge-bj-learn* =
cdcl$_{NOT}$*-merge-bj-learn-proxy2 trail clauses prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$
propagate-conds inv forget-conds backjump-l-cond
for
trail :: $'st \Rightarrow ('v, unit, unit)$ *ann-literals* **and**
clauses :: $'st \Rightarrow 'v$ *clauses* **and**
prepend-trail :: $('v, unit, unit)$ *ann-literal* $\Rightarrow 'st \Rightarrow 'st$ **and**
tl-trail :: $'st \Rightarrow 'st$ **and**
add-cls$_{NOT}$ *remove-cls*$_{NOT}$:: $'v$ *clause* $\Rightarrow 'st \Rightarrow 'st$ **and**
propagate-conds :: $('v, unit, unit)$ *ann-literal* $\Rightarrow 'st \Rightarrow bool$ **and**
inv :: $'st \Rightarrow bool$ **and**
forget-conds :: $'v$ *clause* $\Rightarrow 'st \Rightarrow bool$ **and**

backjump-l-cond :: $'v$ *clause* $\Rightarrow$ $'v$ *clause* $\Rightarrow$ $'v$ *literal* $\Rightarrow$ $'st$ $\Rightarrow$ *bool* +
assumes
dpll-bj-inv: $\bigwedge S\ T.$ *dpll-bj* $S\ T \Longrightarrow$ *inv* $S \Longrightarrow$ *inv* T **and**
learn-inv: $\bigwedge S\ T.$ *learn* $S\ T \Longrightarrow$ *inv* $S \Longrightarrow$ *inv* T
begin

interpretation $cdcl_{NOT}$:
conflict-driven-clause-learning trail clauses prepend-trail tl-trail $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
propagate-conds inv backjump-conds λC -. *distinct-mset* $C \wedge \neg$*tautology* C *forget-conds*
apply *unfold-locales*
apply (*simp only*: $cdcl_{NOT}$*.simps*)
using $cdcl_{NOT}$*-merged-bj-learn-forget*$_{NOT}$ *cdcl-merged-inv learn-inv*
by (*auto simp add*: $cdcl_{NOT}$*.simps dpll-bj-inv*)

lemma *backjump-l-learn-backjump*:
assumes *bt*: *backjump-l* $S\ T$ **and** *inv*: *inv* S **and** *n-d*: *no-dup* (*trail* S)
shows $\exists\, C'\ L.$ *learn* S ($add\text{-}cls_{NOT}$ $(C' + \{\#L\#\})$ S)
$\wedge$ *backjump* ($add\text{-}cls_{NOT}$ $(C' + \{\#L\#\})$ S) T
$\wedge$ *atms-of* $(C' + \{\#L\#\}) \subseteq$ *atms-of-msu* (*clauses* S) $\cup$ *atm-of* ‘ (*lits-of* (*trail* S))
proof −
obtain $C\ F'\ K\ F\ L\ l\ C'$ **where**
tr-S: *trail* $S = F'$ @ *Marked* K () # F **and**
T: $T \sim$ *prepend-trail* (*Propagated* $L\ l$) ($reduce\text{-}trail\text{-}to_{NOT}$ F ($add\text{-}cls_{NOT}$ $(C' + \{\#L\#\})$ S)) **and**
C-cls-S: $C \in\#$ *clauses* S **and**
tr-S-CNot-C: *trail* $S \models$*as CNot* C **and**
undef: *undefined-lit* $F\ L$ **and**
atm-L: *atm-of* $L \in$ *atms-of-msu* (*clauses* S) $\cup$ *atm-of* ‘ (*lits-of* (*trail* S)) **and**
clss-C: *clauses* $S \models$*pm* $C' + \{\#L\#\}$ **and**
$F \models$*as CNot* C' **and**
distinct: *distinct-mset* $(C' + \{\#L\#\})$ **and**
not-tauto: $\neg$ *tautology* $(C' + \{\#L\#\})$
using *bt inv* **by** (*elim backjump-lE*) *simp*
have *atms-C'*: *atms-of* $C' \subseteq$ *atm-of* ‘ (*lits-of* F)
proof −
obtain *ll* :: $'v \Rightarrow ('v$ *literal* $\Rightarrow 'v) \Rightarrow 'v$ *literal set* $\Rightarrow 'v$ *literal* **where**
$\forall\, v\, f\, L.\ v \notin f$ ‘ $L \vee v = f\ (ll\ v\ f\ L) \wedge ll\ v\ f\ L \in L$
by *moura*
then show *?thesis* **unfolding** *tr-S*
by (*metis* (*no-types*) ‹$F \models$*as CNot* C'› *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set*
atms-of-def in-CNot-implies-uminus(*2*) *mem-set-mset-iff subsetI*)
qed
then have *atms-of* $(C' + \{\#L\#\}) \subseteq$ *atms-of-msu* (*clauses* S) $\cup$ *atm-of* ‘ (*lits-of* (*trail* S))
using *atm-L tr-S* **by** *auto*
moreover have *learn*: *learn* S ($add\text{-}cls_{NOT}$ $(C' + \{\#L\#\})$ S)
apply (*rule learn.intros*)
apply (*rule clss-C*)
using *atms-C' atm-L* **apply** (*fastforce simp add*: *tr-S in-plus-implies-atm-of-on-atms-of-ms*)[]
apply *standard*
apply (*rule distinct*)
apply (*rule not-tauto*)
apply *simp*
done
moreover have *bj*: *backjump* ($add\text{-}cls_{NOT}$ $(C' + \{\#L\#\})$ S) T
apply (*rule backjump.intros*)
using ‹$F \models$*as CNot* C'› *C-cls-S tr-S-CNot-C undef T distinct not-tauto n-d*

by (*auto simp*: *tr-S state-eq*$_{NOT}$*-def simp del*: *state-simp*$_{NOT}$)
ultimately show *?thesis* **by** *auto*
qed

lemma *cdcl*$_{NOT}$*-merged-bj-learn-is-tranclp-cdcl*$_{NOT}$:
cdcl$_{NOT}$*-merged-bj-learn S T* $\Longrightarrow$ *inv S* $\Longrightarrow$ *no-dup* (*trail S*) $\Longrightarrow$ *cdcl*$_{NOT}{}^{++}$ *S T*
proof (*induction rule*: *cdcl*$_{NOT}$*-merged-bj-learn.induct*)
case (*cdcl*$_{NOT}$*-merged-bj-learn-decide*$_{NOT}$ *T*)
then have *cdcl*$_{NOT}$ *S T*
using *bj-decide*$_{NOT}$ *cdcl*$_{NOT}$*.simps* **by** *fastforce*
then show *?case* **by** *auto*
next
case (*cdcl*$_{NOT}$*-merged-bj-learn-propagate*$_{NOT}$ *T*)
then have *cdcl*$_{NOT}$ *S T*
using *bj-propagate*$_{NOT}$ *cdcl*$_{NOT}$*.simps* **by** *fastforce*
then show *?case* **by** *auto*
next
case (*cdcl*$_{NOT}$*-merged-bj-learn-forget*$_{NOT}$ *T*)
then have *cdcl*$_{NOT}$ *S T*
using *c-forget*$_{NOT}$ **by** *blast*
then show *?case* **by** *auto*
next
case (*cdcl*$_{NOT}$*-merged-bj-learn-backjump-l T*) **note** *bt* = *this*(*1*) **and** *inv* = *this*(*2*) **and**
n-d = *this*(*3*)
obtain C' :: *'v literal multiset* **and** *L* :: *'v literal* **where**
f3: *learn S* (*add-cls*$_{NOT}$ (C' + {#*L*#}) *S*) $\wedge$
backjump (*add-cls*$_{NOT}$ (C' + {#*L*#}) *S*) *T* $\wedge$
atms-of (C' + {#*L*#}) $\subseteq$ *atms-of-msu* (*clauses S*) $\cup$ *atm-of* ‘ *lits-of* (*trail S*)
using *n-d backjump-l-learn-backjump*[*OF bt inv*] **by** *blast*
then have *f4*: *cdcl*$_{NOT}$ *S* (*add-cls*$_{NOT}$ (C' + {#*L*#}) *S*)
using *n-d c-learn* **by** *blast*
have *cdcl*$_{NOT}$ (*add-cls*$_{NOT}$ (C' + {#*L*#}) *S*) *T*
using *f3 n-d bj-backjump c-dpll-bj* **by** *blast*
then show *?case*
using *f4* **by** (*meson tranclp.r-into-trancl tranclp.trancl-into-trancl*)
qed

lemma *rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-is-rtranclp-cdcl*$_{NOT}$*-and-inv*:
cdcl$_{NOT}$*-merged-bj-learn*** *S T* $\Longrightarrow$ *inv S* $\Longrightarrow$ *no-dup* (*trail S*) $\Longrightarrow$ *cdcl*$_{NOT}{}^{**}$ *S T* $\wedge$ *inv T*
proof (*induction rule*: *rtranclp-induct*)
case *base*
then show *?case* **by** *auto*
next
case (*step T U*) **note** *st* =*this*(*1*) **and** *cdcl*$_{NOT}$ = *this*(*2*) **and** *IH* = *this*(*3*)[*OF this*(*4*−)] **and**
inv = *this*(*4*) **and** *n-d* = *this*(*5*)
have *cdcl*$_{NOT}{}^{**}$ *T U*
using *cdcl*$_{NOT}$*-merged-bj-learn-is-tranclp-cdcl*$_{NOT}$[*OF cdcl*$_{NOT}$] *IH*
cdcl$_{NOT}$*.rtranclp-cdcl*$_{NOT}$*-no-dup inv n-d* **by** *auto*
then have *cdcl*$_{NOT}{}^{**}$ *S U* **using** *IH* **by** *fastforce*
moreover have *inv U* **using** *n-d IH* ‹*cdcl*$_{NOT}{}^{**}$ *T U*› *cdcl*$_{NOT}$*.rtranclp-cdcl*$_{NOT}$*-inv* **by** *blast*
ultimately show *?case* **using** *st* **by** *fast*
qed

lemma *rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-is-rtranclp-cdcl*$_{NOT}$:
cdcl$_{NOT}$*-merged-bj-learn*** *S T* $\Longrightarrow$ *inv S* $\Longrightarrow$*no-dup* (*trail S*) $\Longrightarrow$ *cdcl*$_{NOT}{}^{**}$ *S T*

using *rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-is-rtranclp-cdcl*$_{NOT}$*-and-inv* **by** *blast*

lemma *rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-inv*:
cdcl$_{NOT}$*-merged-bj-learn*** S T $\Longrightarrow$ *inv* S $\Longrightarrow$ *no-dup* (*trail* S) $\Longrightarrow$ *inv* T
using *rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-is-rtranclp-cdcl*$_{NOT}$*-and-inv* **by** *blast*

definition $\mu_C{}'$ :: *'v literal multiset set* $\Rightarrow$ *'st* $\Rightarrow$ *nat* **where**
$\mu_C{}'$ A T $\equiv$ μ_C (*1+card* (*atms-of-ms* A)) (*2+card* (*atms-of-ms* A)) (*trail-weight* T)

definition $\mu_{CDCL}{}'$*-merged* :: *'v literal multiset set* $\Rightarrow$ *'st* $\Rightarrow$ *nat* **where**
$\mu_{CDCL}{}'$*-merged* A T $\equiv$
((*2+card* (*atms-of-ms* A)) ^ (*1+card* (*atms-of-ms* A)) $-$ $\mu_C{}'$ A T) $*$ *2* + *card* (*set-mset* (*clauses* T))

lemma *cdcl*$_{NOT}$*-decreasing-measure'*:
assumes
cdcl$_{NOT}$*-merged-bj-learn* S T **and**
inv: *inv* S **and**
atm-clss: *atms-of-msu* (*clauses* S) $\subseteq$ *atms-of-ms* A **and**
atm-trail: *atm-of* ' *lits-of* (*trail* S) $\subseteq$ *atms-of-ms* A **and**
n-d: *no-dup* (*trail* S) **and**
fin-A: *finite* A
shows $\mu_{CDCL}{}'$*-merged* A T $<$ $\mu_{CDCL}{}'$*-merged* A S
using *assms*(*1*)
proof *induction*
case (*cdcl*$_{NOT}$*-merged-bj-learn-decide*$_{NOT}$ T)
have *clauses* S = *clauses* T
using *cdcl*$_{NOT}$*-merged-bj-learn-decide*$_{NOT}$*.hyps* **by** *auto*
moreover have
(*2* + *card* (*atms-of-ms* A)) ^ (*1* + *card* (*atms-of-ms* A))
$-$ μ_C (*1* + *card* (*atms-of-ms* A)) (*2* + *card* (*atms-of-ms* A)) (*trail-weight* T)
$<$ (*2* + *card* (*atms-of-ms* A)) ^ (*1* + *card* (*atms-of-ms* A))
$-$ μ_C (*1* + *card* (*atms-of-ms* A)) (*2* + *card* (*atms-of-ms* A)) (*trail-weight* S)
apply (*rule dpll-bj-trail-mes-decreasing-prop*)
using *cdcl*$_{NOT}$*-merged-bj-learn-decide*$_{NOT}$ *fin-A atm-clss atm-trail n-d inv*
by (*simp-all add*: *bj-decide*$_{NOT}$ *cdcl*$_{NOT}$*-merged-bj-learn-decide*$_{NOT}$*.hyps*)
ultimately show *?case*
unfolding $\mu_{CDCL}{}'$*-merged-def* $\mu_C{}'$*-def* **by** *simp*
next
case (*cdcl*$_{NOT}$*-merged-bj-learn-propagate*$_{NOT}$ T)
have *clauses* S = *clauses* T
using *cdcl*$_{NOT}$*-merged-bj-learn-propagate*$_{NOT}$*.hyps*
by (*simp add*: *bj-propagate*$_{NOT}$ *inv dpll-bj-clauses*)
moreover have
(*2* + *card* (*atms-of-ms* A)) ^ (*1* + *card* (*atms-of-ms* A))
$-$ μ_C (*1* + *card* (*atms-of-ms* A)) (*2* + *card* (*atms-of-ms* A)) (*trail-weight* T)
$<$ (*2* + *card* (*atms-of-ms* A)) ^ (*1* + *card* (*atms-of-ms* A))
$-$ μ_C (*1* + *card* (*atms-of-ms* A)) (*2* + *card* (*atms-of-ms* A)) (*trail-weight* S)
apply (*rule dpll-bj-trail-mes-decreasing-prop*)
using *inv n-d atm-clss atm-trail fin-A* **by** (*simp-all add*: *bj-propagate*$_{NOT}$
cdcl$_{NOT}$*-merged-bj-learn-propagate*$_{NOT}$*.hyps*)
ultimately show *?case*
unfolding $\mu_{CDCL}{}'$*-merged-def* $\mu_C{}'$*-def* **by** *simp*
next
case (*cdcl*$_{NOT}$*-merged-bj-learn-forget*$_{NOT}$ T)
have *card* (*set-mset* (*clauses* T)) $<$ *card* (*set-mset* (*clauses* S))

using ‹*forget*$_{NOT}$ *S T*› **by** (*metis card-Diff1-less cdcl*$_{NOT}$*-merged-bj-learn-forget*$_{NOT}$*.hyps clauses-remove-cls*$_{NOT}$ *finite-set-mset forget*$_{NOT}$*E mem-set-mset-iff order-refl set-mset-minus-replicate-mset*(*1*) *state-eq*$_{NOT}$*-clauses*)
moreover
have *trail S* = *trail T*
using ‹*forget*$_{NOT}$ *S T*› **by** (*auto elim*: *forget*$_{NOT}$*E*)
then have
(*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*))
− μ_C (*1* + *card* (*atms-of-ms A*)) (*2* + *card* (*atms-of-ms A*)) (*trail-weight T*)
= (*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*))
− μ_C (*1* + *card* (*atms-of-ms A*)) (*2* + *card* (*atms-of-ms A*)) (*trail-weight S*)
by *auto*
ultimately show *?case*
unfolding μ_{CDCL}*'-merged-def* μ_C*'-def* **by** *simp*
next
case (*cdcl*$_{NOT}$*-merged-bj-learn-backjump-l T*) **note** *bj-l* = *this*(*1*)
obtain *C' L* **where**
learn: *learn S* (*add-cls*$_{NOT}$ (*C'* + {#*L*#}) *S*) **and**
bj: *backjump* (*add-cls*$_{NOT}$ (*C'* + {#*L*#}) *S*) *T* **and**
atms-C: *atms-of* (*C'* + {#*L*#}) ⊆ *atms-of-msu* (*clauses S*) ∪ *atm-of* ' (*lits-of* (*trail S*))
using *bj-l inv backjump-l-learn-backjump n-d atm-clss atm-trail* **by** *blast*
have *card-T-S*: *card* (*set-mset* (*clauses T*)) ≤ *1*+ *card* (*set-mset* (*clauses S*))
using *bj-l inv* **by** (*force elim*!: *backjump-lE simp*: *card-insert-if*)
have
((*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*))
− μ_C (*1* + *card* (*atms-of-ms A*)) (*2* + *card* (*atms-of-ms A*)) (*trail-weight T*))
< ((*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*))
− μ_C (*1* + *card* (*atms-of-ms A*)) (*2* + *card* (*atms-of-ms A*))
(*trail-weight* (*add-cls*$_{NOT}$ (*C'* + {#*L*#}) *S*)))
apply (*rule dpll-bj-trail-mes-decreasing-prop*)
using *bj bj-backjump* **apply** *blast*
using *cdcl*$_{NOT}$*.c-learn cdcl*$_{NOT}$*.cdcl*$_{NOT}$*-inv inv learn* **apply** *blast*
using *atms-C atm-clss atm-trail n-d clauses-add-cls*$_{NOT}$ **apply** *simp* **apply** *fast*
using *atm-trail n-d* **apply** *simp*
apply (*simp add*: *n-d*)
using *fin-A* **apply** *simp*
done
then have ((*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*))
− μ_C (*1* + *card* (*atms-of-ms A*)) (*2* + *card* (*atms-of-ms A*)) (*trail-weight T*))
< ((*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*))
− μ_C (*1* + *card* (*atms-of-ms A*)) (*2* + *card* (*atms-of-ms A*)) (*trail-weight S*))
using *n-d* **by** *auto*
then show *?case*
using *card-T-S* **unfolding** μ_{CDCL}*'-merged-def* μ_C*'-def* **by** *linarith*
qed

lemma *wf-cdcl*$_{NOT}$*-merged-bj-learn*:
assumes
fin-A: *finite A*
shows *wf* {(*T*, *S*).
(*inv S* ∧ *atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A* ∧ *atm-of* ' *lits-of* (*trail S*) ⊆ *atms-of-ms A*
∧ *no-dup* (*trail S*))
∧ *cdcl*$_{NOT}$*-merged-bj-learn S T*}
apply (*rule wfP-if-measure*[*of - -* μ_{CDCL}*'-merged A*])
using *cdcl*$_{NOT}$*-decreasing-measure' fin-A* **by** *simp*

lemma *tranclp-cdcl$_{NOT}$-cdcl$_{NOT}$-tranclp*:
  assumes
    cdcl$_{NOT}$-merged-bj-learn^{++} S T **and**
    inv: *inv S* **and**
    atm-clss: *atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A* **and**
    atm-trail: *atm-of ‘ lits-of* (*trail S*) ⊆ *atms-of-ms A* **and**
    n-d: *no-dup* (*trail S*) **and**
    fin-A[*simp*]: *finite A*
  shows (*T*, *S*) ∈ {(*T*, *S*).
    (*inv S* ∧ *atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A* ∧ *atm-of ‘ lits-of* (*trail S*) ⊆ *atms-of-ms A*
    ∧ *no-dup* (*trail S*))
    ∧ *cdcl$_{NOT}$-merged-bj-learn S T*}$^+$ (**is** - ∈ *?P*$^+$)
  using *assms*(*1*)
proof (*induction rule*: *tranclp-induct*)
  case *base*
  then show *?case* **using** *n-d atm-clss atm-trail inv* **by** *auto*
next
  case (*step T U*) **note** *st* = *this*(*1*) **and** *cdcl$_{NOT}$* = *this*(*2*) **and** *IH* = *this*(*3*)
  have *cdcl$_{NOT}$** S T*
    apply (*rule rtranclp-cdcl$_{NOT}$-merged-bj-learn-is-rtranclp-cdcl$_{NOT}$*)
    using *st cdcl$_{NOT}$ inv n-d atm-clss atm-trail inv* **by** *auto*
  have *inv T*
    apply (*rule rtranclp-cdcl$_{NOT}$-merged-bj-learn-inv*)
      using *inv st cdcl$_{NOT}$ n-d atm-clss atm-trail inv* **by** *auto*
  moreover have *atms-of-msu* (*clauses T*) ⊆ *atms-of-ms A*
   using *cdcl$_{NOT}$.rtranclp-cdcl$_{NOT}$-trail-clauses-bound*[*OF* ‹*cdcl$_{NOT}$** S T*› *inv n-d atm-clss atm-trail*]
    by *fast*
  moreover have *atm-of ‘* (*lits-of* (*trail T*))⊆ *atms-of-ms A*
   using *cdcl$_{NOT}$.rtranclp-cdcl$_{NOT}$-trail-clauses-bound*[*OF* ‹*cdcl$_{NOT}$** S T*› *inv n-d atm-clss atm-trail*]
    by *fast*
  moreover have *no-dup* (*trail T*)
    using *cdcl$_{NOT}$.rtranclp-cdcl$_{NOT}$-no-dup*[*OF* ‹*cdcl$_{NOT}$** S T*› *inv n-d*] **by** *fast*
  ultimately have (*U*, *T*) ∈ *?P*
    using *cdcl$_{NOT}$* **by** *auto*
  then show *?case* **using** *IH* **by** (*simp add*: *trancl-into-trancl2*)
qed

lemma *wf-tranclp-cdcl$_{NOT}$-merged-bj-learn*:
  assumes *finite A*
  shows *wf* {(*T*, *S*).
    (*inv S* ∧ *atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A* ∧ *atm-of ‘ lits-of* (*trail S*) ⊆ *atms-of-ms A*
    ∧ *no-dup* (*trail S*))
    ∧ *cdcl$_{NOT}$-merged-bj-learn^{++} S T*}
  apply (*rule wf-subset*)
   apply (*rule wf-trancl*[*OF wf-cdcl$_{NOT}$-merged-bj-learn*])
   using *assms* **apply** *simp*
  using *tranclp-cdcl$_{NOT}$-cdcl$_{NOT}$-tranclp*[*OF* - - - - - ‹*finite A*›] **by** *auto*

lemma *backjump-no-step-backjump-l*:
  backjump S T ⟹ *inv S* ⟹ ¬*no-step backjump-l S*
  apply (*elim backjumpE*)
  apply (*rule bj-merge-can-jump*)
    apply *auto*[*7*]
  by *blast*

lemma *cdcl*$_{NOT}$*-merged-bj-learn-final-state*:
fixes *A* :: *'v literal multiset set* **and** *S T* :: *'st*
assumes
n-s: *no-step cdcl*$_{NOT}$*-merged-bj-learn S* **and**
atms-S: *atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A* **and**
atms-trail: *atm-of* ' *lits-of* (*trail S*) ⊆ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
finite A **and**
inv: *inv S* **and**
decomp: *all-decomposition-implies-m* (*clauses S*) (*get-all-marked-decomposition* (*trail S*))
shows *unsatisfiable* (*set-mset* (*clauses S*))
∨ (*trail S* ⊨*asm clauses S* ∧ *satisfiable* (*set-mset* (*clauses S*)))
proof −
let *?N* = *set-mset* (*clauses S*)
let *?M* = *trail S*
consider
(*sat*) *satisfiable ?N* **and** *?M* ⊨*as ?N*
| (*sat'*) *satisfiable ?N* **and** ¬ *?M* ⊨*as ?N*
| (*unsat*) *unsatisfiable ?N*
by *auto*
then show *?thesis*
proof *cases*
case *sat'* **note** *sat* = *this*(*1*) **and** *M* = *this*(*2*)
obtain *C* **where** *C* ∈ *?N* **and** ¬*?M* ⊨*a C* **using** *M* **unfolding** *true-annots-def* **by** *auto*
obtain *I* :: *'v literal set* **where**
I ⊨*s ?N* **and**
cons: *consistent-interp I* **and**
tot: *total-over-m I ?N* **and**
atm-I-N: *atm-of* '*I* ⊆ *atms-of-ms ?N*
using *sat* **unfolding** *satisfiable-def-min* **by** *auto*
let *?I* = *I* ∪ {*P*| *P*. *P* ∈ *lits-of ?M* ∧ *atm-of P* ∉ *atm-of* ' *I*}
let *?O* = {{#*lit-of L*#} |*L*. *is-marked L* ∧ *L* ∈ *set ?M* ∧ *atm-of* (*lit-of L*) ∉ *atms-of-ms ?N*}
have *cons-I'*: *consistent-interp ?I*
using *cons* **using** ⟨*no-dup ?M*⟩ **unfolding** *consistent-interp-def*
by (*auto simp add*: *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set lits-of-def*
dest!: *no-dup-cannot-not-lit-and-uminus*)
have *tot-I'*: *total-over-m ?I* (*?N* ∪ *unmark ?M*)
using *tot atms-of-s-def* **unfolding** *total-over-m-def total-over-set-def*
by *fastforce*
have {*P* |*P*. *P* ∈ *lits-of ?M* ∧ *atm-of P* ∉ *atm-of* ' *I*} ⊨*s ?O*
using ⟨*I*⊨*s ?N*⟩ *atm-I-N* **by** (*auto simp add*: *atm-of-eq-atm-of true-clss-def lits-of-def*)
then have *I'-N*: *?I* ⊨*s ?N* ∪ *?O*
using ⟨*I*⊨*s ?N*⟩ *true-clss-union-increase* **by** *force*
have *tot'*: *total-over-m ?I* (*?N*∪*?O*)
using *atm-I-N tot* **unfolding** *total-over-m-def total-over-set-def*
by (*force simp*: *image-iff lits-of-def dest*!: *is-marked-ex-Marked*)

have *atms-N-M*: *atms-of-ms ?N* ⊆ *atm-of* ' *lits-of ?M*
proof (*rule ccontr*)
assume ¬ *?thesis*
then obtain *l* :: *'v* **where**
l-N: *l* ∈ *atms-of-ms ?N* **and**
l-M: *l* ∉ *atm-of* ' *lits-of ?M*
by *auto*

have *undefined-lit ?M* (*Pos l*)
using *l-M* **by** (*metis Marked-Propagated-in-iff-in-lits-of atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set literal.sel*(*1*))
have $decide_{NOT}$ *S* (*prepend-trail* (*Marked* (*Pos l*) ()) *S*)
by (*metis* ‹*undefined-lit ?M* (*Pos l*)› $decide_{NOT}$*.intros l-N literal.sel*(*1*) *state-eq*$_{NOT}$*-ref*)
then show *False*
using $cdcl_{NOT}$*-merged-bj-learn-decide*$_{NOT}$ *n-s* **by** *blast*
qed

have *?M* $\models$*as CNot C*
by (*metis atms-N-M* ‹$C \in$ *?N*› ‹$\neg$ *?M* $\models$*a C*› *all-variables-defined-not-imply-cnot atms-of-atms-of-ms-mono atms-of-ms-CNot-atms-of atms-of-ms-CNot-atms-of-ms subsetCE*)
have $\exists\, l \in$ *set ?M. is-marked l*
proof (*rule ccontr*)
let *?O* = {{#*lit-of L*#} |*L. is-marked L* $\wedge$ *L* $\in$ *set ?M* $\wedge$ *atm-of* (*lit-of L*) $\notin$ *atms-of-ms ?N*}
have ϑ[*iff*]: $\bigwedge$*I. total-over-m I* (*?N* $\cup$ *?O* $\cup$ *unmark ?M*)
$\longleftrightarrow$ *total-over-m I* (*?N* $\cup$*unmark ?M*)
unfolding *total-over-set-def total-over-m-def atms-of-ms-def* **by** *auto*
assume $\neg$ *?thesis*
then have [*simp*]:{{#*lit-of L*#} |*L. is-marked L* $\wedge$ *L* $\in$ *set ?M*}
= {{#*lit-of L*#} |*L. is-marked L* $\wedge$ *L* $\in$ *set ?M* $\wedge$ *atm-of* (*lit-of L*) $\notin$ *atms-of-ms ?N*}
by *auto*
then have *?N* $\cup$ *?O* $\models$*ps unmark ?M*
using *all-decomposition-implies-propagated-lits-are-implied*[*OF decomp*] **by** *auto*

then have *?I* $\models$*s unmark ?M*
using *cons-I' I'-N tot-I'* ‹*?I* $\models$*s ?N* $\cup$ *?O*› **unfolding** ϑ *true-clss-clss-def* **by** *blast*
then have *lits-of ?M* $\subseteq$ *?I*
unfolding *true-clss-def lits-of-def* **by** *auto*
then have *?M* $\models$*as ?N*
using *I'-N* ‹$C \in$ *?N*› ‹$\neg$ *?M* $\models$*a C*› *cons-I' atms-N-M*
by (*meson* ‹*trail S* $\models$*as CNot C*› *consistent-CNot-not rev-subsetD sup-ge1 true-annot-def true-annots-def true-cls-mono-set-mset-l true-clss-def*)
then show *False* **using** *M* **by** *fast*
qed
from *List.split-list-first-propE*[*OF this*] **obtain** *K* :: *'v literal* **and** *d* :: *unit* **and**
F F' :: (*'v, unit, unit*) *ann-literal list* **where**
M-K: *?M* = *F'* @ *Marked K* () # *F* **and**
nm: $\forall$*f*$\in$*set F'*. $\neg$*is-marked f*
unfolding *is-marked-def* **by** (*metis* (*full-types*) *old.unit.exhaust*)
let *?K* = *Marked K* ()::(*'v, unit, unit*) *ann-literal*
have *?K* $\in$ *set ?M*
unfolding *M-K* **by** *auto*
let *?C* = *image-mset lit-of* {#*L*$\in$#*mset ?M. is-marked L* $\wedge$ *L*$\neq$*?K*#} :: *'v literal multiset*
let *?C'* = *set-mset* (*image-mset* (λ*L*::*'v literal*. {#*L*#}) (*?C*+{#*lit-of ?K*#}))
have *?N* $\cup$ {{#*lit-of L*#} |*L. is-marked L* $\wedge$ *L* $\in$ *set ?M*} $\models$*ps unmark ?M*
using *all-decomposition-implies-propagated-lits-are-implied*[*OF decomp*] **.**
moreover have *C'*: *?C'* = {{#*lit-of L*#} |*L. is-marked L* $\wedge$ *L* $\in$ *set ?M*}
unfolding *M-K* **apply** *standard*
apply *force*
using *IntI* **by** *auto*
ultimately have *N-C-M*: *?N* $\cup$ *?C'* $\models$*ps unmark ?M*
by *auto*
have *N-M-False*: *?N* $\cup$ (λ*L*. {#*lit-of L*#}) ‘ (*set ?M*) $\models$*ps* {{#}}

using *M* ⟨*?M* ⊨*as CNot C*⟩ ⟨*C*∈*?N*⟩ **unfolding** *true-clss-clss-def true-annots-def Ball-def true-annot-def* **by** (*metis consistent-CNot-not sup.orderE sup-commute true-clss-def true-clss-singleton-lit-of-implies-incl true-clss-union true-clss-union-increase*)

have *undefined-lit F K* **using** ⟨*no-dup ?M*⟩ **unfolding** *M-K* **by** (*simp add*: *defined-lit-map*)
moreover
have *?N* ∪ *?C'* ⊨*ps* {{#}}
proof −
have *A*: *?N* ∪ *?C'* ∪ *unmark ?M* =
?N ∪ *unmark ?M*
unfolding *M-K* **by** *auto*
show *?thesis*
using *true-clss-clss-left-right*[*OF N-C-M*, *of* {{#}}] *N-M-False* **unfolding** *A* **by** *auto*
qed
have *?N* ⊨*p image-mset uminus ?C* + {#−*K*#}
unfolding *true-clss-cls-def true-clss-clss-def total-over-m-def*
proof (*intro allI impI*)
fix *I*
assume
tot: *total-over-set I* (*atms-of-ms* (*?N* ∪ {*image-mset uminus ?C*+ {#− *K*#}})) **and**
cons: *consistent-interp I* **and**
I ⊨*s ?N*
have (*K* ∈ *I* ∧ −*K* ∉ *I*) ∨ (−*K* ∈ *I* ∧ *K* ∉ *I*)
using *cons tot* **unfolding** *consistent-interp-def* **by** (*cases K*) *auto*
have *tot'*: *total-over-set I*
(*atm-of* ‘ *lit-of* ‘ (*set ?M* ∩ {*L. is-marked L* ∧ *L* ≠ *Marked K* ()}))
using *tot* **by** (*auto simp add*: *atms-of-uminus-lit-atm-of-lit-of*)
{ fix *x* :: (*'v*, *unit*, *unit*) *ann-literal*
assume
a3: *lit-of x* ∉ *I* **and**
a1: *x* ∈ *set ?M* **and**
a4: *is-marked x* **and**
a5: *x* ≠ *Marked K* ()
then have *Pos* (*atm-of* (*lit-of x*)) ∈ *I* ∨ *Neg* (*atm-of* (*lit-of x*)) ∈ *I*
using *a5 a4 tot' a1* **unfolding** *total-over-set-def atms-of-s-def* **by** *blast*
moreover have *f6*: *Neg* (*atm-of* (*lit-of x*)) = − *Pos* (*atm-of* (*lit-of x*))
by *simp*
ultimately have − *lit-of x* ∈ *I*
using *f6 a3* **by** (*metis* (*no-types*) *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set literal.sel*(*1*))
} **note** *H* = *this*

have ¬*I* ⊨*s ?C'*
using ⟨*?N* ∪ *?C'* ⊨*ps* {{#}}⟩ *tot cons* ⟨*I* ⊨*s ?N*⟩
unfolding *true-clss-clss-def total-over-m-def*
by (*simp add*: *atms-of-uminus-lit-atm-of-lit-of atms-of-ms-single-image-atm-of-lit-of*)
then show *I* ⊨ *image-mset uminus ?C* + {#− *K*#}
unfolding *true-clss-def true-cls-def Bex-mset-def*
using ⟨(*K* ∈ *I* ∧ −*K* ∉ *I*) ∨ (−*K* ∈ *I* ∧ *K* ∉ *I*)⟩
by (*auto dest*!: *H*)
qed
moreover have *F* ⊨*as CNot* (*image-mset uminus ?C*)
using *nm* **unfolding** *true-annots-def CNot-def M-K* **by** (*auto simp add*: *lits-of-def*)
ultimately have *False*
using *bj-merge-can-jump*[*of S F' K F C* −*K*

image-mset uminus (*image-mset lit-of* {# *L* :# *mset ?M*. *is-marked L* ∧ *L* ≠ *Marked K* ()#})]
⟨*C*∈*?N*⟩ *n-s* ⟨*?M* ⊨*as CNot C*⟩ *bj-backjump inv* **unfolding** *M-K*
by (*auto simp*: $cdcl_{NOT}$*-merged-bj-learn.simps*)
then show *?thesis* **by** *fast*
qed *auto*
qed

lemma *full-*$cdcl_{NOT}$*-merged-bj-learn-final-state*:
fixes *A* :: *'v literal multiset set* **and** *S T* :: *'st*
assumes
full: *full* $cdcl_{NOT}$*-merged-bj-learn S T* **and**
atms-S: *atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A* **and**
atms-trail: *atm-of* ' *lits-of* (*trail S*) ⊆ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
finite A **and**
inv: *inv S* **and**
decomp: *all-decomposition-implies-m* (*clauses S*) (*get-all-marked-decomposition* (*trail S*))
shows *unsatisfiable* (*set-mset* (*clauses T*))
∨ (*trail T* ⊨*asm clauses T* ∧ *satisfiable* (*set-mset* (*clauses T*)))
proof −
have *st*: $cdcl_{NOT}$*-merged-bj-learn*** *S T* **and** *n-s*: *no-step* $cdcl_{NOT}$*-merged-bj-learn T*
using *full* **unfolding** *full-def* **by** *blast+*
then have *st*: $cdcl_{NOT}{}^{**}$ *S T*
using *inv rtranclp-*$cdcl_{NOT}$*-merged-bj-learn-is-rtranclp-*$cdcl_{NOT}$*-and-inv n-d* **by** *auto*
have *atms-of-msu* (*clauses T*) ⊆ *atms-of-ms A* **and** *atm-of* ' *lits-of* (*trail T*) ⊆ *atms-of-ms A*
using $cdcl_{NOT}$*.rtranclp-*$cdcl_{NOT}$*-trail-clauses-bound*[*OF st inv n-d atms-S atms-trail*] **by** *blast+*
moreover have *no-dup* (*trail T*)
using $cdcl_{NOT}$*.rtranclp-*$cdcl_{NOT}$*-no-dup inv n-d st* **by** *blast*
moreover have *inv T*
using $cdcl_{NOT}$*.rtranclp-*$cdcl_{NOT}$*-inv inv st* **by** *blast*
moreover have *all-decomposition-implies-m* (*clauses T*) (*get-all-marked-decomposition* (*trail T*))
using $cdcl_{NOT}$*.rtranclp-*$cdcl_{NOT}$*-all-decomposition-implies inv st decomp n-d* **by** *blast*
ultimately show *?thesis*
using $cdcl_{NOT}$*-merged-bj-learn-final-state*[*of T A*] ⟨*finite A*⟩ *n-s* **by** *fast*
qed

end

2.8.1 Instantiations

locale $cdcl_{NOT}$*-with-backtrack-and-restarts* =
conflict-driven-clause-learning-learning-before-backjump-only-distinct-learnt trail clauses
prepend-trail tl-trail add-cls$_{NOT}$ *remove-cls*$_{NOT}$ *propagate-conds inv backjump-conds*
learn-restrictions forget-restrictions
for
trail :: *'st* ⇒ (*'v*, *unit*, *unit*) *ann-literals* **and**
clauses :: *'st* ⇒ *'v clauses* **and**
prepend-trail :: (*'v*, *unit*, *unit*) *ann-literal* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒ *'st* **and**
add-cls$_{NOT}$ *remove-cls*$_{NOT}$:: *'v clause* ⇒ *'st* ⇒ *'st* **and**
propagate-conds :: (*'v*, *unit*, *unit*) *ann-literal* ⇒ *'st* ⇒ *bool* **and**
inv :: *'st* ⇒ *bool* **and**
backjump-conds :: *'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool* **and**
learn-restrictions forget-restrictions :: *'v clause* ⇒ *'st* ⇒ *bool*
+
fixes *f* :: *nat* ⇒ *nat*

assumes
unbounded: *unbounded f* **and** *f-ge-1*: $\bigwedge n.\ n \geq 1 \Longrightarrow f\ n \geq 1$ **and**
inv-restart: $\bigwedge S\ T.$ *inv S* $\Longrightarrow T \sim$ *reduce-trail-to*$_{NOT}$ ([]::'*a list*) $S \Longrightarrow$ *inv T*
begin

lemma *bound-inv-inv*:
assumes
inv S **and**
n-d: *no-dup* (*trail S*) **and**
atms-clss-S-A: *atms-of-msu* (*clauses S*) $\subseteq$ *atms-of-ms A* **and**
atms-trail-S-A: *atm-of* ‘ *lits-of* (*trail S*) $\subseteq$ *atms-of-ms A* **and**
finite A **and**
cdcl$_{NOT}$: *cdcl*$_{NOT}$ *S T*
shows
atms-of-msu (*clauses T*) $\subseteq$ *atms-of-ms A* **and**
atm-of ‘ *lits-of* (*trail T*) $\subseteq$ *atms-of-ms A* **and**
finite A
proof −
have *cdcl*$_{NOT}$ *S T*
using ‹*inv S*› *cdcl*$_{NOT}$ **by** *linarith*
then have *atms-of-msu* (*clauses T*) $\subseteq$ *atms-of-msu* (*clauses S*) $\cup$ *atm-of* ‘ *lits-of* (*trail S*)
using ‹*inv S*›
by (*meson conflict-driven-clause-learning-ops.cdcl*$_{NOT}$*-atms-of-ms-clauses-decreasing*
conflict-driven-clause-learning-ops-axioms n-d)
then show *atms-of-msu* (*clauses T*) $\subseteq$ *atms-of-ms A*
using *atms-clss-S-A atms-trail-S-A* **by** *blast*
next
show *atm-of* ‘ *lits-of* (*trail T*) $\subseteq$ *atms-of-ms A*
by (*meson* ‹*inv S*› *atms-clss-S-A atms-trail-S-A cdcl*$_{NOT}$ *cdcl*$_{NOT}$*-atms-in-trail-in-set n-d*)
next
show *finite A*
using ‹*finite A*› **by** *simp*
qed

sublocale *cdcl*$_{NOT}$*-increasing-restarts-ops* $\lambda S\ T.\ T \sim$ *reduce-trail-to*$_{NOT}$ ([]::'*a list*) *S cdcl*$_{NOT}$ *f*
$\lambda A\ S.$ *atms-of-msu* (*clauses S*) $\subseteq$ *atms-of-ms A* $\wedge$ *atm-of* ‘ *lits-of* (*trail S*) $\subseteq$ *atms-of-ms A* $\wedge$
finite A
$\mu_{CDCL}{}'$ $\lambda S.$ *inv S* $\wedge$ *no-dup* (*trail S*)
$\mu_{CDCL}{}'$*-bound*
apply *unfold-locales*
apply (*simp add*: *unbounded*)
using *f-ge-1* **apply** *force*
using *bound-inv-inv* **apply** *meson*
apply (*rule cdcl*$_{NOT}$*-decreasing-measure'*; *simp*)
apply (*rule rtranclp-cdcl*$_{NOT}$*-*$\mu_{CDCL}{}'$*-bound*; *simp*)
apply (*rule rtranclp-*$\mu_{CDCL}{}'$*-bound-decreasing*; *simp*)
apply *auto*[]
apply *auto*[]
using *cdcl*$_{NOT}$*-inv cdcl*$_{NOT}$*-no-dup* **apply** *blast*
using *inv-restart* **apply** *auto*[]
done

abbreviation *cdcl*$_{NOT}$*-l* **where**
cdcl$_{NOT}$*-l* $\equiv$
conflict-driven-clause-learning-ops.cdcl$_{NOT}$ *trail clauses prepend-trail tl-trail add-cls*$_{NOT}$

```
  remove-cls_NOT propagate-conds (λ- - - S T. backjump S T)
  (λC S. distinct-mset C ∧ ¬ tautology C ∧ learn-restrictions C S
    ∧ (∃F K F' C' L. trail S = F' @ Marked K () # F ∧ C = C' + {#L#}
       ∧ F ⊨as CNot C' ∧ C' + {#L#} ∉# clauses S))
  (λC S. ¬ (∃F' F K L. trail S = F' @ Marked K () # F ∧ F ⊨as CNot (C − {#L#}))
  ∧ forget-restrictions C S)

lemma cdcl_NOT-with-restart-μ_CDCL'-le-μ_CDCL'-bound:
  assumes
    cdcl_NOT: cdcl_NOT-restart (T, a) (V, b) and
    cdcl_NOT-inv:
      inv T
      no-dup (trail T) and
    bound-inv:
      atms-of-msu (clauses T) ⊆ atms-of-ms A
      atm-of ` lits-of (trail T) ⊆ atms-of-ms A
      finite A
  shows μ_CDCL' A V ≤ μ_CDCL'-bound A T
  using cdcl_NOT-inv bound-inv
proof (induction rule: cdcl_NOT-with-restart-induct[OF cdcl_NOT])
  case (1 m S T n U) note U = this(3)
  show ?case
    apply (rule rtranclp-cdcl_NOT-μ_CDCL'-bound-reduce-trail-to_NOT[of S T])
       using ‹(cdcl_NOT ^^ m) S T› apply (fastforce dest!: relpowp-imp-rtranclp)
      using 1 by auto
next
  case (2 S T n) note full = this(2)
  show ?case
    apply (rule rtranclp-cdcl_NOT-μ_CDCL'-bound)
    using full 2 unfolding full1-def by force+
qed

lemma cdcl_NOT-with-restart-μ_CDCL'-bound-le-μ_CDCL'-bound:
  assumes
    cdcl_NOT: cdcl_NOT-restart (T, a) (V, b) and
    cdcl_NOT-inv:
      inv T
      no-dup (trail T) and
    bound-inv:
      atms-of-msu (clauses T) ⊆ atms-of-ms A
      atm-of ` lits-of (trail T) ⊆ atms-of-ms A
      finite A
  shows μ_CDCL'-bound A V ≤ μ_CDCL'-bound A T
  using cdcl_NOT-inv bound-inv
proof (induction rule: cdcl_NOT-with-restart-induct[OF cdcl_NOT])
  case (1 m S T n U) note U = this(3)
  have μ_CDCL'-bound A T ≤ μ_CDCL'-bound A S
     apply (rule rtranclp-μ_CDCL'-bound-decreasing)
       using ‹(cdcl_NOT ^^ m) S T› apply (fastforce dest: relpowp-imp-rtranclp)
      using 1 by auto
  then show ?case using U unfolding μ_CDCL'-bound-def by auto
next
  case (2 S T n) note full = this(2)
  show ?case
    apply (rule rtranclp-μ_CDCL'-bound-decreasing)
```

using *full 2* **unfolding** *full1-def* **by** *force+*
qed

sublocale $cdcl_{NOT}$*-increasing-restarts - - - - - - f*
$\lambda S\ T.\ T \sim$ *reduce-trail-to*$_{NOT}$ $([]::'a\ list)\ S$
$\lambda A\ S.$ *atms-of-msu* (*clauses* S) $\subseteq$ *atms-of-ms* A
$\wedge$ *atm-of* ‘ *lits-of* (*trail* S) $\subseteq$ *atms-of-ms* A $\wedge$ *finite* A
$\mu_{CDCL}{}'$ $cdcl_{NOT}$
$\lambda S.$ *inv* S $\wedge$ *no-dup* (*trail* S)
$\mu_{CDCL}{}'$*-bound*
apply *unfold-locales*
using $cdcl_{NOT}$*-with-restart-*$\mu_{CDCL}{}'$*-le-*$\mu_{CDCL}{}'$*-bound* **apply** *simp*
using $cdcl_{NOT}$*-with-restart-*$\mu_{CDCL}{}'$*-bound-le-*$\mu_{CDCL}{}'$*-bound* **apply** *simp*
done

lemma $cdcl_{NOT}$*-restart-all-decomposition-implies*:
assumes $cdcl_{NOT}$*-restart* S T **and**
inv (*fst* S) **and**
no-dup (*trail* (*fst* S))
all-decomposition-implies-m (*clauses* (*fst* S)) (*get-all-marked-decomposition* (*trail* (*fst* S)))
shows
all-decomposition-implies-m (*clauses* (*fst* T)) (*get-all-marked-decomposition* (*trail* (*fst* T)))
using *assms* **apply** (*induction*)
using *rtranclp-*$cdcl_{NOT}$*-all-decomposition-implies* **by** (*auto dest*!: *tranclp-into-rtranclp*
simp: *full1-def*)

lemma *rtranclp-*$cdcl_{NOT}$*-restart-all-decomposition-implies*:
assumes $cdcl_{NOT}$*-restart*** S T **and**
inv: *inv* (*fst* S) **and**
n-d: *no-dup* (*trail* (*fst* S)) **and**
decomp:
all-decomposition-implies-m (*clauses* (*fst* S)) (*get-all-marked-decomposition* (*trail* (*fst* S)))
shows
all-decomposition-implies-m (*clauses* (*fst* T)) (*get-all-marked-decomposition* (*trail* (*fst* T)))
using *assms*(*1*)
proof (*induction rule*: *rtranclp-induct*)
case *base*
then show *?case* **using** *decomp* **by** *simp*
next
case (*step* T u) **note** $st = this(1)$ **and** $r = this(2)$ **and** $IH = this(3)$
have *inv* (*fst* T)
using *rtranclp-*$cdcl_{NOT}$*-with-restart-*$cdcl_{NOT}$*-inv*[*OF st*] *inv n-d* **by** *blast*
moreover have *no-dup* (*trail* (*fst* T))
using *rtranclp-*$cdcl_{NOT}$*-with-restart-*$cdcl_{NOT}$*-inv*[*OF st*] *inv n-d* **by** *blast*
ultimately show *?case*
using $cdcl_{NOT}$*-restart-all-decomposition-implies* *r IH n-d* **by** *fast*
qed

lemma $cdcl_{NOT}$*-restart-sat-ext-iff*:
assumes
st: $cdcl_{NOT}$*-restart* S T **and**
n-d: *no-dup* (*trail* (*fst* S)) **and**
inv: *inv* (*fst* S)
shows I $\models$*sextm clauses* (*fst* S) $\longleftrightarrow$ I $\models$*sextm clauses*(*fst* T)
using *assms*

proof (*induction*)
  case (*restart-step m S T n U*)
  then show *?case*
    using *rtranclp-cdcl$_{NOT}$-bj-sat-ext-iff n-d* **by** (*fastforce dest*!: *relpowp-imp-rtranclp*)
next
  case *restart-full*
  then show *?case* **using** *rtranclp-cdcl$_{NOT}$-bj-sat-ext-iff* **unfolding** *full1-def*
  by (*fastforce dest*!: *tranclp-into-rtranclp*)
qed

lemma *rtranclp-cdcl$_{NOT}$-restart-sat-ext-iff*:
  assumes
    st: *cdcl$_{NOT}$-restart** S T* **and**
    n-d: *no-dup* (*trail* (*fst S*)) **and**
    inv: *inv* (*fst S*)
  shows *I* $\models$*sextm clauses* (*fst S*) $\longleftrightarrow$ *I* $\models$*sextm clauses*(*fst T*)
  using *st*
proof (*induction*)
  case *base*
  then show *?case* **by** *simp*
next
  case (*step T U*) **note** *st* = *this*(*1*) **and** *r* = *this*(*2*) **and** *IH* = *this*(*3*)
  have *inv* (*fst T*)
    using *rtranclp-cdcl$_{NOT}$-with-restart-cdcl$_{NOT}$-inv*[*OF st*] *inv n-d* **by** *blast+*
  moreover have *no-dup* (*trail* (*fst T*))
    using *rtranclp-cdcl$_{NOT}$-with-restart-cdcl$_{NOT}$-inv rtranclp-cdcl$_{NOT}$-no-dup st inv n-d* **by** *blast*
  ultimately show *?case*
    using *cdcl$_{NOT}$-restart-sat-ext-iff*[*OF r*] *IH* **by** *blast*
qed

theorem *full-cdcl$_{NOT}$-restart-backjump-final-state*:
  fixes *A* :: *'v literal multiset set* **and** *S T* :: *'st*
  assumes
    full: *full cdcl$_{NOT}$-restart* (*S*, *n*) (*T*, *m*) **and**
    atms-S: *atms-of-msu* (*clauses S*) $\subseteq$ *atms-of-ms A* **and**
    atms-trail: *atm-of* ' *lits-of* (*trail S*) $\subseteq$ *atms-of-ms A* **and**
    n-d: *no-dup* (*trail S*) **and**
    fin-A[*simp*]: *finite A* **and**
    inv: *inv S* **and**
    decomp: *all-decomposition-implies-m* (*clauses S*) (*get-all-marked-decomposition* (*trail S*))
  shows *unsatisfiable* (*set-mset* (*clauses S*))
    $\lor$ (*lits-of* (*trail T*) $\models$*sextm clauses S* $\land$ *satisfiable* (*set-mset* (*clauses S*)))
proof −
  have *st*: *cdcl$_{NOT}$-restart*** (*S*, *n*) (*T*, *m*) **and**
    n-s: *no-step cdcl$_{NOT}$-restart* (*T*, *m*)
    using *full* **unfolding** *full-def* **by** *fast+*
  have *binv-T*: *atms-of-msu* (*clauses T*) $\subseteq$ *atms-of-ms A atm-of* ' *lits-of* (*trail T*) $\subseteq$ *atms-of-ms A*
    using *rtranclp-cdcl$_{NOT}$-with-restart-bound-inv*[*OF st*, *of A*] *inv n-d atms-S atms-trail*
    by *auto*
  moreover have *inv-T*: *no-dup* (*trail T*) *inv T*
    using *rtranclp-cdcl$_{NOT}$-with-restart-cdcl$_{NOT}$-inv*[*OF st*] *inv n-d* **by** *auto*
  moreover have *all-decomposition-implies-m* (*clauses T*) (*get-all-marked-decomposition* (*trail T*))
    using *rtranclp-cdcl$_{NOT}$-restart-all-decomposition-implies*[*OF st*] *inv n-d*
    decomp **by** *auto*
  ultimately have *T*: *unsatisfiable* (*set-mset* (*clauses T*))

$\lor$ (*trail T* $\models$*asm clauses T* $\land$ *satisfiable* (*set-mset* (*clauses T*)))
using *no-step-*$cdcl_{NOT}$*-restart-no-step-*$cdcl_{NOT}$[*of* (*T*, *m*) *A*] *n-s*
$cdcl_{NOT}$*-final-state*[*of T A*] **unfolding** $cdcl_{NOT}$*-NOT-all-inv-def* **by** *auto*
have *eq-sat-S-T*: $\bigwedge I.$ *I* $\models$*sextm clauses S* $\longleftrightarrow$ *I* $\models$*sextm clauses T*
using *rtranclp-*$cdcl_{NOT}$*-restart-sat-ext-iff*[*OF st*] *inv n-d atms-S*
atms-trail **by** *auto*
have *cons-T*: *consistent-interp* (*lits-of* (*trail T*))
using *inv-T*(*1*) *distinctconsistent-interp* **by** *blast*
consider
(*unsat*) *unsatisfiable* (*set-mset* (*clauses T*))
| (*sat*) *trail T* $\models$*asm clauses T* **and** *satisfiable* (*set-mset* (*clauses T*))
using *T* **by** *blast*
then show *?thesis*
proof *cases*
case *unsat*
then have *unsatisfiable* (*set-mset* (*clauses S*))
using *eq-sat-S-T consistent-true-clss-ext-satisfiable true-clss-imp-true-cls-ext*
unfolding *satisfiable-def* **by** *blast*
then show *?thesis* **by** *fast*
next
case *sat*
then have *lits-of* (*trail T*) $\models$*sextm clauses S*
using *rtranclp-*$cdcl_{NOT}$*-restart-sat-ext-iff*[*OF st*] *inv n-d atms-S*
atms-trail **by** (*auto simp*: *true-clss-imp-true-cls-ext true-annots-true-cls*)
moreover then have *satisfiable* (*set-mset* (*clauses S*))
using *cons-T consistent-true-clss-ext-satisfiable* **by** *blast*
ultimately show *?thesis* **by** *blast*
qed
qed
end — end of $cdcl_{NOT}$*-with-backtrack-and-restarts* locale

locale *most-general-*$cdcl_{NOT}$ =
dpll-state trail clauses prepend-trail tl-trail add-cls_{NOT} *remove-*cls_{NOT} +
propagate-ops trail clauses prepend-trail tl-trail add-cls_{NOT} *remove-*cls_{NOT} *propagate-conds* +
backjumping-ops trail clauses prepend-trail tl-trail add-cls_{NOT} *remove-*cls_{NOT} λ- - - - -. *True*
for
trail :: *'st* $\Rightarrow$ (*'v*, *unit*, *unit*) *ann-literals* **and**
clauses :: *'st* $\Rightarrow$ *'v clauses* **and**
prepend-trail :: (*'v*, *unit*, *unit*) *ann-literal* $\Rightarrow$ *'st* $\Rightarrow$ *'st* **and**
tl-trail :: *'st* $\Rightarrow$ *'st* **and**
add-cls_{NOT} *remove-*cls_{NOT}:: *'v clause* $\Rightarrow$ *'st* $\Rightarrow$ *'st* **and**
propagate-conds :: (*'v*, *unit*, *unit*) *ann-literal* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **and**
inv :: *'st* $\Rightarrow$ *bool*
begin
lemma *backjump-bj-can-jump*:
assumes
tr-S: *trail S* = *F'* @ *Marked K* () # *F* **and**
C: *C* $\in$# *clauses S* **and**
tr-S-C: *trail S* $\models$*as CNot C* **and**
undef: *undefined-lit F L* **and**
atm-L: *atm-of L* $\in$ *atms-of-msu* (*clauses S*) $\cup$ *atm-of* ‘ (*lits-of* (*F'* @ *Marked K* () # *F*)) **and**
cls-S-C': *clauses S* $\models$*pm C'* + {#*L*#} **and**
F-C': *F* $\models$*as CNot C'*
shows $\neg$*no-step backjump S*
using *backjump.intros*[*OF tr-S - C tr-S-C undef - cls-S-C' F-C'*,

of prepend-trail (*Propagated L* -) (*reduce-trail-to*$_{NOT}$ *F S*)] *atm-L* **unfolding** *tr-S*
by (*auto simp*: *state-eq*$_{NOT}$*-def simp del*: *state-simp*$_{NOT}$)

sublocale *dpll-with-backjumping-ops* - - - - - - - *inv* λ- - - - -. *True*
using *backjump-bj-can-jump* **by** *unfold-locales auto*
end

The restart does only reset the trail, contrary to Weidenbach's version. But there is a forget rule.

locale *cdcl*$_{NOT}$*-merge-bj-learn-with-backtrack-restarts* =
cdcl$_{NOT}$*-merge-bj-learn trail clauses prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$
propagate-conds inv forget-conds
$\lambda C\ C'\ L'\ S$. *distinct-mset* (C' + {#L'#}) $\wedge$ *backjump-l-cond* $C\ C'\ L'\ S$
for
trail :: *'st* $\Rightarrow$ (*'v*, *unit*, *unit*) *ann-literals* **and**
clauses :: *'st* $\Rightarrow$ *'v clauses* **and**
prepend-trail :: (*'v*, *unit*, *unit*) *ann-literal* $\Rightarrow$ *'st* $\Rightarrow$ *'st* **and**
tl-trail :: *'st* $\Rightarrow$ *'st* **and**
add-cls$_{NOT}$ *remove-cls*$_{NOT}$:: *'v clause* $\Rightarrow$ *'st* $\Rightarrow$ *'st* **and**
propagate-conds :: (*'v*, *unit*, *unit*) *ann-literal* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **and**
inv :: *'st* $\Rightarrow$ *bool* **and**
forget-conds :: *'v clause* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **and**
backjump-l-cond :: *'v clause* $\Rightarrow$ *'v clause* $\Rightarrow$ *'v literal* $\Rightarrow$ *'st* $\Rightarrow$ *bool*
+
fixes *f* :: *nat* $\Rightarrow$ *nat*
assumes
unbounded: *unbounded f* **and** *f-ge-1*: $\bigwedge n.\ n \geq 1 \Longrightarrow f\ n \geq 1$ **and**
inv-restart:$\bigwedge S\ T$. *inv S* $\Longrightarrow$ $T \sim$ *reduce-trail-to*$_{NOT}$ [] $S \Longrightarrow$ *inv T*
begin

interpretation *cdcl*$_{NOT}$:
conflict-driven-clause-learning-ops trail clauses prepend-trail tl-trail add-cls$_{NOT}$ *remove-cls*$_{NOT}$
propagate-conds inv backjump-conds (λC -. *distinct-mset C* $\wedge$ $\neg$ *tautology C*) *forget-conds*
by *unfold-locales*

interpretation *cdcl*$_{NOT}$:
conflict-driven-clause-learning trail clauses prepend-trail tl-trail add-cls$_{NOT}$ *remove-cls*$_{NOT}$
propagate-conds inv backjump-conds (λC -. *distinct-mset C* $\wedge$ $\neg$ *tautology C*) *forget-conds*
apply *unfold-locales*
using *cdcl*$_{NOT}$*-merged-bj-learn-forget*$_{NOT}$ *cdcl-merged-inv learn-inv*
by (*auto simp add*: *cdcl*$_{NOT}$*.simps dpll-bj-inv*)

definition *not-simplified-cls* A = {#C $\in$# A. *tautology C* $\vee$ $\neg$*distinct-mset C*#}

lemma *simple-clss-or-not-simplified-cls*:
assumes *atms-of-msu* (*clauses S*) $\subseteq$ *atms-of-ms A* **and**
x $\in$# *clauses S* **and** *finite A*
shows $x \in$ *simple-clss* (*atms-of-ms A*) $\vee$ x $\in$# *not-simplified-cls* (*clauses S*)
proof −
consider
(*simpl*) $\neg$*tautology x* **and** *distinct-mset x*
| (*n-simp*) *tautology x* $\vee$ $\neg$*distinct-mset x*
by *auto*

then show *?thesis*
proof *cases*
case *simpl*
then have $x \in$ *simple-clss* (*atms-of-ms A*)
by (*meson assms atms-of-atms-of-ms-mono atms-of-ms-finite simple-clss-mono*
distinct-mset-not-tautology-implies-in-simple-clss finite-subset
mem-set-mset-iff subsetCE)
then show *?thesis* **by** *blast*
next
case *n-simp*
then have $x \in\#$ *not-simplified-cls* (*clauses S*)
using ‹$x \in\#$ *clauses S*› **unfolding** *not-simplified-cls-def* **by** *auto*
then show *?thesis* **by** *blast*
qed
qed

lemma $cdcl_{NOT}$*-merged-bj-learn-clauses-bound*:
assumes
$cdcl_{NOT}$*-merged-bj-learn S T* **and**
inv: *inv S* **and**
atms-clss: *atms-of-msu* (*clauses S*) $\subseteq$ *atms-of-ms A* **and**
atms-trail: *atm-of* ‘(*lits-of* (*trail S*)) $\subseteq$ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
fin-A[*simp*]: *finite A*
shows *set-mset* (*clauses T*) $\subseteq$ *set-mset* (*not-simplified-cls* (*clauses S*))
$\cup$ *simple-clss* (*atms-of-ms A*)
using *assms*
proof (*induction rule*: $cdcl_{NOT}$*-merged-bj-learn.induct*)
case $cdcl_{NOT}$*-merged-bj-learn-decide*$_{NOT}$
then show *?case* **using** *dpll-bj-clauses* **by** (*force dest*!: *simple-clss-or-not-simplified-cls*)
next
case $cdcl_{NOT}$*-merged-bj-learn-propagate*$_{NOT}$
then show *?case* **using** *dpll-bj-clauses* **by** (*force dest*!: *simple-clss-or-not-simplified-cls*)
next
case $cdcl_{NOT}$*-merged-bj-learn-forget*$_{NOT}$
then show *?case* **using** *clauses-remove-cls*$_{NOT}$ **unfolding** *state-eq*$_{NOT}$*-def*
by (*force elim*!: *forget*$_{NOT}$*E dest*: *simple-clss-or-not-simplified-cls*)
next
case ($cdcl_{NOT}$*-merged-bj-learn-backjump-l T*) **note** *bj* = *this*(*1*) **and** *inv* = *this*(*2*) **and**
atms-clss = *this*(*3*) **and** *atms-trail* = *this*(*4*) **and** *n-d* = *this*(*5*)

have $cdcl_{NOT}{}^{**}$ *S T*
apply (*rule rtranclp-*$cdcl_{NOT}$*-merged-bj-learn-is-rtranclp-*$cdcl_{NOT}$)
using ‹*backjump-l S T*› *inv* $cdcl_{NOT}$*-merged-bj-learn.simps n-d* **by** *blast+*
have *atm-of* ‘(*lits-of* (*trail T*)) $\subseteq$ *atms-of-ms A*
using $cdcl_{NOT}$*.rtranclp-*$cdcl_{NOT}$*-trail-clauses-bound*[*OF* ‹$cdcl_{NOT}{}^{**}$ *S T*›] *inv atms-trail atms-clss*
n-d **by** *auto*
have *atms-of-msu* (*clauses T*) $\subseteq$ *atms-of-ms A*
using $cdcl_{NOT}$*.rtranclp-*$cdcl_{NOT}$*-trail-clauses-bound*[*OF* ‹$cdcl_{NOT}{}^{**}$ *S T*› *inv n-d atms-clss atms-trail*]
by *fast*
moreover have *no-dup* (*trail T*)
using $cdcl_{NOT}$*.rtranclp-*$cdcl_{NOT}$*-no-dup*[*OF* ‹$cdcl_{NOT}{}^{**}$ *S T*› *inv n-d*] **by** *fast*

obtain F' *K F L l* C' *C* **where**
tr-S: *trail S* = F' @ *Marked K* () # *F* **and**

T: $T \sim$ *prepend-trail* (*Propagated L l*) (*reduce-trail-to*$_{NOT}$ *F* (*add-cls*$_{NOT}$ (C' + {#*L*#}) *S*)) **and**
C ∈# *clauses S* **and**
trail S ⊨*as CNot C* **and**
undef: *undefined-lit F L* **and**
atm-of L = *atm-of K* ∨ *atm-of L* ∈ *atms-of-msu* (*clauses S*)
∨ *atm-of L* ∈ *atm-of* ‘ (*lits-of* F' ∪ *lits-of F*) **and**
clauses S ⊨*pm* C' + {#*L*#} **and**
F ⊨*as CNot* C' **and**
dist: *distinct-mset* (C' + {#*L*#}) **and**
tauto: ¬ *tautology* (C' + {#*L*#}) **and**
backjump-l-cond C C' *L T*
using ‹*backjump-l S T*› **apply** (*induction rule*: *backjump-l.induct*) **by** *auto*

have *atms-of* C' ⊆ *atm-of* ‘ (*lits-of F*)
using ‹*F* ⊨*as CNot* C'› **by** (*simp add*: *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set*
atms-of-def image-subset-iff in-CNot-implies-uminus(*2*))
then have *atms-of* (C'+{#*L*#}) ⊆ *atms-of-ms A*
using *T* ‹*atm-of* ‘ *lits-of* (*trail T*) ⊆ *atms-of-ms A*› *tr-S undef n-d* **by** *auto*
then have *simple-clss* (*atms-of* (C' + {#*L*#})) ⊆ *simple-clss* (*atms-of-ms A*)
apply − **by** (*rule simple-clss-mono*) (*simp-all*)
then have C' + {#*L*#} ∈ *simple-clss* (*atms-of-ms A*)
using *distinct-mset-not-tautology-implies-in-simple-clss*[*OF dist tauto*]
by *auto*
then show *?case*
using *T inv atms-clss undef tr-S n-d*
by (*force dest*!: *simple-clss-or-not-simplified-cls*)
qed

lemma *cdcl*$_{NOT}$*-merged-bj-learn-not-simplified-decreasing*:
assumes *cdcl*$_{NOT}$*-merged-bj-learn S T*
shows (*not-simplified-cls* (*clauses T*)) ⊆# (*not-simplified-cls* (*clauses S*))
using *assms* **apply** *induction*
prefer *4*
unfolding *not-simplified-cls-def* **apply** (*auto elim*!: *backjump-lE forget*$_{NOT}$*E*)[*3*]
by (*elim backjump-lE*) *auto*

lemma *rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-not-simplified-decreasing*:
assumes *cdcl*$_{NOT}$*-merged-bj-learn*** *S T*
shows (*not-simplified-cls* (*clauses T*)) ⊆# (*not-simplified-cls* (*clauses S*))
using *assms* **apply** *induction*
apply *simp*
by (*drule cdcl*$_{NOT}$*-merged-bj-learn-not-simplified-decreasing*) *auto*

lemma *rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-clauses-bound*:
assumes
cdcl$_{NOT}$*-merged-bj-learn*** *S T* **and**
inv S **and**
atms-of-msu (*clauses S*) ⊆ *atms-of-ms A* **and**
atm-of ‘(*lits-of* (*trail S*)) ⊆ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
finite[*simp*]: *finite A*
shows *set-mset* (*clauses T*) ⊆ *set-mset* (*not-simplified-cls* (*clauses S*))
∪ *simple-clss* (*atms-of-ms A*)
using *assms*(*1*−*5*)
proof *induction*

case *base*
then show *?case* **by** (*auto dest*!: *simple-clss-or-not-simplified-cls*)
next
case (*step T U*) **note** *st* = *this*(*1*) **and** $cdcl_{NOT}$ = *this*(*2*) **and** *IH* = *this*(*3*)[*OF this*(*4*−*7*)] **and** *inv* = *this*(*4*) **and** *atms-clss-S* = *this*(*5*) **and** *atms-trail-S* = *this*(*6*) **and** *finite-cls-S* = *this*(*7*)
have *st'*: $cdcl_{NOT}{}^{**}$ *S T*
using *inv rtranclp-*$cdcl_{NOT}$*-merged-bj-learn-is-rtranclp-*$cdcl_{NOT}$*-and-inv st n-d* **by** *blast*
have *inv T*
using *inv rtranclp-*$cdcl_{NOT}$*-merged-bj-learn-inv st n-d* **by** *blast*
moreover
have *atms-of-msu* (*clauses T*) ⊆ *atms-of-ms A* **and**
atm-of ' *lits-of* (*trail T*) ⊆ *atms-of-ms A*
using $cdcl_{NOT}$*.rtranclp-*$cdcl_{NOT}$*-trail-clauses-bound*[*OF st'*] *inv atms-clss-S atms-trail-S n-d*
by *blast*+
moreover moreover have *no-dup* (*trail T*)
using $cdcl_{NOT}$*.rtranclp-*$cdcl_{NOT}$*-no-dup*[*OF* ‹$cdcl_{NOT}{}^{**}$ *S T*› *inv n-d*] **by** *fast*
ultimately have *set-mset* (*clauses U*)
⊆ *set-mset* (*not-simplified-cls* (*clauses T*)) ∪ *simple-clss* (*atms-of-ms A*)
using $cdcl_{NOT}$ *finite* $cdcl_{NOT}$*-merged-bj-learn-clauses-bound*
by (*auto intro*!: $cdcl_{NOT}$*-merged-bj-learn-clauses-bound*)
moreover have *set-mset* (*not-simplified-cls* (*clauses T*))
⊆ *set-mset* (*not-simplified-cls* (*clauses S*))
using *rtranclp-*$cdcl_{NOT}$*-merged-bj-learn-not-simplified-decreasing*[*OF st*] **by** *auto*
ultimately show *?case* **using** *IH inv atms-clss-S*
by (*auto dest*!: *simple-clss-or-not-simplified-cls*)
qed

abbreviation $\mu_{CDCL}{}'$*-bound* **where**
$\mu_{CDCL}{}'$*-bound A T* == ((*2*+*card* (*atms-of-ms A*)) ^ (*1*+*card* (*atms-of-ms A*))) ∗ *2*
+ *card* (*set-mset* (*not-simplified-cls*(*clauses T*)))
+ *3* ^ *card* (*atms-of-ms A*)

lemma *rtranclp-*$cdcl_{NOT}$*-merged-bj-learn-clauses-bound-card*:
assumes
$cdcl_{NOT}$*-merged-bj-learn*** *S T* **and**
inv S **and**
atms-of-msu (*clauses S*) ⊆ *atms-of-ms A* **and**
atm-of '(*lits-of* (*trail S*)) ⊆ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
finite: *finite A*
shows $\mu_{CDCL}{}'$*-merged A T* ≤ $\mu_{CDCL}{}'$*-bound A S*
proof −
have *set-mset* (*clauses T*) ⊆ *set-mset* (*not-simplified-cls*(*clauses S*))
∪ *simple-clss* (*atms-of-ms A*)
using *rtranclp-*$cdcl_{NOT}$*-merged-bj-learn-clauses-bound*[*OF assms*] **.**
moreover have *card* (*set-mset* (*not-simplified-cls*(*clauses S*))
∪ *simple-clss* (*atms-of-ms A*))
≤ *card* (*set-mset* (*not-simplified-cls*(*clauses S*))) + *3* ^ *card* (*atms-of-ms A*)
by (*meson Nat.le-trans atms-of-ms-finite simple-clss-card card-Un-le finite*
nat-add-left-cancel-le)
ultimately have *card* (*set-mset* (*clauses T*))
≤ *card* (*set-mset* (*not-simplified-cls*(*clauses S*))) + *3* ^ *card* (*atms-of-ms A*)
by (*meson Nat.le-trans atms-of-ms-finite simple-clss-finite card-mono*
finite-UnI finite-set-mset local.finite)
moreover have ((*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*)) − $\mu_{C}{}'$ *A T*) ∗ *2*

$\leq$ (*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*)) $*$ *2*
by *auto*
ultimately show *?thesis* **unfolding** μ_{CDCL}*'-merged-def* **by** *auto*
qed

sublocale *cdcl*$_{NOT}$*-increasing-restarts-ops* $\lambda S\ T.\ T \sim$ *reduce-trail-to*$_{NOT}$ ([]::'*a list*) *S*
cdcl$_{NOT}$*-merged-bj-learn f*
$\lambda A\ S.$ *atms-of-msu* (*clauses S*) $\subseteq$ *atms-of-ms A*
$\wedge$ *atm-of* ' *lits-of* (*trail S*) $\subseteq$ *atms-of-ms A* $\wedge$ *finite A*
μ_{CDCL}*'-merged*
$\lambda S.$ *inv S* $\wedge$ *no-dup* (*trail S*)
μ_{CDCL}*'-bound*
apply *unfold-locales*
using *unbounded* **apply** *simp*
using *f-ge-1* **apply** *force*
apply (*blast dest*!: *cdcl*$_{NOT}$*-merged-bj-learn-is-tranclp-cdcl*$_{NOT}$ *tranclp-into-rtranclp*
cdcl$_{NOT}$*.rtranclp-cdcl*$_{NOT}$*-trail-clauses-bound*)
apply (*simp add*: *cdcl*$_{NOT}$*-decreasing-measure'*)
using *rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-clauses-bound-card* **apply** *blast*
apply (*drule rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-not-simplified-decreasing*)
apply (*auto dest*!: *simp*: *card-mono set-mset-mono*)[]
apply *simp*
apply *auto*[]
using *cdcl*$_{NOT}$*-merged-bj-learn-no-dup-inv cdcl-merged-inv* **apply** *blast*
apply (*auto simp*: *inv-restart*)[]
done

lemma *cdcl*$_{NOT}$*-restart-*μ_{CDCL}*'-merged-le-*μ_{CDCL}*'-bound*:
assumes
cdcl$_{NOT}$*-restart T V*
inv (*fst T*) **and**
no-dup (*trail* (*fst T*)) **and**
atms-of-msu (*clauses* (*fst T*)) $\subseteq$ *atms-of-ms A* **and**
atm-of ' *lits-of* (*trail* (*fst T*)) $\subseteq$ *atms-of-ms A* **and**
finite A
shows μ_{CDCL}*'-merged A* (*fst V*) $\leq$ μ_{CDCL}*'-bound A* (*fst T*)
using *assms*
proof *induction*
case (*restart-full S T n*)
show *?case*
unfolding *fst-conv*
apply (*rule rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-clauses-bound-card*)
using *restart-full* **unfolding** *full1-def* **by** (*force dest*!: *tranclp-into-rtranclp*)+
next
case (*restart-step m S T n U*) **note** *st* = *this*(*1*) **and** *U* = *this*(*3*) **and** *inv* = *this*(*4*) **and**
n-d = *this*(*5*) **and** *atms-clss* = *this*(*6*) **and** *atms-trail* = *this*(*7*) **and** *finite* = *this*(*8*)
then have *st'*: *cdcl*$_{NOT}$*-merged-bj-learn*** *S T*
by (*blast dest*: *relpowp-imp-rtranclp*)
then have *st''*: *cdcl*$_{NOT}$** *S T*
using *inv n-d* **apply** − **by** (*rule rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-is-rtranclp-cdcl*$_{NOT}$) *auto*
have *inv T*
apply (*rule rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-inv*)
using *inv st' n-d* **by** *auto*
then have *inv U*
using *U* **by** (*auto simp*: *inv-restart*)

have *atms-of-msu* (*clauses* *T*) ⊆ *atms-of-ms* *A*
using $cdcl_{NOT}$*.rtranclp-*$cdcl_{NOT}$*-trail-clauses-bound*[*OF* *st'*] *inv* *atms-clss* *atms-trail* *n-d*
by *simp*
then have *atms-of-msu* (*clauses* *U*) ⊆ *atms-of-ms* *A*
using *U* **by** *simp*
have *not-simplified-cls* (*clauses* *U*) ⊆# *not-simplified-cls* (*clauses* *T*)
using ‹*U* ∼ *reduce-trail-to*$_{NOT}$ [] *T*› **by** *auto*
moreover have *not-simplified-cls* (*clauses* *T*) ⊆# *not-simplified-cls* (*clauses* *S*)
apply (*rule* *rtranclp-*$cdcl_{NOT}$*-merged-bj-learn-not-simplified-decreasing*)
using ‹($cdcl_{NOT}$*-merged-bj-learn* ^^ *m*) *S* *T*› **by** (*auto* *dest*!: *relpowp-imp-rtranclp*)
ultimately have *U-S*: *not-simplified-cls* (*clauses* *U*) ⊆# *not-simplified-cls* (*clauses* *S*)
by *auto*

have (*set-mset* (*clauses* *U*))
⊆ *set-mset* (*not-simplified-cls* (*clauses* *U*)) ∪ *simple-clss* (*atms-of-ms* *A*)
apply (*rule* *rtranclp-*$cdcl_{NOT}$*-merged-bj-learn-clauses-bound*)
apply *simp*
using ‹*inv* *U*› **apply** *simp*
using ‹*atms-of-msu* (*clauses* *U*) ⊆ *atms-of-ms* *A*› **apply** *simp*
using *U* **apply** *simp*
using *U* **apply** *simp*
using *finite* **apply** *simp*
done
then have *f1*: *card* (*set-mset* (*clauses* *U*)) ≤ *card* (*set-mset* (*not-simplified-cls* (*clauses* *U*))
∪ *simple-clss* (*atms-of-ms* *A*))
by (*simp* *add*: *simple-clss-finite* *card-mono* *local.finite*)

moreover have *set-mset* (*not-simplified-cls* (*clauses* *U*)) ∪ *simple-clss* (*atms-of-ms* *A*)
⊆ *set-mset* (*not-simplified-cls* (*clauses* *S*)) ∪ *simple-clss* (*atms-of-ms* *A*)
using *U-S* **by** *auto*
then have *f2*:
card (*set-mset* (*not-simplified-cls* (*clauses* *U*)) ∪ *simple-clss* (*atms-of-ms* *A*))
≤ *card* (*set-mset* (*not-simplified-cls* (*clauses* *S*)) ∪ *simple-clss* (*atms-of-ms* *A*))
by (*simp* *add*: *simple-clss-finite* *card-mono* *local.finite*)

moreover have *card* (*set-mset* (*not-simplified-cls* (*clauses* *S*))
∪ *simple-clss* (*atms-of-ms* *A*))
≤ *card* (*set-mset* (*not-simplified-cls* (*clauses* *S*))) + *card* (*simple-clss* (*atms-of-ms* *A*))
using *card-Un-le* **by** *blast*
moreover have *card* (*simple-clss* (*atms-of-ms* *A*)) ≤ *3* ^ *card* (*atms-of-ms* *A*)
using *atms-of-ms-finite* *simple-clss-card* *local.finite* **by** *blast*
ultimately have *card* (*set-mset* (*clauses* *U*))
≤ *card* (*set-mset* (*not-simplified-cls* (*clauses* *S*))) + *3* ^ *card* (*atms-of-ms* *A*)
by *linarith*
then show *?case* **unfolding** μ_{CDCL}*'-merged-def* **by** *auto*
qed

lemma $cdcl_{NOT}$*-restart-*μ_{CDCL}*'-bound-le-*μ_{CDCL}*'-bound*:
assumes
$cdcl_{NOT}$*-restart* *T* *V* **and**
no-dup (*trail* (*fst* *T*)) **and**
inv (*fst* *T*) **and**
fin: *finite* *A*
shows μ_{CDCL}*'-bound* *A* (*fst* *V*) ≤ μ_{CDCL}*'-bound* *A* (*fst* *T*)
using *assms*(*1*−*3*)

proof *induction*
case (*restart-full S T n*)
have *not-simplified-cls* (*clauses T*) $\subseteq\#$ *not-simplified-cls* (*clauses S*)
apply (*rule rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-not-simplified-decreasing*)
using ‹*full1 cdcl*$_{NOT}$*-merged-bj-learn S T*› **unfolding** *full1-def*
by (*auto dest*: *tranclp-into-rtranclp*)
then show *?case* **by** (*auto simp*: *card-mono set-mset-mono*)
next
case (*restart-step m S T n U*) **note** *st* = *this*(*1*) **and** *U* = *this*(*3*) **and** *n-d* =*this*(*4*) **and** *inv* = *this*(*5*)
then have *st'*: *cdcl*$_{NOT}$*-merged-bj-learn*** *S T*
by (*blast dest*: *relpowp-imp-rtranclp*)
then have *st''*: *cdcl*$_{NOT}$** *S T*
using *inv n-d* **apply** − **by** (*rule rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-is-rtranclp-cdcl*$_{NOT}$) *auto*
have *inv T*
apply (*rule rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-inv*)
using *inv st' n-d* **by** *auto*
then have *inv U*
using *U* **by** (*auto simp*: *inv-restart*)
have *not-simplified-cls* (*clauses U*) $\subseteq\#$ *not-simplified-cls* (*clauses T*)
using ‹*U* ∼ *reduce-trail-to*$_{NOT}$ [] *T*› **by** *auto*
moreover have *not-simplified-cls* (*clauses T*) $\subseteq\#$ *not-simplified-cls* (*clauses S*)
apply (*rule rtranclp-cdcl*$_{NOT}$*-merged-bj-learn-not-simplified-decreasing*)
using ‹(*cdcl*$_{NOT}$*-merged-bj-learn* ^^ *m*) *S T*› **by** (*auto dest*!: *relpowp-imp-rtranclp*)
ultimately have *U-S*: *not-simplified-cls* (*clauses U*) $\subseteq\#$ *not-simplified-cls* (*clauses S*)
by *auto*
then show *?case* **by** (*auto simp*: *card-mono set-mset-mono*)
qed

sublocale *cdcl*$_{NOT}$*-increasing-restarts* - - - - - - *f* λ*S T*. *T* ∼ *reduce-trail-to*$_{NOT}$ ([]::'*a list*) *S*
λ*A S*. *atms-of-msu* (*clauses S*) ⊆ *atms-of-ms A*
∧ *atm-of* ‘ *lits-of* (*trail S*) ⊆ *atms-of-ms A* ∧ *finite A*
μ_{CDCL}'*-merged cdcl*$_{NOT}$*-merged-bj-learn*
λ*S*. *inv S* ∧ *no-dup* (*trail S*)
λ*A T*. ((*2*+*card* (*atms-of-ms A*)) ^ (*1*+*card* (*atms-of-ms A*))) ∗ *2*
+ *card* (*set-mset* (*not-simplified-cls*(*clauses T*)))
+ *3* ^ *card* (*atms-of-ms A*)
apply *unfold-locales*
using *cdcl*$_{NOT}$*-restart-*μ_{CDCL}'*-merged-le-*μ_{CDCL}'*-bound* **apply** *force*
using *cdcl*$_{NOT}$*-restart-*μ_{CDCL}'*-bound-le-*μ_{CDCL}'*-bound* **by** *fastforce*

lemma *cdcl*$_{NOT}$*-restart-eq-sat-iff*:
assumes
cdcl$_{NOT}$*-restart S T* **and**
no-dup (*trail* (*fst S*))
inv (*fst S*)
shows *I*⊨*sextm clauses* (*fst S*) ⟷ *I* ⊨*sextm clauses* (*fst T*)
using *assms*
proof (*induction rule*: *cdcl*$_{NOT}$*-restart.induct*)
case (*restart-full S T n*)
then have *cdcl*$_{NOT}$*-merged-bj-learn*** *S T*
by (*simp add*: *tranclp-into-rtranclp full1-def*)
then show *?case*
using *cdcl*$_{NOT}$*.rtranclp-cdcl*$_{NOT}$*-bj-sat-ext-iff restart-full.prems*(*1*,*2*)

rtranclp-cdcl$_{NOT}$-merged-bj-learn-is-rtranclp-cdcl$_{NOT}$ **by** *auto*
next
case (*restart-step m S T n U*)
then have *cdcl$_{NOT}$-merged-bj-learn** S T*
by (*auto simp*: *tranclp-into-rtranclp full1-def dest*!: *relpowp-imp-rtranclp*)
then have $I \models$*sextm clauses S* $\longleftrightarrow I \models$*sextm clauses T*
using *cdcl$_{NOT}$.rtranclp-cdcl$_{NOT}$-bj-sat-ext-iff restart-step.prems*(*1*,*2*)
rtranclp-cdcl$_{NOT}$-merged-bj-learn-is-rtranclp-cdcl$_{NOT}$ **by** *auto*
moreover have $I \models$*sextm clauses T* $\longleftrightarrow I \models$*sextm clauses U*
using *restart-step.hyps*(*3*) **by** *auto*
ultimately show *?case* **by** *auto*
qed

lemma *rtranclp-cdcl$_{NOT}$-restart-eq-sat-iff*:
assumes
cdcl$_{NOT}$-restart** S T **and**
inv: *inv* (*fst S*) **and** *n-d*: *no-dup*(*trail* (*fst S*))
shows $I \models$*sextm clauses* (*fst S*) $\longleftrightarrow I \models$*sextm clauses* (*fst T*)
using *assms*(*1*)
proof (*induction rule*: *rtranclp-induct*)
case *base*
then show *?case* **by** *simp*
next
case (*step T U*) **note** *st* = *this*(*1*) **and** *cdcl* = *this*(*2*) **and** *IH* = *this*(*3*)
have *inv* (*fst T*) **and** *no-dup* (*trail* (*fst T*))
using *rtranclp-cdcl$_{NOT}$-with-restart-cdcl$_{NOT}$-inv* **using** *st inv n-d* **by** *blast+*
then have $I \models$*sextm clauses* (*fst T*) $\longleftrightarrow I \models$*sextm clauses* (*fst U*)
using *cdcl$_{NOT}$-restart-eq-sat-iff cdcl* **by** *blast*
then show *?case* **using** *IH* **by** *blast*
qed

lemma *cdcl$_{NOT}$-restart-all-decomposition-implies-m*:
assumes
cdcl$_{NOT}$-restart S T **and**
inv: *inv* (*fst S*) **and** *n-d*: *no-dup*(*trail* (*fst S*)) **and**
all-decomposition-implies-m (*clauses* (*fst S*))
(*get-all-marked-decomposition* (*trail* (*fst S*)))
shows *all-decomposition-implies-m* (*clauses* (*fst T*))
(*get-all-marked-decomposition* (*trail* (*fst T*)))
using *assms*
proof (*induction*)
case (*restart-full S T n*) **note** *full* = *this*(*1*) **and** *inv* = *this*(*2*) **and** *n-d* = *this*(*3*) **and**
decomp = *this*(*4*)
have *st*: *cdcl$_{NOT}$-merged-bj-learn** S T* **and**
n-s: *no-step cdcl$_{NOT}$-merged-bj-learn T*
using *full* **unfolding** *full1-def* **by** (*fast dest*: *tranclp-into-rtranclp*)+
have *st'*: *cdcl$_{NOT}$** S T*
using *inv rtranclp-cdcl$_{NOT}$-merged-bj-learn-is-rtranclp-cdcl$_{NOT}$-and-inv st n-d* **by** *auto*
have *inv T*
using *rtranclp-cdcl$_{NOT}$-cdcl$_{NOT}$-inv*[*OF st*] *inv n-d* **by** *auto*
then show *?case*
using *cdcl$_{NOT}$.rtranclp-cdcl$_{NOT}$-all-decomposition-implies*[*OF - - n-d decomp*] *st' inv* **by** *auto*
next
case (*restart-step m S T n U*) **note** *st* = *this*(*1*) **and** *U* = *this*(*3*) **and** *inv* = *this*(*4*) **and**
n-d = *this*(*5*) **and** *decomp* = *this*(*6*)

show *?case* **using** *U* **by** *auto*
qed

lemma *rtranclp-cdcl*$_{NOT}$*-restart-all-decomposition-implies-m*:
assumes
cdcl$_{NOT}$*-restart*** *S T* **and**
inv: *inv* (*fst S*) **and** *n-d*: *no-dup*(*trail* (*fst S*)) **and**
decomp: *all-decomposition-implies-m* (*clauses* (*fst S*))
(*get-all-marked-decomposition* (*trail* (*fst S*)))
shows *all-decomposition-implies-m* (*clauses* (*fst T*))
(*get-all-marked-decomposition* (*trail* (*fst T*)))
using *assms*
proof (*induction*)
case *base*
then show *?case* **using** *decomp* **by** *simp*
next
case (*step T U*) **note** *st* = *this*(*1*) **and** *cdcl* = *this*(*2*) **and** *IH* = *this*(*3*)[*OF this*(*4*−)] **and**
inv = *this*(*4*) **and** *n-d* = *this*(*5*) **and** *decomp* = *this*(*6*)
have *inv* (*fst T*) **and** *no-dup* (*trail* (*fst T*))
using *rtranclp-cdcl*$_{NOT}$*-with-restart-cdcl*$_{NOT}$*-inv* **using** *st inv n-d* **by** *blast+*
then show *?case*
using *cdcl*$_{NOT}$*-restart-all-decomposition-implies-m*[*OF cdcl*] *IH* **by** *auto*
qed

lemma *full-cdcl*$_{NOT}$*-restart-normal-form*:
assumes
full: *full cdcl*$_{NOT}$*-restart S T* **and**
inv: *inv* (*fst S*) **and** *n-d*: *no-dup*(*trail* (*fst S*)) **and**
decomp: *all-decomposition-implies-m* (*clauses* (*fst S*))
(*get-all-marked-decomposition* (*trail* (*fst S*))) **and**
atms-cls: *atms-of-msu* (*clauses* (*fst S*)) ⊆ *atms-of-ms A* **and**
atms-trail: *atm-of* ‘ *lits-of* (*trail* (*fst S*)) ⊆ *atms-of-ms A* **and**
fin: *finite A*
shows *unsatisfiable* (*set-mset* (*clauses* (*fst S*)))
∨ *lits-of* (*trail* (*fst T*)) ⊨*sextm clauses* (*fst S*) ∧ *satisfiable* (*set-mset* (*clauses* (*fst S*)))
proof −
have *inv-T*: *inv* (*fst T*) **and** *n-d-T*: *no-dup* (*trail* (*fst T*))
using *rtranclp-cdcl*$_{NOT}$*-with-restart-cdcl*$_{NOT}$*-inv* **using** *full inv n-d* **unfolding** *full-def* **by** *blast+*
moreover have
atms-cls-T: *atms-of-msu* (*clauses* (*fst T*)) ⊆ *atms-of-ms A* **and**
atms-trail-T: *atm-of* ‘ *lits-of* (*trail* (*fst T*)) ⊆ *atms-of-ms A*
using *rtranclp-cdcl*$_{NOT}$*-with-restart-bound-inv*[*of S T A*] *full atms-cls atms-trail fin inv n-d*
unfolding *full-def* **by** *blast+*
ultimately have *no-step cdcl*$_{NOT}$*-merged-bj-learn* (*fst T*)
apply −
apply (*rule no-step-cdcl*$_{NOT}$*-restart-no-step-cdcl*$_{NOT}$[*of - A*])
using *full* **unfolding** *full-def* **apply** *simp*
apply *simp*
using *fin* **apply** *simp*
done
moreover have *all-decomposition-implies-m* (*clauses* (*fst T*))
(*get-all-marked-decomposition* (*trail* (*fst T*)))
using *rtranclp-cdcl*$_{NOT}$*-restart-all-decomposition-implies-m*[*of S T*] *inv n-d decomp*
full **unfolding** *full-def* **by** *auto*
ultimately have *unsatisfiable* (*set-mset* (*clauses* (*fst T*)))

$\vee$ *trail* (*fst T*) $\models$*asm clauses* (*fst T*) $\wedge$ *satisfiable* (*set-mset* (*clauses* (*fst T*)))
apply −
apply (*rule* $cdcl_{NOT}$*-merged-bj-learn-final-state*)
using *atms-cls-T atms-trail-T fin n-d-T fin inv-T* **by** *blast+*
then consider
(*unsat*) *unsatisfiable* (*set-mset* (*clauses* (*fst T*)))
| (*sat*) *trail* (*fst T*) $\models$*asm clauses* (*fst T*) **and** *satisfiable* (*set-mset* (*clauses* (*fst T*)))
by *auto*
then show *unsatisfiable* (*set-mset* (*clauses* (*fst S*)))
$\vee$ *lits-of* (*trail* (*fst T*)) $\models$*sextm clauses* (*fst S*) $\wedge$ *satisfiable* (*set-mset* (*clauses* (*fst S*)))
proof *cases*
case *unsat*
then have *unsatisfiable* (*set-mset* (*clauses* (*fst S*)))
unfolding *satisfiable-def* **apply** *auto*
using *rtranclp-*$cdcl_{NOT}$*-restart-eq-sat-iff*[*of S T*] *full inv n-d*
consistent-true-clss-ext-satisfiable true-clss-imp-true-cls-ext
unfolding *satisfiable-def full-def* **by** *blast*
then show *?thesis* **by** *blast*
next
case *sat*
then have *lits-of* (*trail* (*fst T*)) $\models$*sextm clauses* (*fst T*)
using *true-clss-imp-true-cls-ext* **by** (*auto simp*: *true-annots-true-cls*)
then have *lits-of* (*trail* (*fst T*)) $\models$*sextm clauses* (*fst S*)
using *rtranclp-*$cdcl_{NOT}$*-restart-eq-sat-iff*[*of S T*] *full inv n-d* **unfolding** *full-def* **by** *blast*
moreover then have *satisfiable* (*set-mset* (*clauses* (*fst S*)))
using *consistent-true-clss-ext-satisfiable distinctconsistent-interp n-d-T* **by** *fast*
ultimately show *?thesis* **by** *fast*
qed
qed

corollary *full-*$cdcl_{NOT}$*-restart-normal-form-init-state*:
assumes
init-state: *trail S* = [] *clauses S* = *N* **and**
full: *full* $cdcl_{NOT}$*-restart* (*S*, *0*) *T* **and**
inv: *inv S*
shows *unsatisfiable* (*set-mset N*)
$\vee$ *lits-of* (*trail* (*fst T*)) $\models$*sextm N* $\wedge$ *satisfiable* (*set-mset N*)
using *full-*$cdcl_{NOT}$*-restart-normal-form*[*of* (*S*, *0*) *T*] *assms* **by** *auto*

end

end
theory *DPLL-NOT*
imports *CDCL-NOT*
begin

3 DPLL as an instance of NOT

3.1 DPLL with simple backtrack

locale *dpll-with-backtrack*
begin
inductive *backtrack* :: ('*v*, *unit*, *unit*) *ann-literal list* × '*v clauses*
⇒ ('*v*, *unit*, *unit*) *ann-literal list* × '*v clauses* ⇒ *bool* **where**
backtrack-split (*fst S*) = (*M'*, *L* # *M*) ⟹ *is-marked L* ⟹ *D* ∈# *snd S*

$\Longrightarrow$ *fst S* $\models$*as CNot D* $\Longrightarrow$ *backtrack S* (*Propagated* ($-$ (*lit-of L*)) () # *M*, *snd S*)

inductive-cases *backtrackE*[*elim*]: *backtrack* (*M*, *N*) (*M'*, *N'*)
lemma *backtrack-is-backjump*:
fixes *M M'* :: (*'v*, *unit*, *unit*) *ann-literal list*
assumes
backtrack: *backtrack* (*M*, *N*) (*M'*, *N'*) **and**
no-dup: (*no-dup* $\circ$ *fst*) (*M*, *N*) **and**
decomp: *all-decomposition-implies-m N* (*get-all-marked-decomposition M*)
shows
$\exists$ *C F' K F L l C'*.
M = *F'* @ *Marked K* () # *F* $\wedge$
M' = *Propagated L l* # *F* $\wedge$ *N* = *N'* $\wedge$ *C* $\in$# *N* $\wedge$ *F'* @ *Marked K d* # *F* $\models$*as CNot C* $\wedge$
undefined-lit F L $\wedge$ *atm-of L* $\in$ *atms-of-msu N* $\cup$ *atm-of* ‘ *lits-of* (*F'* @ *Marked K d* # *F*) $\wedge$
N $\models$*pm C'* + {#*L*#} $\wedge$ *F* $\models$*as CNot C'*
proof $-$
let *?S* = (*M*, *N*)
let *?T* = (*M'*, *N'*)
obtain *F F' P L D* **where**
b-sp: *backtrack-split M* = (*F'*, *L* # *F*) **and**
is-marked L **and**
D $\in$# *snd ?S* **and**
M $\models$*as CNot D* **and**
bt: *backtrack ?S* (*Propagated* ($-$ (*lit-of L*)) *P* # *F*, *N*) **and**
M': *M'* = *Propagated* ($-$ (*lit-of L*)) *P* # *F* **and**
[*simp*]: *N'* = *N*
using *backtrackE*[*OF backtrack*] **by** (*metis backtrack fstI sndI*)
let *?K* = *lit-of L*
let *?C* = *image-mset lit-of* {#*K*$\in$#*mset M*. *is-marked K* $\wedge$ *K*$\neq$*L*#} :: *'v literal multiset*
let *?C'* = *set-mset* (*image-mset single* (*?C*+{#*?K*#}))
obtain *K* **where** *L*: *L* = *Marked K* () **using** ‹*is-marked L*› **by** (*cases L*) *auto*

have *M*: *M* = *F'* @ *Marked K* () # *F*
using *b-sp* **by** (*metis L backtrack-split-list-eq fst-conv snd-conv*)
moreover have *F'* @ *Marked K* () # *F* $\models$*as CNot D*
using ‹*M*$\models$*as CNot D*› **unfolding** *M* **.**
moreover have *undefined-lit F* ($-$*?K*)
using *no-dup* **unfolding** *M L* **by** (*simp add*: *defined-lit-map*)
moreover have *atm-of* ($-$*K*) $\in$ *atms-of-msu N* $\cup$ *atm-of* ‘ *lits-of* (*F'* @ *Marked K d* # *F*)
by *auto*
moreover
have *set-mset N* $\cup$ *?C'* $\models$*ps* {{#}}
proof $-$
have *A*: *set-mset N* $\cup$ *?C'* $\cup$ *unmark M* =
set-mset N $\cup$ *unmark M*
unfolding *M L* **by** *auto*
have *set-mset N* $\cup$ {{#*lit-of L*#} |*L*. *is-marked L* $\wedge$ *L* $\in$ *set M*}
$\models$*ps unmark M*
using *all-decomposition-implies-propagated-lits-are-implied*[*OF decomp*] **.**
moreover have *C'*: *?C'* = {{#*lit-of L*#} |*L*. *is-marked L* $\wedge$ *L* $\in$ *set M*}
unfolding *M L* **apply** *standard*
apply *force*
using *IntI* **by** *auto*
ultimately have *N-C-M*: *set-mset N* $\cup$ *?C'* $\models$*ps unmark M*
by *auto*

have *set-mset N* $\cup$ (λ*L.* {#*lit-of L*#}) ‘ (*set M*) $\models$*ps* {{#}}
  unfolding *true-clss-clss-def*
  proof (*intro allI impI*, *goal-cases*)
    case (*1 I*) **note** *tot* = *this*(*1*) **and** *cons* = *this*(*2*) **and** *I-N-M* = *this*(*3*)
    have *I* $\models$ *D*
      using *I-N-M* ‹*D* $\in$# *snd ?S*› **unfolding** *true-clss-def* **by** *auto*
    moreover have *I* $\models$*s CNot D*
      using ‹*M* $\models$*as CNot D*› **unfolding** *M* **by** (*metis 1*(*3*) ‹*M* $\models$*as CNot D*›
        true-annots-true-cls true-cls-mono-set-mset-l true-clss-def
        true-clss-singleton-lit-of-implies-incl true-clss-union)
    ultimately show *?case* **using** *cons consistent-CNot-not* **by** *blast*
  qed
then show *?thesis*
  using *true-clss-clss-left-right*[*OF N-C-M*, *of* {{#}}] **unfolding** *A* **by** *auto*
qed
have *N* $\models$*pm image-mset uminus ?C* + {#−*?K*#}
unfolding *true-clss-cls-def true-clss-clss-def total-over-m-def*
proof (*intro allI impI*)
  fix *I*
  assume
   tot: *total-over-set I* (*atms-of-ms* (*set-mset N* $\cup$ {*image-mset uminus ?C* + {#− *?K*#}})) **and**
   cons: *consistent-interp I* **and**
   I $\models$*sm N*
  have ($K \in I \wedge -K \notin I$) $\vee$ ($-K \in I \wedge K \notin I$)
    using *cons tot* **unfolding** *consistent-interp-def L* **by** (*cases K*) *auto*
  have *tI*: *total-over-set I* (*atm-of* ‘ *lit-of* ‘ (*set M* $\cap$ {*L. is-marked L* $\wedge$ *L* $\neq$ *Marked K d*}))
    using *tot* **by** (*auto simp add*: *L atms-of-uminus-lit-atm-of-lit-of*)

  then have *H*: $\bigwedge$*x.*
      lit-of x $\notin$ *I* $\Longrightarrow$ *x* $\in$ *set M* $\Longrightarrow$*is-marked x*
      $\Longrightarrow$ *x* $\neq$ *Marked K d* $\Longrightarrow$ −*lit-of x* $\in$ *I*
    proof −
      fix *x* :: (*'v*, *unit*, *unit*) *ann-literal*
      assume *a1*: *x* $\neq$ *Marked K d*
      assume *a2*: *is-marked x*
      assume *a3*: *x* $\in$ *set M*
      assume *a4*: *lit-of x* $\notin$ *I*
      have *atm-of* (*lit-of x*) $\in$ *atm-of* ‘ *lit-of* ‘
        (*set M* $\cap$ {*m. is-marked m* $\wedge$ *m* $\neq$ *Marked K d*})
        using *a3 a2 a1* **by** *blast*
      then have *Pos* (*atm-of* (*lit-of x*)) $\in$ *I* $\vee$ *Neg* (*atm-of* (*lit-of x*)) $\in$ *I*
        using *tI* **unfolding** *total-over-set-def* **by** *blast*
      then show − *lit-of x* $\in$ *I*
        using *a4* **by** (*metis* (*no-types*) *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set*
          literal.sel(*1*,*2*))
    qed
  have $\neg$*I* $\models$*s ?C'*
    using ‹*set-mset N* $\cup$ *?C'* $\models$*ps* {{#}}› *tot cons* ‹*I* $\models$*sm N*›
    unfolding *true-clss-clss-def total-over-m-def*
    by (*simp add*: *atms-of-uminus-lit-atm-of-lit-of atms-of-ms-single-image-atm-of-lit-of*)
  then show *I* $\models$ *image-mset uminus ?C* + {#− *lit-of L*#}
    unfolding *true-clss-def true-cls-def Bex-mset-def*
    using ‹($K \in I \wedge -K \notin I$) $\vee$ ($-K \in I \wedge K \notin I$)›
    unfolding *L* **by** (*auto dest*!: *H*)
qed

moreover
  have *set* $F' \cap \{K.$ *is-marked* $K \wedge K \neq L\} = \{\}$
    using *backtrack-split-fst-not-marked*[*of* - *M*] *b-sp* **by** *auto*
  then have $F \models as$ *CNot* (*image-mset uminus ?C*)
    unfolding *M CNot-def true-annots-def* **by** (*auto simp add*: *L lits-of-def*)
ultimately show *?thesis*
  using M' ‹$D \in\#$ *snd ?S*› *L* **by** *force*
qed

lemma *backtrack-is-backjump'*:
  fixes M M' :: ('v, *unit*, *unit*) *ann-literal list*
  assumes
    backtrack: *backtrack S T* **and**
    no-dup: (*no-dup* ∘ *fst*) *S* **and**
    decomp: *all-decomposition-implies-m* (*snd S*) (*get-all-marked-decomposition* (*fst S*))
  shows
    $\exists$ C F' K F L l C'.
      fst $S = F'$ @ *Marked* K () # F ∧
      T = (*Propagated* L l # F, *snd* S) ∧ $C \in\#$ *snd* S ∧ *fst* $S \models as$ *CNot* C
      ∧ *undefined-lit* F L ∧ *atm-of* $L \in$ *atms-of-msu* (*snd* S) ∪ *atm-of* ‘ *lits-of* (*fst* S) ∧
      snd $S \models pm$ $C' + \{\#L\#\}$ ∧ $F \models as$ *CNot* C'
  apply (*cases S*, *cases T*)
  using *backtrack-is-backjump*[*of fst S snd S fst T snd T*] *assms* **by** *fastforce*

sublocale *dpll-state fst snd* λL (M, N). $(L$ # $M, N)$ $\lambda(M, N)$. (*tl* M, N)
  λC (M, N). $(M, \{\#C\#\} + N)$ λC (M, N). (M, *remove-mset* C N)
  by *unfold-locales auto*

sublocale *backjumping-ops fst snd* λL (M, N). $(L$ # $M, N)$ $\lambda(M, N)$. (*tl* M, N)
  λC (M, N). $(M, \{\#C\#\} + N)$ λC (M, N). (M, *remove-mset* C N) λ- - - S T. *backtrack S T*
  by *unfold-locales*

lemma *backtrack-is-backjump''*:
  fixes M M' :: ('v, *unit*, *unit*) *ann-literal list*
  assumes
    backtrack: *backtrack S T* **and**
    no-dup: (*no-dup* ∘ *fst*) *S* **and**
    decomp: *all-decomposition-implies-m* (*snd S*) (*get-all-marked-decomposition* (*fst S*))
    shows *backjump S T*
proof –
  obtain C F' K F L l C' **where**
    1: *fst* $S = F'$ @ *Marked* K () # F **and**
    2: T = (*Propagated* L l # F, *snd* S) **and**
    3: $C \in\#$ *snd* S **and**
    4: *fst* $S \models as$ *CNot* C **and**
    5: *undefined-lit* F L **and**
    6: *atm-of* $L \in$ *atms-of-msu* (*snd* S) ∪ *atm-of* ‘ *lits-of* (*fst* S) **and**
    7: *snd* $S \models pm$ $C' + \{\#L\#\}$ **and**
    8: $F \models as$ *CNot* C'
  using *backtrack-is-backjump'*[*OF assms*] **by** *blast*
  show *?thesis*
    using *backjump.intros*[*OF 1 - 3 4 5 6 7 8*] *2 backtrack 1 5*
    by (*auto simp*: *state-eq*$_{NOT}$*-def simp del*: *state-simp*$_{NOT}$)
qed

lemma *can-do-bt-step*:
  assumes
    M: *fst* $S = F'$ @ *Marked* K d # F **and**
    $C \in\#$ *snd* S **and**
    C: *fst* $S \models as$ *CNot* C
  shows $\neg$ *no-step backtrack* S
proof –
  obtain L G' G **where**
    backtrack-split (*fst* S) = (G', L # G)
    unfolding M **by** (*induction* F' *rule*: *ann-literal-list-induct*) *auto*
  moreover then have *is-marked* L
    by (*metis backtrack-split-snd-hd-marked list.distinct*(*1*) *list.sel*(*1*) *snd-conv*)
  ultimately show *?thesis*
    using *backtrack.intros*[*of* S G' L G C] ‹$C \in\#$ *snd* S› C **unfolding** M **by** *auto*
qed

end

sublocale *dpll-with-backtrack* $\subseteq$ *dpll-with-backjumping-ops fst snd* λL (M, N). (L # M, N)
  λ(M, N). (*tl* M, N) λC (M, N). (M, {#C#} + N) λC (M, N). (M, *remove-mset* C N) λ- -. *True*
  λ(M, N). *no-dup* M $\wedge$ *all-decomposition-implies-m* N (*get-all-marked-decomposition* M)
  λ- - - S T. *backtrack* S T
  by *unfold-locales* (*metis* (*mono-tags*, *lifting*) *dpll-with-backtrack.backtrack-is-backjump''*
  dpll-with-backtrack.can-do-bt-step prod.case-eq-if comp-apply)

sublocale *dpll-with-backtrack* $\subseteq$ *dpll-with-backjumping fst snd* λL (M, N). (L # M, N)
  λ(M, N). (*tl* M, N) λC (M, N). (M, {#C#} + N) λC (M, N). (M, *remove-mset* C N) λ- -. *True*
  λ(M, N). *no-dup* M $\wedge$ *all-decomposition-implies-m* N (*get-all-marked-decomposition* M)
  λ- - - S T. *backtrack* S T
  apply *unfold-locales*
  using *dpll-bj-no-dup dpll-bj-all-decomposition-implies-inv* **apply** *fastforce*
  done

sublocale *dpll-with-backtrack* $\subseteq$ *conflict-driven-clause-learning-ops*
  fst snd λL (M, N). (L # M, N)
  λ(M, N). (*tl* M, N) λC (M, N). (M, {#C#} + N) λC (M, N). (M, *remove-mset* C N) λ- -. *True*
  λ(M, N). *no-dup* M $\wedge$ *all-decomposition-implies-m* N (*get-all-marked-decomposition* M)
  λ- - - S T. *backtrack* S T λ- -. *False* λ- -. *False*
  by *unfold-locales*

sublocale *dpll-with-backtrack* $\subseteq$ *conflict-driven-clause-learning*
  fst snd λL (M, N). (L # M, N)
  λ(M, N). (*tl* M, N) λC (M, N). (M, {#C#} + N) λC (M, N). (M, *remove-mset* C N) λ- -. *True*
  λ(M, N). *no-dup* M $\wedge$ *all-decomposition-implies-m* N (*get-all-marked-decomposition* M)
  λ- - - S T. *backtrack* S T λ- -. *False* λ- -. *False*
  apply *unfold-locales*
  using $cdcl_{NOT}$*.simps dpll-bj-inv* $forget_{NOT}E$ $learn_{NOT}E$ **by** *blast*

context *dpll-with-backtrack*
begin
lemma *wf-tranclp-dpll-inital-state*:
  assumes *fin*: *finite* A
  shows *wf* {((M'::('*v*, *unit*, *unit*) *ann-literals*, N'::'*v clauses*), ([], N))|M' N' N.
    $dpll\text{-}bj^{++}$ ([], N) (M', N') $\wedge$ *atms-of-msu* $N \subseteq$ *atms-of-ms* A}
  using *wf-tranclp-dpll-bj*[*OF assms*(*1*)] **by** (*rule wf-subset*) *auto*

corollary *full-dpll-final-state-conclusive*:
  fixes $M\ M'$ :: ($'v$, *unit*, *unit*) *ann-literal list*
  assumes
    full: *full dpll-bj* ([], N) (M', N')
  shows *unsatisfiable* (*set-mset* N) $\vee$ ($M' \models asm\ N \wedge$ *satisfiable* (*set-mset* N))
  using *assms full-dpll-backjump-final-state*[*of* ([],N) (M', N') *set-mset* N] **by** *auto*

corollary *full-dpll-normal-form-from-init-state*:
  fixes $M\ M'$ :: ($'v$, *unit*, *unit*) *ann-literal list*
  assumes
    full: *full dpll-bj* ([], N) (M', N')
  shows $M' \models asm\ N \longleftrightarrow$ *satisfiable* (*set-mset* N)
proof −
  have *no-dup* M'
    using *rtranclp-dpll-bj-no-dup*[*of* ([], N) (M', N')]
    full **unfolding** *full-def* **by** *auto*
  then have $M' \models asm\ N \Longrightarrow$ *satisfiable* (*set-mset* N)
    using *distinctconsistent-interp satisfiable-carac′ true-annots-true-cls* **by** *blast*
  then show *?thesis*
  using *full-dpll-final-state-conclusive*[*OF full*] **by** *auto*
qed

lemma $cdcl_{NOT}$*-is-dpll*:
  $cdcl_{NOT}$ S T $\longleftrightarrow$ *dpll-bj* S T
  by (*auto simp*: $cdcl_{NOT}$*.simps learn.simps* $forget_{NOT}$*.simps*)

Another proof of termination:

lemma *wf* {(T, S). *dpll-bj* S T $\wedge$ $cdcl_{NOT}$*-NOT-all-inv* A S}
  unfolding $cdcl_{NOT}$*-is-dpll*[*symmetric*]
  by (*rule wf-*$cdcl_{NOT}$*-no-learn-and-forget-infinite-chain*)
  (*auto simp*: *learn.simps* $forget_{NOT}$*.simps*)
end

3.2 Adding restarts

locale *dpll-withbacktrack-and-restarts* =
  dpll-with-backtrack +
  fixes f :: *nat* $\Rightarrow$ *nat*
  assumes *unbounded*: *unbounded* f **and** *f-ge-1*:$\bigwedge n.\ n \geq 1 \Longrightarrow f\ n \geq 1$
begin
  sublocale $cdcl_{NOT}$*-increasing-restarts* *fst snd* λL (M, N). (L # M, N) λ(M, N). (*tl* M, N)
    λC (M, N). (M, {#C#} + N) λC (M, N). (M, *remove-mset* C N) f λ(-, N) S. S = ([], N)
  λA (M, N). *atms-of-msu* $N \subseteq$ *atms-of-ms* A $\wedge$ *atm-of* ‘ *lits-of* $M \subseteq$ *atms-of-ms* A $\wedge$ *finite* A
    $\wedge$ *all-decomposition-implies-m* N (*get-all-marked-decomposition* M)
  λA T. (*2+card* (*atms-of-ms* A)) ^ (*1+card* (*atms-of-ms* A))
              − μ_C (*1+card* (*atms-of-ms* A)) (*2+card* (*atms-of-ms* A)) (*trail-weight* T) *dpll-bj*
  λ(M, N). *no-dup* M $\wedge$ *all-decomposition-implies-m* N (*get-all-marked-decomposition* M)
  λA -. (*2+card* (*atms-of-ms* A)) ^ (*1+card* (*atms-of-ms* A))
  apply *unfold-locales*
        apply (*rule unbounded*)
       using *f-ge-1* **apply** *fastforce*
      apply (*smt dpll-bj-all-decomposition-implies-inv dpll-bj-atms-in-trail-in-set*
        dpll-bj-clauses dpll-bj-no-dup prod.case-eq-if)
     apply (*rule dpll-bj-trail-mes-decreasing-prop*; *auto*)
    apply (*rename-tac A T U*, *case-tac T*, *simp*)

apply (*rename-tac A T U*, *case-tac U*, *simp*)
using *dpll-bj-clauses dpll-bj-all-decomposition-implies-inv dpll-bj-no-dup* **by** *fastforce+*
end

end
theory *DPLL-W*
imports *Main Partial-Clausal-Logic Partial-Annotated-Clausal-Logic List-More Wellfounded-More DPLL-NOT*
begin

4 DPLL

4.1 Rules

type-synonym $'a$ $dpll_W$*-ann-literal* = ($'a$, *unit*, *unit*) *ann-literal*
type-synonym $'a$ $dpll_W$*-ann-literals* = ($'a$, *unit*, *unit*) *ann-literals*
type-synonym $'v$ $dpll_W$*-state* = $'v$ $dpll_W$*-ann-literals* × $'v$ *clauses*

abbreviation *trail* :: $'v$ $dpll_W$*-state* ⇒ $'v$ $dpll_W$*-ann-literals* **where**
trail ≡ *fst*
abbreviation *clauses* :: $'v$ $dpll_W$*-state* ⇒ $'v$ *clauses* **where**
clauses ≡ *snd*

The definition of DPLL is given in figure 2.13 page 70 of CW.

inductive $dpll_W$ :: $'v$ $dpll_W$*-state* ⇒ $'v$ $dpll_W$*-state* ⇒ *bool* **where**
propagate: *C* + {#*L*#} ∈# *clauses S* ⟹ *trail S* ⊨*as CNot C* ⟹ *undefined-lit* (*trail S*) *L*
⟹ $dpll_W$ *S* (*Propagated L* () # *trail S*, *clauses S*) |
decided: *undefined-lit* (*trail S*) *L* ⟹ *atm-of L* ∈ *atms-of-msu* (*clauses S*)
⟹ $dpll_W$ *S* (*Marked L* () # *trail S*, *clauses S*) |
backtrack: *backtrack-split* (*trail S*) = (*M'*, *L* # *M*) ⟹ *is-marked L* ⟹ *D* ∈# *clauses S*
⟹ *trail S* ⊨*as CNot D* ⟹ $dpll_W$ *S* (*Propagated* (− (*lit-of L*)) () # *M*, *clauses S*)

4.2 Invariants

lemma $dpll_W$*-distinct-inv*:
assumes $dpll_W$ *S S'*
and *no-dup* (*trail S*)
shows *no-dup* (*trail S'*)
using *assms*
proof (*induct rule*: $dpll_W$*.induct*)
case (*decided L S*)
then show *?case* **using** *defined-lit-map* **by** *force*
next
case (*propagate C L S*)
then show *?case* **using** *defined-lit-map* **by** *force*
next
case (*backtrack S M' L M D*) **note** *extracted* = *this*(*1*) **and** *no-dup* = *this*(*5*)
show *?case*
using *no-dup backtrack-split-list-eq*[*of trail S*, *symmetric*] **unfolding** *extracted* **by** *auto*
qed

lemma $dpll_W$*-consistent-interp-inv*:
assumes $dpll_W$ *S S'*
and *consistent-interp* (*lits-of* (*trail S*))
and *no-dup* (*trail S*)

shows *consistent-interp* (*lits-of* (*trail* S'))
using *assms*
proof (*induct rule*: $dpll_W$*.induct*)
case (*backtrack* $S\ M'\ L\ M\ D$) **note** *extracted* = *this*(*1*) **and** *marked* = *this*(*2*) **and** D = *this*(*4*) **and** *cons* = *this*(*5*) **and** *no-dup* = *this*(*6*)
have *no-dup'*: *no-dup M*
by (*metis* (*no-types*) *backtrack-split-list-eq distinct.simps*(*2*) *distinct-append extracted list.simps*(*9*) *map-append no-dup snd-conv*)
then have *insert* (*lit-of L*) (*lits-of M*) ⊆ *lits-of* (*trail S*)
using *backtrack-split-list-eq*[*of trail S*, *symmetric*] **unfolding** *extracted* **by** *auto*
then have *cons*: *consistent-interp* (*insert* (*lit-of L*) (*lits-of M*))
using *consistent-interp-subset cons* **by** *blast*
moreover
have *lit-of L* ∉ *lits-of M*
using *no-dup backtrack-split-list-eq*[*of trail S*, *symmetric*] *extracted*
unfolding *lits-of-def* **by** *force*
moreover
have *atm-of* (−*lit-of L*) ∉ (λ*m*. *atm-of* (*lit-of m*)) ‘ *set M*
using *no-dup backtrack-split-list-eq*[*of trail S*, *symmetric*] **unfolding** *extracted* **by** *force*
then have −*lit-of L* ∉ *lits-of M*
unfolding *lits-of-def* **by** *force*
ultimately show *?case* **by** *simp*
qed (*auto intro*: *consistent-add-undefined-lit-consistent*)

lemma $dpll_W$*-vars-in-snd-inv*:
assumes $dpll_W$ $S\ S'$
and *atm-of* ‘ (*lits-of* (*trail S*)) ⊆ *atms-of-msu* (*clauses S*)
shows *atm-of* ‘ (*lits-of* (*trail* S')) ⊆ *atms-of-msu* (*clauses* S')
using *assms*
proof (*induct rule*: $dpll_W$*.induct*)
case (*backtrack* $S\ M'\ L\ M\ D$)
then have *atm-of* (*lit-of L*) ∈ *atms-of-msu* (*clauses S*)
using *backtrack-split-list-eq*[*of trail S*, *symmetric*] **by** *auto*
moreover
have *atm-of* ‘ *lits-of* (*trail S*) ⊆ *atms-of-msu* (*clauses S*)
using *backtrack*(*5*) **by** *simp*
then have ⋀*xb*. *xb* ∈ *set M* ⟹ *atm-of* (*lit-of xb*) ∈ *atms-of-msu* (*clauses S*)
using *backtrack-split-list-eq*[*symmetric*, *of trail S*] *backtrack.hyps*(*1*)
unfolding *lits-of-def* **by** *auto*
ultimately show *?case* **by** (*auto simp* : *lits-of-def*)
qed (*auto simp*: *in-plus-implies-atm-of-on-atms-of-ms*)

lemma *atms-of-ms-lit-of-atms-of*: *atms-of-ms* ((λ*a*. {#*lit-of a*#}) ‘ *c*) = *atm-of* ‘ *lit-of* ‘ *c*
unfolding *atms-of-ms-def* **using** *image-iff* **by** *force*

Lemma theorem 2.8.2 page 71 of CW

lemma $dpll_W$*-propagate-is-conclusion*:
assumes $dpll_W$ $S\ S'$
and *all-decomposition-implies-m* (*clauses S*) (*get-all-marked-decomposition* (*trail S*))
and *atm-of* ‘ *lits-of* (*trail S*) ⊆ *atms-of-msu* (*clauses S*)
shows *all-decomposition-implies-m* (*clauses* S') (*get-all-marked-decomposition* (*trail* S'))
using *assms*
proof (*induct rule*: $dpll_W$*.induct*)
case (*decided L S*)
then show *?case* **unfolding** *all-decomposition-implies-def* **by** *simp*

next
  case (*propagate C L S*) **note** *inS* = *this*(*1*) **and** *cnot* = *this*(*2*) **and** *IH* = *this*(*4*) **and** *undef* = *this*(*3*) **and** *atms-incl* = *this*(*5*)
  let *?I* = *set* (*map* (λ*a*. {#*lit-of a*#}) (*trail S*)) ∪ *set-mset* (*clauses S*)
  have *?I* ⊨*p C* + {#*L*#} **by** (*auto simp add*: *inS*)
  moreover have *?I* ⊨*ps CNot C* **using** *true-annots-true-clss-cls cnot* **by** *fastforce*
  ultimately have *?I* ⊨*p* {#*L*#} **using** *true-clss-cls-plus-CNot*[*of ?I C L*] *inS* **by** *blast*
  {
    assume *get-all-marked-decomposition* (*trail S*) = []
    then have *?case* **by** *blast*
  }
  moreover {
    assume *n*: *get-all-marked-decomposition* (*trail S*) ≠ []
    have *1*: ⋀*a b*. (*a*, *b*) ∈ *set* (*tl* (*get-all-marked-decomposition* (*trail S*)))
      ⟹ (*unmark a* ∪ *set-mset* (*clauses S*)) ⊨*ps unmark b*
      using *IH* **unfolding** *all-decomposition-implies-def* **by** (*fastforce simp add*: *list.set-sel*(*2*) *n*)
    moreover have *2*: ⋀*a c*. *hd* (*get-all-marked-decomposition* (*trail S*)) = (*a*, *c*)
      ⟹ (*unmark a* ∪ *set-mset* (*clauses S*)) ⊨*ps* (*unmark c*)
      by (*metis IH all-decomposition-implies-cons-pair all-decomposition-implies-single list.collapse n*)
    moreover have *3*: ⋀*a c*. *hd* (*get-all-marked-decomposition* (*trail S*)) = (*a*, *c*)
      ⟹ (*unmark a* ∪ *set-mset* (*clauses S*)) ⊨*p* {#*L*#}
      proof −
        fix *a c*
        assume *h*: *hd* (*get-all-marked-decomposition* (*trail S*)) = (*a*, *c*)
        have *h'*: *trail S* = *c* @ *a* **using** *get-all-marked-decomposition-decomp h* **by** *blast*
        have *I*: *set* (*map* (λ*a*. {#*lit-of a*#}) *a*) ∪ *set-mset* (*clauses S*)
          ∪ *unmark c* ⊨*ps CNot C*
          using ‹*?I* ⊨*ps CNot C*› **unfolding** *h'* **by** (*simp add*: *Un-commute Un-left-commute*)
        have
          atms-of-ms (*CNot C*) ⊆ *atms-of-ms* (*set* (*map* (λ*a*. {#*lit-of a*#}) *a*) ∪ *set-mset* (*clauses S*))
            and
          atms-of-ms (*unmark c*) ⊆ *atms-of-ms* (*set* (*map* (λ*a*. {#*lit-of a*#}) *a*)
            ∪ *set-mset* (*clauses S*))
            apply (*metis CNot-plus Un-subset-iff atms-of-atms-of-ms-mono atms-of-ms-CNot-atms-of atms-of-ms-union inS mem-set-mset-iff sup.coboundedI2*)
          using *inS atms-of-atms-of-ms-mono atms-incl* **by** (*fastforce simp*: *h'*)

        then have *unmark a* ∪ *set-mset* (*clauses S*) ⊨*ps CNot C*
          using *true-clss-clss-left-right*[*OF - I*] *h 2* **by** *auto*
        then show *unmark a* ∪ *set-mset* (*clauses S*) ⊨*p* {#*L*#}
          by (*metis* (*no-types*) *Un-insert-right inS insertI1 mk-disjoint-insert inS mem-set-mset-iff true-clss-cls-in true-clss-cls-plus-CNot*)
      qed
    ultimately have *?case*
      by (*cases hd* (*get-all-marked-decomposition* (*trail S*)))
        (*auto simp*: *all-decomposition-implies-def*)
  }
  ultimately show *?case* **by** *auto*
next
  case (*backtrack S M' L M D*) **note** *extracted* = *this*(*1*) **and** *marked* = *this*(*2*) **and** *D* = *this*(*3*) **and** *cnot* = *this*(*4*) **and** *cons* = *this*(*4*) **and** *IH* = *this*(*5*) **and** *atms-incl* = *this*(*6*)
  have *S*: *trail S* = *M'* @ *L* # *M*
    using *backtrack-split-list-eq*[*of trail S*] **unfolding** *extracted* **by** *auto*
  have *M'*: ∀ *l* ∈ *set M'*. ¬*is-marked l*

using *extracted backtrack-split-fst-not-marked*[*of - trail S*] **by** *simp*
have *n*: *get-all-marked-decomposition* (*trail S*) ≠ [] **by** *auto*
then have *all-decomposition-implies-m* (*clauses S*) ((*L* # *M*, *M'*)
tl (*get-all-marked-decomposition* (*trail S*)))
by (*metis* (*no-types*) *IH extracted get-all-marked-decomposition-backtrack-split list.exhaust-sel*)
then have *1*: *unmark* (*L* # *M*) ∪ *set-mset* (*clauses S*) ⊨*ps*(*λa.*{#*lit-of a*#}) ' *set M'*
by *simp*
moreover
have *unmark* (*L* # *M*) ∪ *unmark M'* ⊨*ps CNot D*
by (*metis* (*mono-tags*, *lifting*) *S Un-commute cons image-Un set-append*
true-annots-true-clss-clss)
then have *2*: *unmark* (*L* # *M*) ∪ *set-mset* (*clauses S*) ∪ *unmark M'*
⊨*ps CNot D*
by (*metis* (*no-types*, *lifting*) *Un-assoc Un-left-commute true-clss-clss-union-l-r*)
ultimately
have *set* (*map* (*λa.* {#*lit-of a*#}) (*L* # *M*)) ∪ *set-mset* (*clauses S*) ⊨*ps CNot D*
using *true-clss-clss-left-right* **by** *fastforce*
then have *set* (*map* (*λa.* {#*lit-of a*#}) (*L* # *M*)) ∪ *set-mset* (*clauses S*) ⊨*p* {#}
by (*metis* (*mono-tags*, *lifting*) *D Un-def mem-Collect-eq set-mset-def*
true-clss-clss-contradiction-true-clss-cls-false)
then have *IL*: *unmark M* ∪ *set-mset* (*clauses S*) ⊨*p* {#−*lit-of L*#}
using *true-clss-clss-false-left-right* **by** *auto*
show *?case* **unfolding** *S all-decomposition-implies-def*
proof
fix *x P level*
assume *x*: *x* ∈ *set* (*get-all-marked-decomposition*
(*fst* (*Propagated* (− *lit-of L*) *P* # *M*, *clauses S*)))
let *?M'* = *Propagated* (− *lit-of L*) *P* # *M*
let *?hd* = *hd* (*get-all-marked-decomposition ?M'*)
let *?tl* = *tl* (*get-all-marked-decomposition ?M'*)
have *x* = *?hd* ∨ *x* ∈ *set ?tl*
using *x*
by (*cases get-all-marked-decomposition ?M'*)
auto
moreover {
assume *x'*: *x* ∈ *set ?tl*
have *L'*: *Marked* (*lit-of L*) () = *L* **using** *marked* **by** (*cases L*, *auto*)
have *x* ∈ *set* (*get-all-marked-decomposition* (*M'* @ *L* # *M*))
using *x' get-all-marked-decomposition-except-last-choice-equal*[*of M' lit-of L P M*]
L' **by** (*metis* (*no-types*) *M' list.set-sel*(*2*) *tl-Nil*)
then have *case x of* (*Ls*, *seen*) ⇒ *unmark Ls* ∪ *set-mset* (*clauses S*)
⊨*ps unmark seen*
using *marked IH* **by** (*cases L*) (*auto simp add*: *S all-decomposition-implies-def*)
}
moreover {
assume *x'*: *x* = *?hd*
have *tl*: *tl* (*get-all-marked-decomposition* (*M'* @ *L* # *M*)) ≠ []
proof −
have *f1*: ⋀*ms. length* (*get-all-marked-decomposition* (*M'* @ *ms*))
= *length* (*get-all-marked-decomposition ms*)
by (*simp add*: *M' get-all-marked-decomposition-remove-unmarked-length*)
have *Suc* (*length* (*get-all-marked-decomposition M*)) ≠ *Suc 0*
by *blast*
then show *?thesis*
using *f1 marked* **by** (*metis* (*no-types*) *get-all-marked-decomposition.simps*(*1*) *length-tl*

list.sel(3) list.size(3) ann-literal.collapse(1))
qed
obtain $M0'$ $M0$ **where**
L0: *hd* (*tl* (*get-all-marked-decomposition* (M' @ L # M))) = ($M0$, $M0'$)
by (*cases hd* (*tl* (*get-all-marked-decomposition* (M' @ L # M))))
have x'': x = ($M0$, *Propagated* (−*lit-of* L) P # $M0'$)
unfolding x' **using** *get-all-marked-decomposition-last-choice tl* M' *L0*
by (*metis marked ann-literal.collapse*(*1*))
obtain *l-get-all-marked-decomposition* **where**
get-all-marked-decomposition (*trail S*) = (L # M, M') # ($M0$, $M0'$) #
l-get-all-marked-decomposition
using *get-all-marked-decomposition-backtrack-split extracted* **by** (*metis* (*no-types*) *L0 S hd-Cons-tl n tl*)
then have $M = M0'$ @ $M0$ **using** *get-all-marked-decomposition-hd-hd* **by** *fastforce*
then have IL': *unmark M0* ∪ *set-mset* (*clauses S*)
∪ *unmark* $M0'$ ⊨*ps* {{#− *lit-of L*#}}
using *IL* **by** (*simp add*: *Un-commute Un-left-commute image-Un*)
moreover have *H*: *unmark M0* ∪ *set-mset* (*clauses S*)
⊨*ps unmark* $M0'$
using *IH* x'' **unfolding** *all-decomposition-implies-def* **by** (*metis* (*no-types*, *lifting*) *L0 S list.set-sel*(*1*) *list.set-sel*(*2*) *old.prod.case tl tl-Nil*)
ultimately have *case x of* (*Ls*, *seen*) ⇒ *unmark Ls* ∪ *set-mset* (*clauses S*)
⊨*ps unmark seen*
using *true-clss-clss-left-right* **unfolding** x'' **by** *auto*
}
ultimately show *case x of* (*Ls*, *seen*) ⇒
unmark Ls ∪ *set-mset* (*snd* (*?M'*, *clauses S*))
⊨*ps unmark seen*
unfolding *snd-conv* **by** *blast*
qed
qed

Lemma theorem 2.8.3 page 72 of CW

theorem $dpll_W$*-propagate-is-conclusion-of-decided*:
assumes $dpll_W$ S S'
and *all-decomposition-implies-m* (*clauses S*) (*get-all-marked-decomposition* (*trail S*))
and *atm-of* ‘ *lits-of* (*trail S*) ⊆ *atms-of-msu* (*clauses S*)
shows *set-mset* (*clauses* S') ∪ {{#*lit-of L*#} |*L. is-marked L* ∧ *L* ∈ *set* (*trail* S')}
⊨*ps* (λ*a.* {#*lit-of a*#}) ‘ ⋃(*set* ‘ *snd* ‘ *set* (*get-all-marked-decomposition* (*trail* S')))
using *all-decomposition-implies-trail-is-implied*[*OF* $dpll_W$*-propagate-is-conclusion*[*OF assms*]] **.**

Lemma theorem 2.8.4 page 72 of CW

lemma *only-propagated-vars-unsat*:
assumes *marked*: ∀ *x* ∈ *set M*. ¬ *is-marked x*
and *DN*: *D* ∈ *N* **and** *D*: *M* ⊨*as CNot D*
and *inv*: *all-decomposition-implies N* (*get-all-marked-decomposition M*)
and *atm-incl*: *atm-of* ‘ *lits-of M* ⊆ *atms-of-ms N*
shows *unsatisfiable N*
proof (*rule ccontr*)
assume ¬ *unsatisfiable N*
then obtain *I* **where**
I: *I* ⊨*s N* **and**
cons: *consistent-interp I* **and**
tot: *total-over-m I N*
unfolding *satisfiable-def* **by** *auto*

```
  then have I-D: I ⊨ D
    using DN unfolding true-clss-def by auto

  have l0: {{#lit-of L#} |L. is-marked L ∧ L ∈ set M} = {} using marked by auto
  have atms-of-ms (N ∪ unmark M) = atms-of-ms N
    using atm-incl unfolding atms-of-ms-def lits-of-def by auto

  then have total-over-m I (N ∪ (λa. {#lit-of a#}) ` (set M))
    using tot unfolding total-over-m-def by auto
  then have I ⊨s (λa. {#lit-of a#}) ` (set M)
    using all-decomposition-implies-propagated-lits-are-implied[OF inv] cons I
    unfolding true-clss-clss-def l0 by auto
  then have IM: I ⊨s unmark M by auto
  {
    fix K
    assume K ∈# D
    then have −K ∈ lits-of M
      by (auto split: split-if-asm
        intro: allE[OF D[unfolded true-annots-def Ball-def], of {#−K#}])
    then have −K ∈ I using IM true-clss-singleton-lit-of-implies-incl by fastforce
  }
  then have ¬ I ⊨ D using cons unfolding true-cls-def consistent-interp-def by auto
  then show False using I-D by blast
qed

lemma dpll_W-same-clauses:
  assumes dpll_W S S'
  shows clauses S = clauses S'
  using assms by (induct rule: dpll_W.induct, auto)

lemma rtranclp-dpll_W-inv:
  assumes rtranclp dpll_W S S'
  and inv: all-decomposition-implies-m (clauses S) (get-all-marked-decomposition (trail S))
  and atm-incl: atm-of ` lits-of (trail S) ⊆ atms-of-msu (clauses S)
  and consistent-interp (lits-of (trail S))
  and no-dup (trail S)
  shows all-decomposition-implies-m (clauses S') (get-all-marked-decomposition (trail S'))
  and atm-of ` lits-of (trail S') ⊆ atms-of-msu (clauses S')
  and clauses S = clauses S'
  and consistent-interp (lits-of (trail S'))
  and no-dup (trail S')
  using assms
proof (induct rule: rtranclp-induct)
  case base
  show
    all-decomposition-implies-m (clauses S) (get-all-marked-decomposition (trail S)) and
    atm-of ` lits-of (trail S) ⊆ atms-of-msu (clauses S) and
    clauses S = clauses S and
    consistent-interp (lits-of (trail S)) and
    no-dup (trail S) using assms by auto
next
  case (step S' S'') note dpll_WStar = this(1) and IH = this(3,4,5,6,7) and
    dpll_W = this(2)
  moreover
    assume
```

inv: *all-decomposition-implies-m* (*clauses S*) (*get-all-marked-decomposition* (*trail S*)) **and**
atm-incl: *atm-of* ‘ *lits-of* (*trail S*) ⊆ *atms-of-msu* (*clauses S*) **and**
cons: *consistent-interp* (*lits-of* (*trail S*)) **and**
no-dup (*trail S*)
ultimately have *decomp*: *all-decomposition-implies-m* (*clauses S'*)
(*get-all-marked-decomposition* (*trail S'*)) **and**
atm-incl': *atm-of* ‘ *lits-of* (*trail S'*) ⊆ *atms-of-msu* (*clauses S'*) **and**
snd: *clauses S* = *clauses S'* **and**
cons': *consistent-interp* (*lits-of* (*trail S'*)) **and**
no-dup': *no-dup* (*trail S'*) **by** *blast+*
show *clauses S* = *clauses S''* **using** $dpll_W$*-same-clauses*[*OF* $dpll_W$] *snd* **by** *metis*

show *all-decomposition-implies-m* (*clauses S''*) (*get-all-marked-decomposition* (*trail S''*))
using $dpll_W$*-propagate-is-conclusion*[*OF* $dpll_W$] *decomp atm-incl'* **by** *auto*
show *atm-of* ‘ *lits-of* (*trail S''*) ⊆ *atms-of-msu* (*clauses S''*)
using $dpll_W$*-vars-in-snd-inv*[*OF* $dpll_W$] *atm-incl atm-incl'* **by** *auto*
show *no-dup* (*trail S''*) **using** $dpll_W$*-distinct-inv*[*OF* $dpll_W$] *no-dup'* $dpll_W$ **by** *auto*
show *consistent-interp* (*lits-of* (*trail S''*))
using *cons' no-dup'* $dpll_W$*-consistent-interp-inv*[*OF* $dpll_W$] **by** *auto*
qed

definition $dpll_W$*-all-inv S* ≡
(*all-decomposition-implies-m* (*clauses S*) (*get-all-marked-decomposition* (*trail S*))
∧ *atm-of* ‘ *lits-of* (*trail S*) ⊆ *atms-of-msu* (*clauses S*)
∧ *consistent-interp* (*lits-of* (*trail S*))
∧ *no-dup* (*trail S*))

lemma $dpll_W$*-all-inv-dest*[*dest*]:
assumes $dpll_W$*-all-inv S*
shows *all-decomposition-implies-m* (*clauses S*) (*get-all-marked-decomposition* (*trail S*))
and *atm-of* ‘ *lits-of* (*trail S*) ⊆ *atms-of-msu* (*clauses S*)
and *consistent-interp* (*lits-of* (*trail S*)) ∧ *no-dup* (*trail S*)
using *assms* **unfolding** $dpll_W$*-all-inv-def lits-of-def* **by** *auto*

lemma *rtranclp-*$dpll_W$*-all-inv*:
assumes *rtranclp* $dpll_W$ *S S'*
and $dpll_W$*-all-inv S*
shows $dpll_W$*-all-inv S'*
using *assms rtranclp-*$dpll_W$*-inv*[*OF assms*(*1*)] **unfolding** $dpll_W$*-all-inv-def lits-of-def* **by** *blast*

lemma $dpll_W$*-all-inv*:
assumes $dpll_W$ *S S'*
and $dpll_W$*-all-inv S*
shows $dpll_W$*-all-inv S'*
using *assms rtranclp-*$dpll_W$*-all-inv* **by** *blast*

lemma *rtranclp-*$dpll_W$*-inv-starting-from-0*:
assumes *rtranclp* $dpll_W$ *S S'*
and *inv*: *trail S* = []
shows $dpll_W$*-all-inv S'*
proof −
have $dpll_W$*-all-inv S*
using *assms* **unfolding** *all-decomposition-implies-def* $dpll_W$*-all-inv-def* **by** *auto*
then show *?thesis* **using** *rtranclp-*$dpll_W$*-all-inv*[*OF assms*(*1*)] **by** *blast*
qed

lemma $dpll_W$*-can-do-step*:
assumes *consistent-interp* (*set M*)
and *distinct M*
and *atm-of* ‘ (*set M*) ⊆ *atms-of-msu N*
shows *rtranclp* $dpll_W$ ([], *N*) (*map* (λ*M*. *Marked M* ()) *M*, *N*)
using *assms*
proof (*induct M*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons L M*)
then have *undefined-lit* (*map* (λ*M*. *Marked M* ()) *M*) *L*
unfolding *defined-lit-def consistent-interp-def* **by** *auto*
moreover have *atm-of L* ∈ *atms-of-msu N* **using** *Cons.prems*(*3*) **by** *auto*
ultimately have $dpll_W$ (*map* (λ*M*. *Marked M* ()) *M*, *N*) (*map* (λ*M*. *Marked M* ()) (*L* # *M*), *N*)
using $dpll_W$*.decided* **by** *auto*
moreover have *consistent-interp* (*set M*) **and** *distinct M* **and** *atm-of* ‘ *set M* ⊆ *atms-of-msu N*
using *Cons.prems* **unfolding** *consistent-interp-def* **by** *auto*
ultimately show *?case* **using** *Cons.hyps* **by** *auto*
qed

definition *conclusive-*$dpll_W$*-state* (*S*:: ′*v* $dpll_W$*-state*) ⟷
(*trail S* ⊨*asm clauses S* ∨ ((∀ *L* ∈ *set* (*trail S*). ¬*is-marked L*)
∧ (∃ *C* ∈# *clauses S*. *trail S* ⊨*as CNot C*)))

lemma $dpll_W$*-strong-completeness*:
assumes *set M* ⊨*sm N*
and *consistent-interp* (*set M*)
and *distinct M*
and *atm-of* ‘ (*set M*) ⊆ *atms-of-msu N*
shows $dpll_W{}^{**}$ ([], *N*) (*map* (λ*M*. *Marked M* ()) *M*, *N*)
and *conclusive-*$dpll_W$*-state* (*map* (λ*M*. *Marked M* ()) *M*, *N*)
proof −
show *rtranclp* $dpll_W$ ([], *N*) (*map* (λ*M*. *Marked M* ()) *M*, *N*) **using** $dpll_W$*-can-do-step assms* **by** *auto*
have *map* (λ*M*. *Marked M* ()) *M* ⊨*asm N* **using** *assms*(*1*) *true-annots-marked-true-cls* **by** *auto*
then show *conclusive-*$dpll_W$*-state* (*map* (λ*M*. *Marked M* ()) *M*, *N*)
unfolding *conclusive-*$dpll_W$*-state-def* **by** *auto*
qed

lemma $dpll_W$*-sound*:
assumes
rtranclp $dpll_W$ ([], *N*) (*M*, *N*) **and**
∀ *S*. ¬$dpll_W$ (*M*, *N*) *S*
shows *M* ⊨*asm N* ⟷ *satisfiable* (*set-mset N*) (**is** *?A* ⟷ *?B*)
proof
let *?M*′= *lits-of M*
assume *?A*
then have *?M*′ ⊨*sm N* **by** (*simp add*: *true-annots-true-cls*)
moreover have *consistent-interp ?M*′
using *rtranclp-*$dpll_W$*-inv-starting-from-0*[*OF assms*(*1*)] **by** *auto*
ultimately show *?B* **by** *auto*
next

```
  assume ?B
  show ?A
    proof (rule ccontr)
      assume n: ¬ ?A
      have (∃L. undefined-lit M L ∧ atm-of L ∈ atms-of-msu N) ∨ (∃D∈#N. M ⊨as CNot D)
        proof –
          obtain D :: 'a clause where D: D ∈# N and ¬ M ⊨a D
            using n unfolding true-annots-def Ball-def by auto
          then have (∃L. undefined-lit M L ∧ atm-of L ∈ atms-of D) ∨ M ⊨as CNot D
             unfolding true-annots-def Ball-def CNot-def true-annot-def
             using atm-of-lit-in-atms-of true-annot-iff-marked-or-true-lit true-cls-def by blast
          then show ?thesis
            by (metis Bex-mset-def D atms-of-atms-of-ms-mono mem-set-mset-iff rev-subsetD)
        qed
      moreover {
        assume ∃L. undefined-lit M L ∧ atm-of L ∈ atms-of-msu N
        then have False using assms(2) decided by fastforce
      }
      moreover {
        assume ∃D∈#N. M ⊨as CNot D
        then obtain D where DN: D ∈# N and MD: M ⊨as CNot D by auto
        {
          assume ∀l ∈ set M. ¬ is-marked l
          moreover have dpll_W-all-inv ([], N)
            using assms unfolding all-decomposition-implies-def dpll_W-all-inv-def by auto
          ultimately have unsatisfiable (set-mset N)
            using only-propagated-vars-unsat[of M D set-mset N] DN MD
            rtranclp-dpll_W-all-inv[OF assms(1)] by force
          then have False using ‹?B› by blast
        }
        moreover {
          assume l: ∃l ∈ set M. is-marked l
          then have False
            using backtrack[of (M, N) - - - D ] DN MD assms(2)
              backtrack-split-some-is-marked-then-snd-has-hd[OF l]
            by (metis backtrack-split-snd-hd-marked fst-conv list.distinct(1) list.sel(1) snd-conv)
        }
        ultimately have False by blast
      }
      ultimately show False by blast
     qed
qed
```

4.3 Termination

```
definition dpll_W-mes M n =
  map (λl. if is-marked l then 2 else (1::nat)) (rev M) @ replicate (n − length M) 3

lemma length-dpll_W-mes:
  assumes length M ≤ n
  shows length (dpll_W-mes M n) = n
  using assms unfolding dpll_W-mes-def by auto

lemma distinctcard-atm-of-lit-of-eq-length:
  assumes no-dup S
  shows card (atm-of ` lits-of S) = length S
```

using *assms* **by** (*induct S*) (*auto simp add*: *image-image lits-of-def*)

lemma $dpll_W$*-card-decrease*:
  assumes *dpll*: $dpll_W$ *S S′* **and** *length* (*trail S′*) ≤ *card vars*
  and *length* (*trail S*) ≤ *card vars*
  shows ($dpll_W$*-mes* (*trail S′*) (*card vars*), $dpll_W$*-mes* (*trail S*) (*card vars*))
    ∈ *lexn* {(*a*, *b*). *a* < *b*} (*card vars*)
  using *assms*
proof (*induct rule*: $dpll_W$*.induct*)
  case (*propagate C L S*)
  have *m*: *map* (λ*l*. *if is-marked l then 2 else 1*) (*rev* (*trail S*))
      @ *replicate* (*card vars* − *length* (*trail S*)) *3*
    = *map* (λ*l*. *if is-marked l then 2 else 1*) (*rev* (*trail S*)) @ *3*
      # *replicate* (*card vars* − *Suc* (*length* (*trail S*))) *3*
    using *propagate.prems*[*simplified*] **using** *Suc-diff-le* **by** *fastforce*
  then show *?case*
    using *propagate.prems*(*1*) **unfolding** $dpll_W$*-mes-def* **by** (*fastforce simp add*: *lexn-conv assms*(*2*))
next
  case (*decided S L*)
  have *m*: *map* (λ*l*. *if is-marked l then 2 else 1*) (*rev* (*trail S*))
     @ *replicate* (*card vars* − *length* (*trail S*)) *3*
   = *map* (λ*l*. *if is-marked l then 2 else 1*) (*rev* (*trail S*)) @ *3*
    # *replicate* (*card vars* − *Suc* (*length* (*trail S*))) *3*
    using *decided.prems*[*simplified*] **using** *Suc-diff-le* **by** *fastforce*
  then show *?case*
    using *decided.prems* **unfolding** $dpll_W$*-mes-def* **by** (*force simp add*: *lexn-conv assms*(*2*))
next
  case (*backtrack S M′ L M D*)
  have *L*: *is-marked L* **using** *backtrack.hyps*(*2*) **by** *auto*
  have *S*: *trail S* = *M′* @ *L* # *M*
    using *backtrack.hyps*(*1*) *backtrack-split-list-eq*[*of trail S*] **by** *auto*
  show *?case*
    using *backtrack.prems L* **unfolding** $dpll_W$*-mes-def S* **by** (*fastforce simp add*: *lexn-conv assms*(*2*))
qed

Proposition theorem 2.8.7 page 73 of CW

lemma $dpll_W$*-card-decrease′*:
  assumes *dpll*: $dpll_W$ *S S′*
  and *atm-incl*: *atm-of* ‘ *lits-of* (*trail S*) ⊆ *atms-of-msu* (*clauses S*)
  and *no-dup*: *no-dup* (*trail S*)
  shows ($dpll_W$*-mes* (*trail S′*) (*card* (*atms-of-msu* (*clauses S′*))),
      $dpll_W$*-mes* (*trail S*) (*card* (*atms-of-msu* (*clauses S*)))) ∈ *lex* {(*a*, *b*). *a* < *b*}
proof −
  have *finite* (*atms-of-msu* (*clauses S*)) **unfolding** *atms-of-ms-def* **by** *auto*
  then have *1*: *length* (*trail S*) ≤ *card* (*atms-of-msu* (*clauses S*))
    using *distinctcard-atm-of-lit-of-eq-length*[*OF no-dup*] *atm-incl card-mono* **by** *metis*

  moreover
    have *no-dup′*: *no-dup* (*trail S′*) **using** *dpll* $dpll_W$*-distinct-inv no-dup* **by** *blast*
    have *SS′*: *clauses S′* = *clauses S* **using** *dpll* **by** (*auto dest*!: $dpll_W$*-same-clauses*)
    have *atm-incl′*: *atm-of* ‘ *lits-of* (*trail S′*) ⊆ *atms-of-msu* (*clauses S′*)
      using *atm-incl dpll* $dpll_W$*-vars-in-snd-inv*[*OF dpll*] **by** *force*
    have *finite* (*atms-of-msu* (*clauses S′*))
      unfolding *atms-of-ms-def* **by** *auto*
    then have *2*: *length* (*trail S′*) ≤ *card* (*atms-of-msu* (*clauses S*))

```
    using distinctcard-atm-of-lit-of-eq-length[OF no-dup'] atm-incl' card-mono SS' by metis

  ultimately have (dpll_W-mes (trail S') (card (atms-of-msu (clauses S))),
      dpll_W-mes (trail S) (card (atms-of-msu (clauses S))))
    ∈ lexn {(a, b). a < b} (card (atms-of-msu (clauses S)))
    using dpll_W-card-decrease[OF assms(1), of atms-of-msu (clauses S)] by blast
  then have (dpll_W-mes (trail S') (card (atms-of-msu (clauses S))),
        dpll_W-mes (trail S) (card (atms-of-msu (clauses S)))) ∈ lex {(a, b). a < b}
    unfolding lex-def by auto
  then show (dpll_W-mes (trail S') (card (atms-of-msu (clauses S'))),
        dpll_W-mes (trail S) (card (atms-of-msu (clauses S)))) ∈ lex {(a, b). a < b}
    using dpll_W-same-clauses[OF assms(1)] by auto
qed

lemma wf-lexn: wf (lexn {(a, b). (a::nat) < b} (card (atms-of-msu (clauses S))))
proof -
  have m: {(a, b). a < b} = measure id by auto
  show ?thesis apply (rule wf-lexn) unfolding m by auto
qed

lemma dpll_W-wf:
  wf {(S', S). dpll_W-all-inv S ∧ dpll_W S S'}
  apply (rule wf-wf-if-measure'[OF wf-lex-less, of - -
        λS. dpll_W-mes (trail S) (card (atms-of-msu (clauses S)))])
  using dpll_W-card-decrease' by fast


lemma dpll_W-tranclp-star-commute:
  {(S', S). dpll_W-all-inv S ∧ dpll_W S S'}^+ = {(S', S). dpll_W-all-inv S ∧ tranclp dpll_W S S'}
    (is ?A = ?B)
proof
  { fix S S'
    assume (S, S') ∈ ?A
    then have (S, S') ∈ ?B
      by (induct rule: trancl.induct, auto)
  }
  then show ?A ⊆ ?B by blast
  { fix S S'
    assume (S, S') ∈ ?B
    then have dpll_W^++ S' S and dpll_W-all-inv S' by auto
    then have (S, S') ∈ ?A
      proof (induct rule: tranclp.induct)
        case r-into-trancl
        then show ?case by (simp-all add: r-into-trancl')
      next
        case (trancl-into-trancl S S' S'')
        then have (S', S) ∈ {a. case a of (S', S) ⇒ dpll_W-all-inv S ∧ dpll_W S S'}^+ by blast
        moreover have dpll_W-all-inv S'
          using rtranclp-dpll_W-all-inv[OF tranclp-into-rtranclp[OF trancl-into-trancl.hyps(1)]]
          trancl-into-trancl.prems by auto
        ultimately have (S'', S') ∈ {(pa, p). dpll_W-all-inv p ∧ dpll_W p pa}^+
          using ‹dpll_W-all-inv S'› trancl-into-trancl.hyps(3) by blast
        then show ?case
          using ‹(S', S) ∈ {a. case a of (S', S) ⇒ dpll_W-all-inv S ∧ dpll_W S S'}^+› by auto
      qed
```

```
  }
  then show ?B ⊆ ?A by blast
qed

lemma dpll_W-wf-tranclp: wf {(S', S). dpll_W-all-inv S ∧ dpll_W^++ S S'}
  unfolding dpll_W-tranclp-star-commute[symmetric] by (simp add: dpll_W-wf wf-trancl)

lemma dpll_W-wf-plus:
  shows wf {(S', ([], N))| S'. dpll_W^++ ([], N) S'}  (is wf ?P)
  apply (rule wf-subset[OF dpll_W-wf-tranclp, of ?P])
  using assms unfolding dpll_W-all-inv-def by auto
```

4.4 Final States

```
lemma dpll_W-no-more-step-is-a-conclusive-state:
  assumes ∀ S'. ¬dpll_W S S'
  shows conclusive-dpll_W-state S
proof –
  have vars: ∀ s ∈ atms-of-msu (clauses S). s ∈ atm-of ‘ lits-of (trail S)
    proof (rule ccontr)
      assume ¬ (∀ s∈atms-of-msu (clauses S). s ∈ atm-of ‘ lits-of (trail S))
      then obtain L where
        L-in-atms: L ∈ atms-of-msu (clauses S) and
        L-notin-trail: L ∉ atm-of ‘ lits-of (trail S) by metis
      obtain L' where L': atm-of L' = L by (meson literal.sel(2))
      then have undefined-lit (trail S) L'
        unfolding Marked-Propagated-in-iff-in-lits-of by (metis L-notin-trail atm-of-uminus imageI)
      then show False using dpll_W.decided assms(1) L-in-atms L' by blast
    qed
  show ?thesis
    proof (rule ccontr)
      assume not-final: ¬ ?thesis
      then have
        ¬ trail S ⊨asm clauses S and
        (∃ L∈set (trail S). is-marked L) ∨ (∀ C∈#clauses S. ¬trail S ⊨as CNot C)
        unfolding conclusive-dpll_W-state-def by auto
      moreover {
        assume ∃ L∈set (trail S). is-marked L
        then obtain L M' M where L: backtrack-split (trail S) = (M', L # M)
          using backtrack-split-some-is-marked-then-snd-has-hd by blast
        obtain D where D ∈# clauses S and ¬ trail S ⊨a D
          using ‹¬ trail S ⊨asm clauses S› unfolding true-annots-def by auto
        then have ∀ s∈atms-of-ms {D}. s ∈ atm-of ‘ lits-of (trail S)
          using vars unfolding atms-of-ms-def by auto
        then have trail S ⊨as CNot D
          using all-variables-defined-not-imply-cnot[of D] ‹¬ trail S ⊨a D› by auto
        moreover have is-marked L
          using L by (metis backtrack-split-snd-hd-marked list.distinct(1) list.sel(1) snd-conv)
        ultimately have False
          using assms(1) dpll_W.backtrack L ‹D ∈# clauses S› ‹trail S ⊨as CNot D› by blast
      }
      moreover {
        assume tr: ∀ C∈#clauses S. ¬trail S ⊨as CNot C
        obtain C where C-in-cls: C ∈# clauses S and trC: ¬ trail S ⊨a C
          using ‹¬ trail S ⊨asm clauses S› unfolding true-annots-def by auto
        have ∀ s∈atms-of-ms {C}. s ∈ atm-of ‘ lits-of (trail S)
```

```
        using vars ‹C ∈# clauses S› unfolding atms-of-ms-def by auto
      then have trail S ⊨as CNot C
        by (meson C-in-cls tr trC all-variables-defined-not-imply-cnot)
      then have False using tr C-in-cls by auto
    }
    ultimately show False by blast
  qed
qed
```

```
lemma dpll_W-conclusive-state-correct:
  assumes dpll_W** ([], N) (M, N) and conclusive-dpll_W-state (M, N)
  shows M ⊨asm N ⟷ satisfiable (set-mset N) (is ?A ⟷ ?B)
proof
  let ?M'= lits-of M
  assume ?A
  then have ?M' ⊨sm N by (simp add: true-annots-true-cls)
  moreover have consistent-interp ?M'
    using rtranclp-dpll_W-inv-starting-from-0[OF assms(1)] by auto
  ultimately show ?B by auto
next
  assume ?B
  show ?A
    proof (rule ccontr)
      assume n: ¬ ?A
      have no-mark: ∀ L∈set M. ¬ is-marked L  ∃ C ∈# N. M ⊨as CNot C
        using n assms(2) unfolding conclusive-dpll_W-state-def by auto
      moreover obtain D where DN: D ∈# N and MD: M ⊨as CNot D using no-mark by auto
      ultimately have unsatisfiable (set-mset N)
        using only-propagated-vars-unsat rtranclp-dpll_W-all-inv[OF assms(1)]
        unfolding dpll_W-all-inv-def by force
      then show False using ‹?B› by blast
    qed
qed
```

4.5 Link with NOT's DPLL

```
interpretation dpll_W-NOT: dpll-with-backtrack .

lemma state-eq_NOT-iff-eq[iff, simp]: dpll_W-NOT.state-eq_NOT S T ⟷ S = T
  unfolding dpll_W-NOT.state-eq_NOT-def by (cases S, cases T) auto

declare dpll_W-NOT.state-simp_NOT[simp del]

lemma dpll_W-dpll_W-bj:
  assumes inv: dpll_W-all-inv S and dpll: dpll_W S T
  shows dpll_W-NOT.dpll-bj S T
  using dpll inv
  apply (induction rule: dpll_W.induct)
     using dpll_W-NOT.dpll-bj.simps apply fastforce
    using dpll_W-NOT.bj-decide_NOT apply fastforce
  apply (frule dpll_W-NOT.backtrack.intros[of - -  - - -], simp-all)
  apply (rule dpll_W-NOT.dpll-bj.bj-backjump)
  apply (rule dpll_W-NOT.backtrack-is-backjump'',
    simp-all add: dpll_W-all-inv-def)
  done
```

lemma *dpll*$_W$*-bj-dpll*:
  assumes *inv*: *dpll*$_W$*-all-inv S* **and** *dpll*: *dpll*$_W$*-*$_{NOT}$*.dpll-bj S T*
  shows *dpll*$_W$ *S T*
  using *dpll*
  apply (*induction rule*: *dpll*$_W$*-*$_{NOT}$*.dpll-bj.induct*)
    apply (*elim dpll*$_W$*-*$_{NOT}$*.decide*$_{NOT}$*E*, *cases S*)
    using *decided* **apply** *fastforce*
   apply (*elim dpll*$_W$*-*$_{NOT}$*.propagate*$_{NOT}$*E*, *cases S*)
   using *dpll*$_W$*.simps* **apply** *fastforce*
  apply (*elim dpll*$_W$*-*$_{NOT}$*.backjumpE*, *cases S*)
  by (*simp add*: *dpll*$_W$*.simps dpll-with-backtrack.backtrack.simps*)

lemma *rtranclp-dpll*$_W$*-rtranclp-dpll*$_W$*-*$_{NOT}$:
  assumes *dpll*$_W$** *S T* **and** *dpll*$_W$*-all-inv S*
  shows *dpll*$_W$*-*$_{NOT}$*.dpll-bj*** *S T*
  using *assms* **apply** (*induction*)
   apply *simp*
  by (*auto intro*: *rtranclp-dpll*$_W$*-all-inv dpll*$_W$*-dpll*$_W$*-bj rtranclp.rtrancl-into-rtrancl*)

lemma *rtranclp-dpll-rtranclp-dpll*$_W$:
  assumes *dpll*$_W$*-*$_{NOT}$*.dpll-bj*** *S T* **and** *dpll*$_W$*-all-inv S*
  shows *dpll*$_W$** *S T*
  using *assms* **apply** (*induction*)
   apply *simp*
  by (*auto intro*: *dpll*$_W$*-bj-dpll rtranclp.rtrancl-into-rtrancl rtranclp-dpll*$_W$*-all-inv*)

lemma *dpll-conclusive-state-correctness*:
  assumes *dpll*$_W$*-*$_{NOT}$*.dpll-bj*** ([], *N*) (*M*, *N*) **and** *conclusive-dpll*$_W$*-state* (*M*, *N*)
  shows *M* ⊨*asm N* ⟷ *satisfiable* (*set-mset N*)
proof −
  have *dpll*$_W$*-all-inv* ([], *N*)
    unfolding *dpll*$_W$*-all-inv-def* **by** *auto*
  show *?thesis*
    apply (*rule dpll*$_W$*-conclusive-state-correct*)
      apply (*simp add*: ‹*dpll*$_W$*-all-inv* ([], *N*)› *assms*(*1*) *rtranclp-dpll-rtranclp-dpll*$_W$)
    using *assms*(*2*) **by** *simp*
qed

end
theory *CDCL-W-Level*
imports *Partial-Annotated-Clausal-Logic*
begin

4.5.1 Level of literals and clauses

Getting the level of a variable, implies that the list has to be reversed. Here is the funtion after reversing.

fun *get-rev-level* :: (*'v*, *nat*, *'a*) *ann-literals* ⇒ *nat* ⇒ *'v literal* ⇒ *nat* **where**
get-rev-level [] - - = *0* |
get-rev-level (*Marked l level* # *Ls*) *n L* =
  (*if atm-of l* = *atm-of L then level else get-rev-level Ls level L*) |
get-rev-level (*Propagated l* - # *Ls*) *n L* =
  (*if atm-of l* = *atm-of L then n else get-rev-level Ls n L*)

abbreviation *get-level M L* ≡ *get-rev-level* (*rev M*) *0 L*

lemma *get-rev-level-uminus*[*simp*]: *get-rev-level M n*$(-L)$ = *get-rev-level M n L*
  by (*induct arbitrary*: *n rule*: *get-rev-level.induct*) *auto*

lemma *atm-of-notin-get-rev-level-eq-0*[*simp*]:
  assumes *atm-of L* $\notin$ *atm-of* ‘ *lits-of M*
  shows *get-rev-level M n L* = *0*
  using *assms* **by** (*induct M arbitrary*: *n rule*: *ann-literal-list-induct*) *auto*

lemma *get-rev-level-ge-0-atm-of-in*:
  assumes *get-rev-level M n L* > *n*
  shows *atm-of L* $\in$ *atm-of* ‘ *lits-of M*
  using *assms* **by** (*induct M arbitrary*: *n rule*: *ann-literal-list-induct*) *fastforce+*

In *get-rev-level* (resp. *get-level*), the beginning (resp. the end) can be skipped if the literal is not in the beginning (resp. the end).

lemma *get-rev-level-skip*[*simp*]:
  assumes *atm-of L* $\notin$ *atm-of* ‘ *lits-of M*
  shows *get-rev-level* (*M* @ *Marked K i* # *M'*) *n L* = *get-rev-level* (*Marked K i* # *M'*) *i L*
  using *assms* **by** (*induct M arbitrary*: *n i rule*: *ann-literal-list-induct*) *auto*

lemma *get-rev-level-notin-end*[*simp*]:
  assumes *atm-of L* $\notin$ *atm-of* ‘ *lits-of M'*
  shows *get-rev-level* (*M* @ *M'*) *n L* = *get-rev-level M n L*
  using *assms* **by** (*induct M arbitrary*: *n rule*: *ann-literal-list-induct*) *auto*

If the literal is at the beginning, then the end can be skipped

lemma *get-rev-level-skip-end*[*simp*]:
  assumes *atm-of L* $\in$ *atm-of* ‘ *lits-of M*
  shows *get-rev-level* (*M* @ *M'*) *n L* = *get-rev-level M n L*
  using *assms* **by** (*induct arbitrary*: *n rule*: *ann-literal-list-induct*) *auto*

lemma *get-level-skip-beginning*:
  assumes *atm-of L'* $\neq$ *atm-of* (*lit-of K*)
  shows *get-level* (*K* # *M*) *L'* = *get-level M L'*
  using *assms* **by** *auto*

lemma *get-level-skip-beginning-not-marked-rev*:
  assumes *atm-of L* $\notin$ *atm-of* ‘ *lit-of* ‘(*set S*)
  and $\forall$ *s*$\in$*set S*. $\neg$*is-marked s*
  shows *get-level* (*M* @ *rev S*) *L* = *get-level M L*
  using *assms* **by** (*induction S rule*: *ann-literal-list-induct*) *auto*

lemma *get-level-skip-beginning-not-marked*[*simp*]:
  assumes *atm-of L* $\notin$ *atm-of* ‘ *lit-of* ‘(*set S*)
  and $\forall$ *s*$\in$*set S*. $\neg$*is-marked s*
  shows *get-level* (*M* @ *S*) *L* = *get-level M L*
  using *get-level-skip-beginning-not-marked-rev*[*of L rev S M*] *assms* **by** *auto*

lemma *get-rev-level-skip-beginning-not-marked*[*simp*]:
  assumes *atm-of L* $\notin$ *atm-of* ‘ *lit-of* ‘(*set S*)
  and $\forall$ *s*$\in$*set S*. $\neg$*is-marked s*
  shows *get-rev-level* (*rev S* @ *rev M*) *0 L* = *get-level M L*
  using *get-level-skip-beginning-not-marked-rev*[*of L rev S M*] *assms* **by** *auto*

lemma *get-level-skip-in-all-not-marked*:
  fixes M :: $('a, nat, 'b)$ *ann-literal list* **and** L :: *'a literal*
  assumes $\forall\, m{\in}set\ M.\ \neg\ is\text{-}marked\ m$
  and $atm\text{-}of\ L \in atm\text{-}of\ \text{‘}\ lit\text{-}of\ \text{‘}\ (set\ M)$
  shows $get\text{-}rev\text{-}level\ M\ n\ L = n$
  using *assms* **by** (*induction M rule*: *ann-literal-list-induct*) *auto*

lemma *get-level-skip-all-not-marked*[*simp*]:
  fixes M
  defines $M' \equiv rev\ M$
  assumes $\forall\, m{\in}set\ M.\ \neg\ is\text{-}marked\ m$
  shows $get\text{-}level\ M\ L = 0$
proof −
  have M: $M = rev\ M'$
    unfolding *M′-def* **by** *auto*
  show *?thesis*
    using *assms* **unfolding** M **by** (*induction M′ rule*: *ann-literal-list-induct*) *auto*
qed

abbreviation $MMax\ M \equiv Max\ (set\text{-}mset\ M)$

the $\{\#0{::}'a\#\}$ is there to ensures that the set is not empty.

definition *get-maximum-level* :: $('a, nat, 'b)$ *ann-literal list* $\Rightarrow$ *'a literal multiset* $\Rightarrow$ *nat*
  where
$get\text{-}maximum\text{-}level\ M\ D = MMax\ (\{\#0\#\} + image\text{-}mset\ (get\text{-}level\ M)\ D)$

lemma *get-maximum-level-ge-get-level*:
  $L \in\#\ D \Longrightarrow get\text{-}maximum\text{-}level\ M\ D \geq get\text{-}level\ M\ L$
  unfolding *get-maximum-level-def* **by** *auto*

lemma *get-maximum-level-empty*[*simp*]:
  $get\text{-}maximum\text{-}level\ M\ \{\#\} = 0$
  unfolding *get-maximum-level-def* **by** *auto*

lemma *get-maximum-level-exists-lit-of-max-level*:
  $D \neq \{\#\} \Longrightarrow \exists\, L{\in}\#\ D.\ get\text{-}level\ M\ L = get\text{-}maximum\text{-}level\ M\ D$
  unfolding *get-maximum-level-def*
  apply (*induct D*)
   apply *simp*
  by (*rename-tac D x*, *case-tac* $D = \{\#\}$) (*auto simp add*: *max-def*)

lemma *get-maximum-level-empty-list*[*simp*]:
  $get\text{-}maximum\text{-}level\ []\ D = 0$
  unfolding *get-maximum-level-def* **by** (*simp add*: *image-constant-conv*)

lemma *get-maximum-level-single*[*simp*]:
  $get\text{-}maximum\text{-}level\ M\ \{\#L\#\} = get\text{-}level\ M\ L$
  unfolding *get-maximum-level-def* **by** *simp*

lemma *get-maximum-level-plus*:
  $get\text{-}maximum\text{-}level\ M\ (D + D') = max\ (get\text{-}maximum\text{-}level\ M\ D)\ (get\text{-}maximum\text{-}level\ M\ D')$
  by (*induct D*) (*auto simp add*: *get-maximum-level-def*)

lemma *get-maximum-level-exists-lit*:

assumes *n*: *n* > *0*
and *max*: *get-maximum-level M D* = *n*
shows ∃ *L* ∈#*D*. *get-level M L* = *n*
proof −
have *f*: *finite* (*insert 0* ((λ*L*. *get-level M L*) ‘ *set-mset D*)) **by** *auto*
then have *n* ∈ ((λ*L*. *get-level M L*) ‘ *set-mset D*)
using *n max Max-in*[*OF f*] **unfolding** *get-maximum-level-def* **by** *simp*
then show ∃ *L* ∈# *D*. *get-level M L* = *n* **by** *auto*
qed

lemma *get-maximum-level-skip-first*[*simp*]:
assumes *atm-of L* ∉ *atms-of D*
shows *get-maximum-level* (*Propagated L C* # *M*) *D* = *get-maximum-level M D*
using *assms* **unfolding** *get-maximum-level-def atms-of-def*
atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set
by (*smt atm-of-in-atm-of-set-in-uminus get-level-skip-beginning image-iff ann-literal.sel*(*2*)
multiset.map-cong0)

lemma *get-maximum-level-skip-beginning*:
assumes *DH*: *atms-of D* ⊆ *atm-of* ‘*lits-of H*
shows *get-maximum-level* (*c* @ *Marked Kh i* # *H*) *D* = *get-maximum-level H D*
proof −
have (*get-rev-level* (*rev H* @ *Marked Kh i* # *rev c*) *0*) ‘ *set-mset D*
= (*get-rev-level* (*rev H*) *0*) ‘ *set-mset D*
using *DH* **unfolding** *atms-of-def*
by (*metis* (*no-types*, *lifting*) *get-rev-level-skip-end image-cong image-subset-iff lits-of-rev*)+
then show *?thesis* **using** *DH* **unfolding** *get-maximum-level-def* **by** *auto*
qed

lemma *get-maximum-level-D-single-propagated*:
get-maximum-level [*Propagated x21 x22*] *D* = *0*
proof −
have *A*: *insert 0* ((λ*L*. *0*) ‘ (*set-mset D* ∩ {*L*. *atm-of x21* = *atm-of L*})
∪ (λ*L*. *0*) ‘ (*set-mset D* ∩ {*L*. *atm-of x21* ≠ *atm-of L*})) = {*0*}
by *auto*
show *?thesis* **unfolding** *get-maximum-level-def* **by** (*simp add*: *A*)
qed

lemma *get-maximum-level-skip-notin*:
assumes *D*: ∀ *L*∈#*D*. *atm-of L* ∈ *atm-of* ‘*lits-of M*
shows *get-maximum-level M D* = *get-maximum-level* (*Propagated x21 x22* # *M*) *D*
proof −
have *A*: (*get-rev-level* (*rev M* @ [*Propagated x21 x22*]) *0*) ‘ *set-mset D*
= (*get-rev-level* (*rev M*) *0*) ‘ *set-mset D*
using *D* **by** (*auto intro*!: *image-cong simp add*: *lits-of-def*)
show *?thesis* **unfolding** *get-maximum-level-def* **by** (*auto simp*: *A*)
qed

lemma *get-maximum-level-skip-un-marked-not-present*:
assumes ∀ *L*∈#*D*. *atm-of L* ∈ *atm-of* ‘ *lits-of aa* **and**
∀ *m*∈*set M*. ¬ *is-marked m*
shows *get-maximum-level aa D* = *get-maximum-level* (*M* @ *aa*) *D*
using *assms* **by** (*induction M rule*: *ann-literal-list-induct*)
(*auto intro*!: *get-maximum-level-skip-notin*[*of D* - @ *aa*] *simp add*: *image-Un*)

fun *get-maximum-possible-level*:: ('*b*, *nat*, '*c*) *ann-literal list* ⇒ *nat* **where**
get-maximum-possible-level [] = *0* |
get-maximum-possible-level (*Marked K i* # *l*) = *max i* (*get-maximum-possible-level l*) |
get-maximum-possible-level (*Propagated - -* # *l*) = *get-maximum-possible-level l*

lemma *get-maximum-possible-level-append*[*simp*]:
get-maximum-possible-level (*M*@*M'*)
= *max* (*get-maximum-possible-level M*) (*get-maximum-possible-level M'*)
by (*induct M rule*: *ann-literal-list-induct*) *auto*

lemma *get-maximum-possible-level-rev*[*simp*]:
get-maximum-possible-level (*rev M*) = *get-maximum-possible-level M*
by (*induct M rule*: *ann-literal-list-induct*) *auto*

lemma *get-maximum-possible-level-ge-get-rev-level*:
max (*get-maximum-possible-level M*) *i* ≥ *get-rev-level M i L*
by (*induct M arbitrary*: *i rule*: *ann-literal-list-induct*) (*auto simp add*: *le-max-iff-disj*)

lemma *get-maximum-possible-level-ge-get-level*[*simp*]:
get-maximum-possible-level M ≥ *get-level M L*
using *get-maximum-possible-level-ge-get-rev-level*[*of rev - 0*] **by** *auto*

lemma *get-maximum-possible-level-ge-get-maximum-level*[*simp*]:
get-maximum-possible-level M ≥ *get-maximum-level M D*
using *get-maximum-level-exists-lit-of-max-level* **unfolding** *Bex-mset-def*
by (*metis get-maximum-level-empty get-maximum-possible-level-ge-get-level le0*)

fun *get-all-mark-of-propagated* **where**
get-all-mark-of-propagated [] = [] |
get-all-mark-of-propagated (*Marked - -* # *L*) = *get-all-mark-of-propagated L* |
get-all-mark-of-propagated (*Propagated - mark* # *L*) = *mark* # *get-all-mark-of-propagated L*

lemma *get-all-mark-of-propagated-append*[*simp*]:
get-all-mark-of-propagated (*A* @ *B*) = *get-all-mark-of-propagated A* @ *get-all-mark-of-propagated B*
by (*induct A rule*: *ann-literal-list-induct*) *auto*

4.5.2 Properties about the levels

fun *get-all-levels-of-marked* :: ('*b*, '*a*, '*c*) *ann-literal list* ⇒ '*a list* **where**
get-all-levels-of-marked [] = [] |
get-all-levels-of-marked (*Marked l level* # *Ls*) = *level* # *get-all-levels-of-marked Ls* |
get-all-levels-of-marked (*Propagated - -* # *Ls*) = *get-all-levels-of-marked Ls*

lemma *get-all-levels-of-marked-nil-iff-not-is-marked*:
get-all-levels-of-marked xs = [] ⟷ (∀ *x* ∈ *set xs*. ¬*is-marked x*)
using *assms* **by** (*induction xs rule*: *ann-literal-list-induct*) *auto*

lemma *get-all-levels-of-marked-cons*:
get-all-levels-of-marked (*a* # *b*) =
(*if is-marked a then* [*level-of a*] *else* []) @ *get-all-levels-of-marked b*
by (*cases a*) *simp-all*

lemma *get-all-levels-of-marked-append*[*simp*]:
get-all-levels-of-marked (*a* @ *b*) = *get-all-levels-of-marked a* @ *get-all-levels-of-marked b*
by (*induct a*) (*simp-all add*: *get-all-levels-of-marked-cons*)

lemma *in-get-all-levels-of-marked-iff-decomp*:
  $i \in set\ (get\text{-}all\text{-}levels\text{-}of\text{-}marked\ M) \longleftrightarrow (\exists\, c\ K\ c'.\ M = c\ @\ Marked\ K\ i\ \#\ c')$ (**is** $?A \longleftrightarrow ?B$)
proof
  assume *?B*
  then show *?A* **by** *auto*
next
  assume *?A*
  then show *?B*
    apply (*induction M rule*: *ann-literal-list-induct*)
      apply *auto*[]
     apply (*metis append-Cons append-Nil get-all-levels-of-marked.simps*(*2*) *set-ConsD*)
    by (*metis append-Cons get-all-levels-of-marked.simps*(*3*))
qed

lemma *get-rev-level-less-max-get-all-levels-of-marked*:
  $get\text{-}rev\text{-}level\ M\ n\ L \le Max\ (set\ (n\ \#\ get\text{-}all\text{-}levels\text{-}of\text{-}marked\ M))$
  by (*induct M arbitrary*: *n rule*: *get-all-levels-of-marked.induct*)
     (*simp-all add*: *max.coboundedI2*)

lemma *get-rev-level-ge-min-get-all-levels-of-marked*:
  assumes $atm\text{-}of\ L \in atm\text{-}of\ `\ lits\text{-}of\ M$
  shows $get\text{-}rev\text{-}level\ M\ n\ L \ge Min\ (set\ (n\ \#\ get\text{-}all\text{-}levels\text{-}of\text{-}marked\ M))$
  using *assms* **by** (*induct M arbitrary*: *n rule*: *get-all-levels-of-marked.induct*)
    (*auto simp add*: *min-le-iff-disj*)

lemma *get-all-levels-of-marked-rev-eq-rev-get-all-levels-of-marked*[*simp*]:
  $get\text{-}all\text{-}levels\text{-}of\text{-}marked\ (rev\ M) = rev\ (get\text{-}all\text{-}levels\text{-}of\text{-}marked\ M)$
  by (*induct M rule*: *get-all-levels-of-marked.induct*)
     (*simp-all add*: *max.coboundedI2*)

lemma *get-maximum-possible-level-max-get-all-levels-of-marked*:
  $get\text{-}maximum\text{-}possible\text{-}level\ M = Max\ (insert\ 0\ (set\ (get\text{-}all\text{-}levels\text{-}of\text{-}marked\ M)))$
  by (*induct M rule*: *ann-literal-list-induct*) (*auto simp*: *insert-commute*)

lemma *get-rev-level-in-levels-of-marked*:
  $get\text{-}rev\text{-}level\ M\ n\ L \in \{0,\ n\} \cup set\ (get\text{-}all\text{-}levels\text{-}of\text{-}marked\ M)$
  by (*induction M arbitrary*: *n rule*: *ann-literal-list-induct*) (*force simp add*: *atm-of-eq-atm-of*)+

lemma *get-rev-level-in-atms-in-levels-of-marked*:
  $atm\text{-}of\ L \in atm\text{-}of\ `\ (lits\text{-}of\ M) \Longrightarrow get\text{-}rev\text{-}level\ M\ n\ L \in \{n\} \cup set\ (get\text{-}all\text{-}levels\text{-}of\text{-}marked\ M)$
  by (*induction M arbitrary*: *n rule*: *ann-literal-list-induct*) (*auto simp add*: *atm-of-eq-atm-of*)

lemma *get-all-levels-of-marked-no-marked*:
  $(\forall\, l{\in}set\ Ls.\ \neg\ is\text{-}marked\ l) \longleftrightarrow get\text{-}all\text{-}levels\text{-}of\text{-}marked\ Ls = []$
  by (*induction Ls*) (*auto simp add*: *get-all-levels-of-marked-cons*)

lemma *get-level-in-levels-of-marked*:
  $get\text{-}level\ M\ L \in \{0\} \cup set\ (get\text{-}all\text{-}levels\text{-}of\text{-}marked\ M)$
  using *get-rev-level-in-levels-of-marked*[*of rev M 0 L*] **by** *auto*

The zero is here to avoid empty-list issues with *last*:

lemma *get-level-get-rev-level-get-all-levels-of-marked*:
  assumes $atm\text{-}of\ L \notin atm\text{-}of\ `\ (lits\text{-}of\ M)$
  shows $get\text{-}level\ (K\ @\ M)\ L = get\text{-}rev\text{-}level\ (rev\ K)\ (last\ (0\ \#\ get\text{-}all\text{-}levels\text{-}of\text{-}marked\ (rev\ M)))$

L
using *assms*
proof (*induct M arbitrary*: *K*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a M*)
then have *H*: $\bigwedge K$. *get-level* (*K* @ *M*) *L*
= *get-rev-level* (*rev K*) (*last* (*0* # *get-all-levels-of-marked* (*rev M*))) *L*
by *auto*
have *get-level* ((*K* @ [*a*])@ *M*) *L*
= *get-rev-level* (*a* # *rev K*) (*last* (*0* # *get-all-levels-of-marked* (*rev M*))) *L*
using *H*[*of K* @ [*a*]] **by** *simp*
then show *?case* **using** *Cons*(*2*) **by** (*cases a*) *auto*
qed

lemma *get-rev-level-can-skip-correctly-ordered*:
assumes
no-dup M **and**
atm-of L $\notin$ *atm-of* ‘ (*lits-of M*) **and**
get-all-levels-of-marked M = *rev* [*Suc 0*..<*Suc* (*length* (*get-all-levels-of-marked M*))]
shows *get-rev-level* (*rev M* @ *K*) *0 L* = *get-rev-level K* (*length* (*get-all-levels-of-marked M*)) *L*
using *assms*
proof (*induct M arbitrary*: *K rule*: *ann-literal-list-induct*)
case *nil*
then show *?case* **by** *simp*
next
case (*marked L' i M K*)
then have
i: *i* = *Suc* (*length* (*get-all-levels-of-marked M*)) **and**
get-all-levels-of-marked M = *rev* [*Suc 0*..<*Suc* (*length* (*get-all-levels-of-marked M*))]
by *auto*
then have *get-rev-level* (*rev M* @ (*Marked L' i* # *K*)) *0 L*
= *get-rev-level* (*Marked L' i* # *K*) (*length* (*get-all-levels-of-marked M*)) *L*
using *marked* **by** *auto*
then show *?case* **using** *marked* **unfolding** *i* **by** *auto*
next
case (*proped L' D M K*)
then have *get-all-levels-of-marked M* = *rev* [*Suc 0*..<*Suc* (*length* (*get-all-levels-of-marked M*))]
by *auto*
then have *get-rev-level* (*rev M* @ (*Propagated L' D* # *K*)) *0 L*
= *get-rev-level* (*Propagated L' D* # *K*) (*length* (*get-all-levels-of-marked M*)) *L*
using *proped* **by** *auto*
then show *?case* **using** *proped* **by** *auto*
qed

lemma *get-level-skip-beginning-hd-get-all-levels-of-marked*:
assumes *atm-of L* $\notin$ *atm-of* ‘ *lits-of S*
and *get-all-levels-of-marked S* $\neq$ []
shows *get-level* (*M*@ *S*) *L* = *get-rev-level* (*rev M*) (*hd* (*get-all-levels-of-marked S*)) *L*
using *assms*
proof (*induction S arbitrary*: *M rule*: *ann-literal-list-induct*)
case *nil*
then show *?case* **by** (*auto simp add*: *lits-of-def*)
next

case (*marked K m*) **note** *notin* = *this*(*2*)
then show *?case* **by** (*auto simp add*: *lits-of-def*)
next
case (*proped L l*) **note** *IH* = *this*(*1*) **and** *L* = *this*(*2*) **and** *neq* = *this*(*3*)
show *?case* **using** *IH*[*of M*@[*Propagated L l*]] *L neq* **by** (*auto simp add*: *atm-of-eq-atm-of*)
qed

end
theory *CDCL-W*
imports *Partial-Annotated-Clausal-Logic List-More CDCL-W-Level Wellfounded-More*

begin
declare *set-mset-minus-replicate-mset*[*simp*]

lemma *Bex-set-set-Bex-set*[*iff*]: $(\exists x{\in}set\text{-}mset\ C.\ P) \longleftrightarrow (\exists x{\in}\#C.\ P)$
by *auto*

5 Weidenbach's CDCL

declare *upt.simps*(*2*)[*simp del*]

5.1 The State

locale $state_W$ =
fixes
trail :: *'st* ⇒ (*'v*, *nat*, *'v clause*) *ann-literals* **and**
init-clss :: *'st* ⇒ *'v clauses* **and**
learned-clss :: *'st* ⇒ *'v clauses* **and**
backtrack-lvl :: *'st* ⇒ *nat* **and**
conflicting :: *'st* ⇒*'v clause option* **and**

cons-trail :: (*'v*, *nat*, *'v clause*) *ann-literal* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒*'st* **and**
add-init-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
add-learned-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
remove-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
update-backtrack-lvl :: *nat* ⇒ *'st* ⇒ *'st* **and**
update-conflicting :: *'v clause option* ⇒ *'st* ⇒ *'st* **and**

init-state :: *'v clauses* ⇒ *'st* **and**
restart-state :: *'st* ⇒ *'st*
assumes
trail-cons-trail[*simp*]:
⋀*L st*. *undefined-lit* (*trail st*) (*lit-of L*) ⟹ *trail* (*cons-trail L st*) = *L* # *trail st* **and**
trail-tl-trail[*simp*]: ⋀*st*. *trail* (*tl-trail st*) = *tl* (*trail st*) **and**
trail-add-init-cls[*simp*]:
⋀*st C*. *no-dup* (*trail st*) ⟹ *trail* (*add-init-cls C st*) = *trail st* **and**
trail-add-learned-cls[*simp*]:
⋀*C st*. *no-dup* (*trail st*) ⟹ *trail* (*add-learned-cls C st*) = *trail st* **and**
trail-remove-cls[*simp*]:
⋀*C st*. *trail* (*remove-cls C st*) = *trail st* **and**
trail-update-backtrack-lvl[*simp*]: ⋀*st C*. *trail* (*update-backtrack-lvl C st*) = *trail st* **and**
trail-update-conflicting[*simp*]: ⋀*C st*. *trail* (*update-conflicting C st*) = *trail st* **and**

init-clss-cons-trail[*simp*]:

$\bigwedge$*M st. undefined-lit* (*trail st*) (*lit-of M*)$\Longrightarrow$ *init-clss* (*cons-trail M st*) = *init-clss st*
and
init-clss-tl-trail[*simp*]:
$\bigwedge$*st. init-clss* (*tl-trail st*) = *init-clss st* **and**
init-clss-add-init-cls[*simp*]:
$\bigwedge$*st C. no-dup* (*trail st*) $\Longrightarrow$ *init-clss* (*add-init-cls C st*) = $\{\#C\#\}$ + *init-clss st* **and**
init-clss-add-learned-cls[*simp*]:
$\bigwedge$*C st. no-dup* (*trail st*) $\Longrightarrow$ *init-clss* (*add-learned-cls C st*) = *init-clss st* **and**
init-clss-remove-cls[*simp*]:
$\bigwedge$*C st. init-clss* (*remove-cls C st*) = *remove-mset C* (*init-clss st*) **and**
init-clss-update-backtrack-lvl[*simp*]:
$\bigwedge$*st C. init-clss* (*update-backtrack-lvl C st*) = *init-clss st* **and**
init-clss-update-conflicting[*simp*]:
$\bigwedge$*C st. init-clss* (*update-conflicting C st*) = *init-clss st* **and**

learned-clss-cons-trail[*simp*]:
$\bigwedge$*M st. undefined-lit* (*trail st*) (*lit-of M*) $\Longrightarrow$
learned-clss (*cons-trail M st*) = *learned-clss st* **and**
learned-clss-tl-trail[*simp*]:
$\bigwedge$*st. learned-clss* (*tl-trail st*) = *learned-clss st* **and**
learned-clss-add-init-cls[*simp*]:
$\bigwedge$*st C. no-dup* (*trail st*) $\Longrightarrow$ *learned-clss* (*add-init-cls C st*) = *learned-clss st* **and**
learned-clss-add-learned-cls[*simp*]:
$\bigwedge$*C st. no-dup* (*trail st*) $\Longrightarrow$ *learned-clss* (*add-learned-cls C st*) = $\{\#C\#\}$ + *learned-clss st*
and
learned-clss-remove-cls[*simp*]:
$\bigwedge$*C st. learned-clss* (*remove-cls C st*) = *remove-mset C* (*learned-clss st*) **and**
learned-clss-update-backtrack-lvl[*simp*]:
$\bigwedge$*st C. learned-clss* (*update-backtrack-lvl C st*) = *learned-clss st* **and**
learned-clss-update-conflicting[*simp*]:
$\bigwedge$*C st. learned-clss* (*update-conflicting C st*) = *learned-clss st* **and**

backtrack-lvl-cons-trail[*simp*]:
$\bigwedge$*M st. undefined-lit* (*trail st*) (*lit-of M*) $\Longrightarrow$
backtrack-lvl (*cons-trail M st*) = *backtrack-lvl st* **and**
backtrack-lvl-tl-trail[*simp*]:
$\bigwedge$*st. backtrack-lvl* (*tl-trail st*) = *backtrack-lvl st* **and**
backtrack-lvl-add-init-cls[*simp*]:
$\bigwedge$*st C. no-dup* (*trail st*) $\Longrightarrow$ *backtrack-lvl* (*add-init-cls C st*) = *backtrack-lvl st* **and**
backtrack-lvl-add-learned-cls[*simp*]:
$\bigwedge$*C st. no-dup* (*trail st*) $\Longrightarrow$ *backtrack-lvl* (*add-learned-cls C st*) = *backtrack-lvl st* **and**
backtrack-lvl-remove-cls[*simp*]:
$\bigwedge$*C st. backtrack-lvl* (*remove-cls C st*) = *backtrack-lvl st* **and**
backtrack-lvl-update-backtrack-lvl[*simp*]:
$\bigwedge$*st k. backtrack-lvl* (*update-backtrack-lvl k st*) = *k* **and**
backtrack-lvl-update-conflicting[*simp*]:
$\bigwedge$*C st. backtrack-lvl* (*update-conflicting C st*) = *backtrack-lvl st* **and**

conflicting-cons-trail[*simp*]:
$\bigwedge$*M st. undefined-lit* (*trail st*) (*lit-of M*) $\Longrightarrow$
conflicting (*cons-trail M st*) = *conflicting st* **and**
conflicting-tl-trail[*simp*]:
$\bigwedge$*st. conflicting* (*tl-trail st*) = *conflicting st* **and**
conflicting-add-init-cls[*simp*]:
$\bigwedge$*st C. no-dup* (*trail st*) $\Longrightarrow$ *conflicting* (*add-init-cls C st*) = *conflicting st* **and**

conflicting-add-learned-cls[*simp*]:
$\bigwedge C$ *st*. *no-dup* (*trail st*) $\Longrightarrow$ *conflicting* (*add-learned-cls C st*) = *conflicting st* **and**
conflicting-remove-cls[*simp*]:
$\bigwedge C$ *st*. *conflicting* (*remove-cls C st*) = *conflicting st* **and**
conflicting-update-backtrack-lvl[*simp*]:
$\bigwedge$*st C*. *conflicting* (*update-backtrack-lvl C st*) = *conflicting st* **and**
conflicting-update-conflicting[*simp*]:
$\bigwedge C$ *st*. *conflicting* (*update-conflicting C st*) = *C* **and**

init-state-trail[*simp*]: $\bigwedge N$. *trail* (*init-state N*) = [] **and**
init-state-clss[*simp*]: $\bigwedge N$. *init-clss* (*init-state N*) = *N* **and**
init-state-learned-clss[*simp*]: $\bigwedge N$. *learned-clss* (*init-state N*) = {#} **and**
init-state-backtrack-lvl[*simp*]: $\bigwedge N$. *backtrack-lvl* (*init-state N*) = *0* **and**
init-state-conflicting[*simp*]: $\bigwedge N$. *conflicting* (*init-state N*) = *None* **and**

trail-restart-state[*simp*]: *trail* (*restart-state S*) = [] **and**
init-clss-restart-state[*simp*]: *init-clss* (*restart-state S*) = *init-clss S* **and**
learned-clss-restart-state[*intro*]: *learned-clss* (*restart-state S*) ⊆# *learned-clss S* **and**
backtrack-lvl-restart-state[*simp*]: *backtrack-lvl* (*restart-state S*) = *0* **and**
conflicting-restart-state[*simp*]: *conflicting* (*restart-state S*) = *None*
begin

definition *clauses* :: *'st* ⇒ *'v clauses* **where**
clauses S = *init-clss S* + *learned-clss S*

lemma
shows
clauses-cons-trail[*simp*]:
undefined-lit (*trail S*) (*lit-of M*) $\Longrightarrow$*clauses* (*cons-trail M S*) = *clauses S* **and**

clss-tl-trail[*simp*]: *clauses* (*tl-trail S*) = *clauses S* **and**
clauses-add-learned-cls-unfolded:
no-dup (*trail S*) $\Longrightarrow$ *clauses* (*add-learned-cls U S*) = {#*U*#} + *learned-clss S* + *init-clss S*
and
clauses-add-init-cls[*simp*]:
no-dup (*trail S*) $\Longrightarrow$ *clauses* (*add-init-cls N S*) = {#*N*#} + *init-clss S* + *learned-clss S* **and**
clauses-update-backtrack-lvl[*simp*]: *clauses* (*update-backtrack-lvl k S*) = *clauses S* **and**
clauses-update-conflicting[*simp*]: *clauses* (*update-conflicting D S*) = *clauses S* **and**
clauses-remove-cls[*simp*]:
clauses (*remove-cls C S*) = *clauses S* − *replicate-mset* (*count* (*clauses S*) *C*) *C* **and**
clauses-add-learned-cls[*simp*]:
no-dup (*trail S*) $\Longrightarrow$ *clauses* (*add-learned-cls C S*) = {#*C*#} + *clauses S* **and**
clauses-restart[*simp*]: *clauses* (*restart-state S*) ⊆# *clauses S* **and**
clauses-init-state[*simp*]: $\bigwedge N$. *clauses* (*init-state N*) = *N*
prefer *9* **using** *clauses-def learned-clss-restart-state* **apply** *fastforce*
by (*auto simp*: *ac-simps replicate-mset-plus clauses-def intro*: *multiset-eqI*)

abbreviation *state* :: *'st* ⇒ (*'v*, *nat*, *'v clause*) *ann-literal list* × *'v clauses* × *'v clauses*
× *nat* × *'v clause option* **where**
state S ≡ (*trail S*, *init-clss S*, *learned-clss S*, *backtrack-lvl S*, *conflicting S*)

abbreviation *incr-lvl* :: *'st* ⇒ *'st* **where**
incr-lvl S ≡ *update-backtrack-lvl* (*backtrack-lvl S* + *1*) *S*

definition *state-eq* :: *'st* ⇒ *'st* ⇒ *bool* (**infix** ∼ *50*) **where**

$S \sim T \longleftrightarrow$ *state* S = *state* T

lemma *state-eq-ref*[*simp*, *intro*]:
  $S \sim S$
  unfolding *state-eq-def* **by** *auto*

lemma *state-eq-sym*:
  $S \sim T \longleftrightarrow T \sim S$
  unfolding *state-eq-def* **by** *auto*

lemma *state-eq-trans*:
  $S \sim T \Longrightarrow T \sim U \Longrightarrow S \sim U$
  unfolding *state-eq-def* **by** *auto*

lemma
  shows
    state-eq-trail: $S \sim T \Longrightarrow$ *trail* S = *trail* T **and**
    state-eq-init-clss: $S \sim T \Longrightarrow$ *init-clss* S = *init-clss* T **and**
    state-eq-learned-clss: $S \sim T \Longrightarrow$ *learned-clss* S = *learned-clss* T **and**
    state-eq-backtrack-lvl: $S \sim T \Longrightarrow$ *backtrack-lvl* S = *backtrack-lvl* T **and**
    state-eq-conflicting: $S \sim T \Longrightarrow$ *conflicting* S = *conflicting* T **and**
    state-eq-clauses: $S \sim T \Longrightarrow$ *clauses* S = *clauses* T **and**
    state-eq-undefined-lit: $S \sim T \Longrightarrow$ *undefined-lit* (*trail* S) L = *undefined-lit* (*trail* T) L
  unfolding *state-eq-def clauses-def* **by** *auto*

lemmas *state-simp*[*simp*] = *state-eq-trail state-eq-init-clss state-eq-learned-clss*
  state-eq-backtrack-lvl state-eq-conflicting state-eq-clauses state-eq-undefined-lit

lemma *atms-of-ms-learned-clss-restart-state-in-atms-of-ms-learned-clssI*[*intro*]:
  $x \in$ *atms-of-msu* (*learned-clss* (*restart-state* S)) $\Longrightarrow x \in$ *atms-of-msu* (*learned-clss* S)
  by (*meson atms-of-ms-mono learned-clss-restart-state set-mset-mono subsetCE*)

function *reduce-trail-to* :: *'a list* $\Rightarrow$ *'st* $\Rightarrow$ *'st* **where**
reduce-trail-to F S =
  (*if length* (*trail* S) = *length* F $\vee$ *trail* S = [] *then* S *else reduce-trail-to* F (*tl-trail* S))
by *fast+*
termination
  by (*relation measure* (λ(-, S). *length* (*trail* S))) *simp-all*

declare *reduce-trail-to.simps*[*simp del*]

lemma
  shows
  reduce-trail-to-nil[*simp*]: *trail* S = [] $\Longrightarrow$ *reduce-trail-to* F S = S **and**
  reduce-trail-to-eq-length[*simp*]: *length* (*trail* S) = *length* F $\Longrightarrow$ *reduce-trail-to* F S = S
  by (*auto simp*: *reduce-trail-to.simps*)

lemma *reduce-trail-to-length-ne*:
  length (*trail* S) $\neq$ *length* F $\Longrightarrow$ *trail* S $\neq$ [] $\Longrightarrow$
    reduce-trail-to F S = *reduce-trail-to* F (*tl-trail* S)
  by (*auto simp*: *reduce-trail-to.simps*)

lemma *trail-reduce-trail-to-length-le*:
  assumes *length* F > *length* (*trail* S)
  shows *trail* (*reduce-trail-to* F S) = []

using *assms* **apply** (*induction F S rule*: *reduce-trail-to.induct*)
by (*metis* (*no-types*, *hide-lams*) *length-tl less-imp-diff-less less-irrefl trail-tl-trail reduce-trail-to.simps*)

lemma *trail-reduce-trail-to-nil*[*simp*]:
trail (*reduce-trail-to* [] *S*) = []
apply (*induction* []:: (*'v*, *nat*, *'v clause*) *ann-literals S rule*: *reduce-trail-to.induct*)
by (*metis length-0-conv reduce-trail-to-length-ne reduce-trail-to-nil*)

lemma *clauses-reduce-trail-to-nil*:
clauses (*reduce-trail-to* [] *S*) = *clauses S*
proof (*induction* [] *S rule*: *reduce-trail-to.induct*)
case (*1 Sa*)
then have *clauses* (*reduce-trail-to* ([]::*'a list*) (*tl-trail Sa*)) = *clauses* (*tl-trail Sa*)
∨ *trail Sa* = []
by *fastforce*
then show *clauses* (*reduce-trail-to* ([]::*'a list*) *Sa*) = *clauses Sa*
by (*metis* (*no-types*) *length-0-conv reduce-trail-to-eq-length clss-tl-trail reduce-trail-to-length-ne*)
qed

lemma *reduce-trail-to-skip-beginning*:
assumes *trail S* = *F'* @ *F*
shows *trail* (*reduce-trail-to F S*) = *F*
using *assms* **by** (*induction F' arbitrary*: *S*) (*auto simp*: *reduce-trail-to-length-ne*)

lemma *clauses-reduce-trail-to*[*simp*]:
clauses (*reduce-trail-to F S*) = *clauses S*
apply (*induction F S rule*: *reduce-trail-to.induct*)
by (*metis clss-tl-trail reduce-trail-to.simps*)

lemma *conflicting-update-trial*[*simp*]:
conflicting (*reduce-trail-to F S*) = *conflicting S*
apply (*induction F S rule*: *reduce-trail-to.induct*)
by (*metis conflicting-tl-trail reduce-trail-to.simps*)

lemma *backtrack-lvl-update-trial*[*simp*]:
backtrack-lvl (*reduce-trail-to F S*) = *backtrack-lvl S*
apply (*induction F S rule*: *reduce-trail-to.induct*)
by (*metis backtrack-lvl-tl-trail reduce-trail-to.simps*)

lemma *init-clss-update-trial*[*simp*]:
init-clss (*reduce-trail-to F S*) = *init-clss S*
apply (*induction F S rule*: *reduce-trail-to.induct*)
by (*metis init-clss-tl-trail reduce-trail-to.simps*)

lemma *learned-clss-update-trial*[*simp*]:
learned-clss (*reduce-trail-to F S*) = *learned-clss S*
apply (*induction F S rule*: *reduce-trail-to.induct*)
by (*metis learned-clss-tl-trail reduce-trail-to.simps*)

lemma *trail-eq-reduce-trail-to-eq*:
trail S = *trail T* ⟹ *trail* (*reduce-trail-to F S*) = *trail* (*reduce-trail-to F T*)
apply (*induction F S arbitrary*: *T rule*: *reduce-trail-to.induct*)
by (*metis trail-tl-trail reduce-trail-to.simps*)

lemma *reduce-trail-to-state-eq*$_{NOT}$*-compatible*:
  assumes *ST*: $S \sim T$
  shows *reduce-trail-to F S* $\sim$ *reduce-trail-to F T*
proof −
  have *trail* (*reduce-trail-to F S*) = *trail* (*reduce-trail-to F T*)
    using *trail-eq-reduce-trail-to-eq*[*of S T F*] *ST* **by** *auto*
  then show *?thesis* **using** *ST* **by** (*auto simp del*: *state-simp simp*: *state-eq-def*)
qed

lemma *reduce-trail-to-trail-tl-trail-decomp*[*simp*]:
  trail S = *F'* @ *Marked K d* # *F* $\Longrightarrow$ (*trail* (*reduce-trail-to F S*)) = *F*
  apply (*rule reduce-trail-to-skip-beginning*[*of - F'* @ *Marked K d* # []])
  by (*cases F'*) (*auto simp add*:*tl-append reduce-trail-to-skip-beginning*)

lemma *reduce-trail-to-add-learned-cls*[*simp*]:
  no-dup (*trail S*) $\Longrightarrow$
    trail (*reduce-trail-to F* (*add-learned-cls C S*)) = *trail* (*reduce-trail-to F S*)
  by (*rule trail-eq-reduce-trail-to-eq*) *auto*

lemma *reduce-trail-to-add-init-cls*[*simp*]:
  no-dup (*trail S*) $\Longrightarrow$
    trail (*reduce-trail-to F* (*add-init-cls C S*)) = *trail* (*reduce-trail-to F S*)
  by (*rule trail-eq-reduce-trail-to-eq*) *auto*

lemma *reduce-trail-to-remove-learned-cls*[*simp*]:
  trail (*reduce-trail-to F* (*remove-cls C S*)) = *trail* (*reduce-trail-to F S*)
  by (*rule trail-eq-reduce-trail-to-eq*) *auto*

lemma *reduce-trail-to-update-conflicting*[*simp*]:
  trail (*reduce-trail-to F* (*update-conflicting C S*)) = *trail* (*reduce-trail-to F S*)
  by (*rule trail-eq-reduce-trail-to-eq*) *auto*

lemma *reduce-trail-to-update-backtrack-lvl*[*simp*]:
  trail (*reduce-trail-to F* (*update-backtrack-lvl C S*)) = *trail* (*reduce-trail-to F S*)
  by (*rule trail-eq-reduce-trail-to-eq*) *auto*

lemma *in-get-all-marked-decomposition-marked-or-empty*:
  assumes (*a*, *b*) $\in$ *set* (*get-all-marked-decomposition M*)
  shows *a* = [] $\vee$ (*is-marked* (*hd a*))
  using *assms*
proof (*induct M arbitrary*: *a b*)
  case *Nil* **then show** *?case* **by** *simp*
next
  case (*Cons m M*)
  show *?case*
    proof (*cases m*)
      case (*Marked l mark*)
      then show *?thesis* **using** *Cons* **by** *auto*
    next
      case (*Propagated l mark*)
      then show *?thesis* **using** *Cons* **by** (*cases get-all-marked-decomposition M*) *force+*
    qed
qed

lemma *in-get-all-marked-decomposition-trail-update-trail*[*simp*]:
  assumes *H*: (*L* # *M1*, *M2*) ∈ *set* (*get-all-marked-decomposition* (*trail S*))
  shows *trail* (*reduce-trail-to M1 S*) = *M1*
proof –
  obtain *K mark* **where**
    L: *L* = *Marked K mark*
    using *H* **by** (*cases L*) (*auto dest*!: *in-get-all-marked-decomposition-marked-or-empty*)
  obtain *c* **where**
    tr-S: *trail S* = *c* @ *M2* @ *L* # *M1*
    using *H* **by** *auto*
  show *?thesis*
    by (*rule reduce-trail-to-trail-tl-trail-decomp*[*of* - *c* @ *M2 K mark*])
     (*auto simp*: *tr-S L*)
qed

fun *append-trail* **where**
append-trail [] *S* = *S* |
append-trail (*L* # *M*) *S* = *append-trail M* (*cons-trail L S*)

lemma *trail-append-trail*:
  no-dup (*M* @ *trail S*) ⟹ *trail* (*append-trail M S*) = *rev M* @ *trail S*
  by (*induction M arbitrary*: *S*) (*auto simp*: *defined-lit-map*)

lemma *init-clss-append-trail*:
  no-dup (*M* @ *trail S*) ⟹ *init-clss* (*append-trail M S*) = *init-clss S*
  by (*induction M arbitrary*: *S*) (*auto simp*: *defined-lit-map*)

lemma *learned-clss-append-trail*:
  no-dup (*M* @ *trail S*) ⟹ *learned-clss* (*append-trail M S*) = *learned-clss S*
  by (*induction M arbitrary*: *S*) (*auto simp*: *defined-lit-map*)

lemma *conflicting-append-trail*:
  no-dup (*M* @ *trail S*) ⟹ *conflicting* (*append-trail M S*) = *conflicting S*
  by (*induction M arbitrary*: *S*) (*auto simp*: *defined-lit-map*)

lemma *backtrack-lvl-append-trail*:
  no-dup (*M* @ *trail S*) ⟹ *backtrack-lvl* (*append-trail M S*) = *backtrack-lvl S*
  by (*induction M arbitrary*: *S*) (*auto simp*: *defined-lit-map*)

lemma *clauses-append-trail*:
  no-dup (*M* @ *trail S*) ⟹ *clauses* (*append-trail M S*) = *clauses S*
  by (*induction M arbitrary*: *S*) (*auto simp*: *defined-lit-map*)

lemmas *state-access-simp* =
  trail-append-trail init-clss-append-trail learned-clss-append-trail backtrack-lvl-append-trail
  clauses-append-trail conflicting-append-trail

This function is useful for proofs to speak of a global trail change, but is a bad for programs and code in general.

fun *delete-trail-and-rebuild* **where**
delete-trail-and-rebuild M S = *append-trail* (*rev M*) (*reduce-trail-to* ([]:: *'v list*) *S*)

end

5.2 Special Instantiation: using Triples as State

5.3 CDCL Rules

Because of the strategy we will later use, we distinguish propagate, conflict from the other rules

locale
  cdcl$_W$ =
   state$_W$ *trail init-clss learned-clss backtrack-lvl conflicting cons-trail tl-trail add-init-cls add-learned-cls remove-cls update-backtrack-lvl update-conflicting init-state restart-state*
  for
   trail :: $'st \Rightarrow ('v, nat, 'v\ clause)$ *ann-literals* **and**
   init-clss :: $'st \Rightarrow 'v\ clauses$ **and**
   learned-clss :: $'st \Rightarrow 'v\ clauses$ **and**
   backtrack-lvl :: $'st \Rightarrow nat$ **and**
   conflicting :: $'st \Rightarrow 'v\ clause\ option$ **and**

   cons-trail :: $('v, nat, 'v\ clause)$ *ann-literal* $\Rightarrow 'st \Rightarrow 'st$ **and**
   tl-trail :: $'st \Rightarrow 'st$ **and**
   add-init-cls :: $'v\ clause \Rightarrow 'st \Rightarrow 'st$ **and**
   add-learned-cls :: $'v\ clause \Rightarrow 'st \Rightarrow 'st$ **and**
   remove-cls :: $'v\ clause \Rightarrow 'st \Rightarrow 'st$ **and**
   update-backtrack-lvl :: $nat \Rightarrow 'st \Rightarrow 'st$ **and**
   update-conflicting :: $'v\ clause\ option \Rightarrow 'st \Rightarrow 'st$ **and**

   init-state :: $'v\ clauses \Rightarrow 'st$ **and**
   restart-state :: $'st \Rightarrow 'st$
begin

inductive *propagate* :: $'st \Rightarrow 'st \Rightarrow bool$ **where**
propagate-rule[*intro*]:
  $state\ S = (M, N, U, k, None) \Longrightarrow C + \{\#L\#\} \in\#\ clauses\ S \Longrightarrow M \models as\ CNot\ C$
  $\Longrightarrow undefined\text{-}lit\ (trail\ S)\ L$
  $\Longrightarrow T \sim cons\text{-}trail\ (Propagated\ L\ (C + \{\#L\#\}))\ S$
  $\Longrightarrow propagate\ S\ T$
inductive-cases *propagateE*[*elim*]: *propagate S T*
thm *propagateE*

inductive *conflict* :: $'st \Rightarrow 'st \Rightarrow bool$ **where**
conflict-rule[*intro*]: $state\ S = (M, N, U, k, None) \Longrightarrow D \in\#\ clauses\ S \Longrightarrow M \models as\ CNot\ D$
  $\Longrightarrow T \sim update\text{-}conflicting\ (Some\ D)\ S$
  $\Longrightarrow conflict\ S\ T$

inductive-cases *conflictE*[*elim*]: *conflict S S'*

inductive *backtrack* :: $'st \Rightarrow 'st \Rightarrow bool$ **where**
backtrack-rule[*intro*]: $state\ S = (M, N, U, k, Some\ (D + \{\#L\#\}))$
  $\Longrightarrow (Marked\ K\ (i+1)\ \#\ M1, M2) \in set\ (get\text{-}all\text{-}marked\text{-}decomposition\ M)$
  $\Longrightarrow get\text{-}level\ M\ L = k$
  $\Longrightarrow get\text{-}level\ M\ L = get\text{-}maximum\text{-}level\ M\ (D+\{\#L\#\})$
  $\Longrightarrow get\text{-}maximum\text{-}level\ M\ D = i$
  $\Longrightarrow T \sim cons\text{-}trail\ (Propagated\ L\ (D+\{\#L\#\}))$
      (*reduce-trail-to M1*
       (*add-learned-cls* $(D + \{\#L\#\})$
        (*update-backtrack-lvl i*
         (*update-conflicting None S*))))

$\Longrightarrow$ *backtrack S T*
inductive-cases *backtrackE*[*elim*]: *backtrack S S'*
thm *backtrackE*

inductive *decide* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **where**
decide-rule[*intro*]: *state S* = (*M*, *N*, *U*, *k*, *None*)
$\Longrightarrow$ *undefined-lit M L* $\Longrightarrow$ *atm-of L* $\in$ *atms-of-msu* (*init-clss S*)
$\Longrightarrow$ *T* $\sim$ *cons-trail* (*Marked L* (*k*+*1*)) (*incr-lvl S*)
$\Longrightarrow$ *decide S T*
inductive-cases *decideE*[*elim*]: *decide S S'*
thm *decideE*

inductive *skip* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **where**
skip-rule[*intro*]: *state S* = (*Propagated L C'* # *M*, *N*, *U*, *k*, *Some D*) $\Longrightarrow$ $-L \notin\#$ *D* $\Longrightarrow$ *D* $\neq$ {#}
$\Longrightarrow$ *T* $\sim$ *tl-trail S*
$\Longrightarrow$ *skip S T*
inductive-cases *skipE*[*elim*]: *skip S S'*
thm *skipE*

get-maximum-level (*Propagated L* (*C* + {#*L*#}) # *M*) *D* = *k* $\vee$ *k* = *0* is equivalent to *get-maximum-level* (*Propagated L* (*C* + {#*L*#}) # *M*) *D* = *k*

inductive *resolve* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **where**
resolve-rule[*intro*]:
state S = (*Propagated L* (*C* + {#*L*#}) # *M*, *N*, *U*, *k*, *Some* (*D* + {#$-L$#}))
$\Longrightarrow$ *get-maximum-level* (*Propagated L* (*C* + {#*L*#}) # *M*) *D* = *k*
$\Longrightarrow$ *T* $\sim$ *update-conflicting* (*Some* (*D* #$\cup$ *C*)) (*tl-trail S*)
$\Longrightarrow$ *resolve S T*
inductive-cases *resolveE*[*elim*]: *resolve S S'*
thm *resolveE*

inductive *restart* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **where**
restart: *state S* = (*M*, *N*, *U*, *k*, *None*) $\Longrightarrow$ $\neg$*M* $\models$*asm clauses S*
$\Longrightarrow$ *T* $\sim$ *restart-state S*
$\Longrightarrow$ *restart S T*
inductive-cases *restartE*[*elim*]: *restart S T*
thm *restartE*

We add the condition *C* $\notin\#$ *init-clss S*, to maintain consistency even without the strategy.

inductive *forget* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **where**
forget-rule: *state S* = (*M*, *N*, {#*C*#} + *U*, *k*, *None*)
$\Longrightarrow$ $\neg$*M* $\models$*asm clauses S*
$\Longrightarrow$ *C* $\notin$ *set* (*get-all-mark-of-propagated* (*trail S*))
$\Longrightarrow$ *C* $\notin\#$ *init-clss S*
$\Longrightarrow$ *C* $\in\#$ *learned-clss S*
$\Longrightarrow$ *T* $\sim$ *remove-cls C S*
$\Longrightarrow$ *forget S T*
inductive-cases *forgetE*[*elim*]: *forget S T*

inductive $cdcl_W$*-rf* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **for** *S* :: *'st* **where**
restart: *restart S T* $\Longrightarrow$ $cdcl_W$*-rf S T* |
forget: *forget S T* $\Longrightarrow$ $cdcl_W$*-rf S T*

inductive $cdcl_W$*-bj* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **where**
skip[*intro*]: *skip S S'* $\Longrightarrow$ $cdcl_W$*-bj S S'* |
resolve[*intro*]: *resolve S S'* $\Longrightarrow$ $cdcl_W$*-bj S S'* |

backtrack[*intro*]: *backtrack* $S\ S' \Longrightarrow$ $cdcl_W$*-bj* $S\ S'$

inductive-cases $cdcl_W$*-bjE*: $cdcl_W$*-bj* $S\ T$

inductive $cdcl_W$*-o*:: *'st* ⇒ *'st* ⇒ *bool* **for** S :: *'st* **where**
decide[*intro*]: *decide* $S\ S' \Longrightarrow$ $cdcl_W$*-o* $S\ S'$ |
bj[*intro*]: $cdcl_W$*-bj* $S\ S' \Longrightarrow$ $cdcl_W$*-o* $S\ S'$

inductive $cdcl_W$ :: *'st* ⇒ *'st* ⇒ *bool* **for** S :: *'st* **where**
propagate: *propagate* $S\ S' \Longrightarrow$ $cdcl_W$ $S\ S'$ |
conflict: *conflict* $S\ S' \Longrightarrow$ $cdcl_W$ $S\ S'$ |
other: $cdcl_W$*-o* $S\ S' \Longrightarrow$ $cdcl_W$ $S\ S'$|
rf: $cdcl_W$*-rf* $S\ S' \Longrightarrow$ $cdcl_W$ $S\ S'$

lemma *rtranclp-propagate-is-rtranclp-*$cdcl_W$:
  propagate** $S\ S' \Longrightarrow$ $cdcl_W{}^{**}$ $S\ S'$
  by (*induction rule*: *rtranclp-induct*) (*fastforce dest*!: *propagate*)+

lemma $cdcl_W$*-all-rules-induct*[*consumes 1*, *case-names propagate conflict forget restart decide skip resolve backtrack*]:
  fixes S :: *'st*
  assumes
    $cdcl_W$: $cdcl_W$ $S\ S'$ **and**
    propagate: $\bigwedge T$. *propagate* $S\ T \Longrightarrow P\ S\ T$ **and**
    conflict: $\bigwedge T$. *conflict* $S\ T \Longrightarrow P\ S\ T$ **and**
    forget: $\bigwedge T$. *forget* $S\ T \Longrightarrow P\ S\ T$ **and**
    restart: $\bigwedge T$. *restart* $S\ T \Longrightarrow P\ S\ T$ **and**
    decide: $\bigwedge T$. *decide* $S\ T \Longrightarrow P\ S\ T$ **and**
    skip: $\bigwedge T$. *skip* $S\ T \Longrightarrow P\ S\ T$ **and**
    resolve: $\bigwedge T$. *resolve* $S\ T \Longrightarrow P\ S\ T$ **and**
    backtrack: $\bigwedge T$. *backtrack* $S\ T \Longrightarrow P\ S\ T$
  shows $P\ S\ S'$
  using *assms*(*1*)
proof (*induct* S' *rule*: $cdcl_W$*.induct*)
  case (*propagate* S') **note** *propagate* = *this*(*1*)
  then show *?case* **using** *assms*(*2*) **by** *auto*
next
  case (*conflict* S')
  then show *?case* **using** *assms*(*3*) **by** *auto*
next
  case (*other* S')
  then show *?case*
    proof (*induct rule*: $cdcl_W$*-o.induct*)
      case (*decide* U)
      then show *?case* **using** *assms*(*6*) **by** *auto*
    next
      case (*bj* S')
      then show *?case* **using** *assms*(*7*−*9*) **by** (*induction rule*: $cdcl_W$*-bj.induct*) *auto*
    qed
next
  case (*rf* S')
  then show *?case*
    by (*induct rule*: $cdcl_W$*-rf.induct*) (*fast dest*: *forget restart*)+
qed

lemma *cdcl*$_W$*-all-induct*[*consumes 1*, *case-names propagate conflict forget restart decide skip resolve backtrack*]:
fixes S :: $'st$
assumes
cdcl$_W$: *cdcl*$_W$ S S' **and**
propagateH: $\bigwedge C\ L\ T.\ C + \{\#L\#\} \in\#$ *clauses* $S \Longrightarrow$ *trail* $S \models as$ *CNot* C
$\Longrightarrow$ *undefined-lit* (*trail* S) $L \Longrightarrow$ *conflicting* S = *None*
$\Longrightarrow T \sim$ *cons-trail* (*Propagated* L ($C + \{\#L\#\}$)) S
$\Longrightarrow P\ S\ T$ **and**
conflictH: $\bigwedge D\ T.\ D \in\#$ *clauses* $S \Longrightarrow$ *conflicting* S = *None* $\Longrightarrow$ *trail* $S \models as$ *CNot* D
$\Longrightarrow T \sim$ *update-conflicting* (*Some* D) S
$\Longrightarrow P\ S\ T$ **and**
forgetH: $\bigwedge C\ T.\ \neg$*trail* $S \models asm$ *clauses* S
$\Longrightarrow C \notin$ *set* (*get-all-mark-of-propagated* (*trail* S))
$\Longrightarrow C \notin\#$ *init-clss* S
$\Longrightarrow C \in\#$ *learned-clss* S
$\Longrightarrow$ *conflicting* S = *None*
$\Longrightarrow T \sim$ *remove-cls* C S
$\Longrightarrow P\ S\ T$ **and**
restartH: $\bigwedge T.\ \neg$*trail* $S \models asm$ *clauses* S
$\Longrightarrow$ *conflicting* S = *None*
$\Longrightarrow T \sim$ *restart-state* S
$\Longrightarrow P\ S\ T$ **and**
decideH: $\bigwedge L\ T.$ *conflicting* S = *None* $\Longrightarrow$ *undefined-lit* (*trail* S) L
$\Longrightarrow$ *atm-of* $L \in$ *atms-of-msu* (*init-clss* S)
$\Longrightarrow T \sim$ *cons-trail* (*Marked* L (*backtrack-lvl* S +*1*)) (*incr-lvl* S)
$\Longrightarrow P\ S\ T$ **and**
skipH: $\bigwedge L\ C'\ M\ D\ T.$ *trail* S = *Propagated* L C' # M
$\Longrightarrow$ *conflicting* S = *Some* $D \Longrightarrow -L \notin\# D \Longrightarrow D \neq \{\#\}$
$\Longrightarrow T \sim$ *tl-trail* S
$\Longrightarrow P\ S\ T$ **and**
resolveH: $\bigwedge L\ C\ M\ D\ T.$
trail S = *Propagated* L (($C + \{\#L\#\}$)) # M
$\Longrightarrow$ *conflicting* S = *Some* ($D + \{\#-L\#\}$)
$\Longrightarrow$ *get-maximum-level* (*Propagated* L ($C + \{\#L\#\}$) # M) D = *backtrack-lvl* S
$\Longrightarrow T \sim$ (*update-conflicting* (*Some* (D #∪ C)) (*tl-trail* S))
$\Longrightarrow P\ S\ T$ **and**
backtrackH: $\bigwedge K\ i\ M1\ M2\ L\ D\ T.$
(*Marked* K (*Suc* i) # $M1$, $M2$) $\in$ *set* (*get-all-marked-decomposition* (*trail* S))
$\Longrightarrow$ *get-level* (*trail* S) L = *backtrack-lvl* S
$\Longrightarrow$ *conflicting* S = *Some* ($D + \{\#L\#\}$)
$\Longrightarrow$ *get-maximum-level* (*trail* S) (D+$\{\#L\#\}$) = *get-level* (*trail* S) L
$\Longrightarrow$ *get-maximum-level* (*trail* S) $D \equiv i$
$\Longrightarrow T \sim$ *cons-trail* (*Propagated* L (D+$\{\#L\#\}$))
(*reduce-trail-to* $M1$
(*add-learned-cls* ($D + \{\#L\#\}$)
(*update-backtrack-lvl* i
(*update-conflicting None* S))))
$\Longrightarrow P\ S\ T$
shows $P\ S\ S'$
using *cdcl*$_W$
proof (*induct* S S' *rule*: *cdcl*$_W$*-all-rules-induct*)
case (*propagate* S')
then show *?case* **by** (*elim propagateE*) (*frule propagateH*; *simp*)
next

case (*conflict* S')
then show *?case* **by** (*elim conflictE*) (*frule conflictH*; *simp*)
next
case (*restart* S')
then show *?case* **by** (*elim restartE*) (*frule restartH*; *simp*)
next
case (*decide* T)
then show *?case* **by** (*elim decideE*) (*frule decideH*; *simp*)
next
case (*backtrack* S')
then show *?case* **by** (*elim backtrackE*) (*frule backtrackH*; *simp del*: *state-simp add*: *state-eq-def*)
next
case (*forget* S')
then show *?case* **using** *forgetH* **by** *auto*
next
case (*skip* S')
then show *?case* **using** *skipH* **by** *auto*
next
case (*resolve* S')
then show *?case* **by** (*elim resolveE*) (*frule resolveH*; *simp*)
qed

lemma $cdcl_W$*-o-induct*[*consumes 1*, *case-names decide skip resolve backtrack*]:
fixes S :: *'st*
assumes $cdcl_W$: $cdcl_W$*-o* S T **and**
decideH: $\bigwedge L\ T.$ *conflicting* S = *None* $\Longrightarrow$ *undefined-lit* (*trail* S) L
$\Longrightarrow$ *atm-of* $L \in$ *atms-of-msu* (*init-clss* S)
$\Longrightarrow$ $T \sim$ *cons-trail* (*Marked* L (*backtrack-lvl* S +*1*)) (*incr-lvl* S)
$\Longrightarrow$ $P\ S\ T$ **and**
skipH: $\bigwedge L\ C'\ M\ D\ T.$ *trail* S = *Propagated* L C' # M
$\Longrightarrow$ *conflicting* S = *Some* D $\Longrightarrow$ $-L \notin\#$ D $\Longrightarrow$ $D \neq \{\#\}$
$\Longrightarrow$ $T \sim$ *tl-trail* S
$\Longrightarrow$ $P\ S\ T$ **and**
resolveH: $\bigwedge L\ C\ M\ D\ T.$
trail S = *Propagated* L ((C + {#L#})) # M
$\Longrightarrow$ *conflicting* S = *Some* (D + {#$-L$#})
$\Longrightarrow$ *get-maximum-level* (*Propagated* L (C + {#L#}) # M) D = *backtrack-lvl* S
$\Longrightarrow$ $T \sim$ *update-conflicting* (*Some* (D #$\cup$ C)) (*tl-trail* S)
$\Longrightarrow$ $P\ S\ T$ **and**
backtrackH: $\bigwedge K\ i\ M1\ M2\ L\ D\ T.$
(*Marked* K (*Suc* i) # *M1*, *M2*) $\in$ *set* (*get-all-marked-decomposition* (*trail* S))
$\Longrightarrow$ *get-level* (*trail* S) L = *backtrack-lvl* S
$\Longrightarrow$ *conflicting* S = *Some* (D + {#L#})
$\Longrightarrow$ *get-level* (*trail* S) L = *get-maximum-level* (*trail* S) (D+{#L#})
$\Longrightarrow$ *get-maximum-level* (*trail* S) $D \equiv i$
$\Longrightarrow$ $T \sim$ *cons-trail* (*Propagated* L (D+{#L#}))
(*reduce-trail-to M1*
(*add-learned-cls* (D + {#L#})
(*update-backtrack-lvl* i
(*update-conflicting None* S))))
$\Longrightarrow$ $P\ S\ T$
shows $P\ S\ T$
using $cdcl_W$ **apply** (*induct* T *rule*: $cdcl_W$*-o.induct*)
using *assms*(*2*) **apply** *auto*[*1*]
apply (*elim* $cdcl_W$*-bjE skipE resolveE backtrackE*)

apply (*frule skipH*; *simp*)
apply (*frule resolveH*; *simp*)
apply (*frule backtrackH*; *simp-all del*: *state-simp add*: *state-eq-def*)
done

thm $cdcl_W$*-o.induct*
lemma $cdcl_W$*-o-all-rules-induct*[*consumes 1*, *case-names decide backtrack skip resolve*]:
fixes *S T* :: *'st*
assumes
$cdcl_W$*-o S T* **and**
$\bigwedge T$. *decide S T* $\Longrightarrow$ *P S T* **and**
$\bigwedge T$. *backtrack S T* $\Longrightarrow$ *P S T* **and**
$\bigwedge T$. *skip S T* $\Longrightarrow$ *P S T* **and**
$\bigwedge T$. *resolve S T* $\Longrightarrow$ *P S T*
shows *P S T*
using *assms* **by** (*induct T rule*: $cdcl_W$*-o.induct*) (*auto simp*: $cdcl_W$*-bj.simps*)

lemma $cdcl_W$*-o-rule-cases*[*consumes 1*, *case-names decide backtrack skip resolve*]:
fixes *S T* :: *'st*
assumes
$cdcl_W$*-o S T* **and**
decide S T $\Longrightarrow$ *P* **and**
backtrack S T $\Longrightarrow$ *P* **and**
skip S T $\Longrightarrow$ *P* **and**
resolve S T $\Longrightarrow$ *P*
shows *P*
using *assms* **by** (*auto simp*: $cdcl_W$*-o.simps* $cdcl_W$*-bj.simps*)

5.4 Invariants

5.4.1 Properties of the trail

We here establish that: * the marks are exactly 1..k where k is the level * the consistency of the trail * the fact that there is no duplicate in the trail.

lemma *backtrack-lit-skiped*:
assumes *L*: *get-level* (*trail S*) *L* = *backtrack-lvl S*
and *M1*: (*Marked K* (*i* + *1*) # *M1*, *M2*) ∈ *set* (*get-all-marked-decomposition* (*trail S*))
and *no-dup*: *no-dup* (*trail S*)
and *bt-l*: *backtrack-lvl S* = *length* (*get-all-levels-of-marked* (*trail S*))
and *order*: *get-all-levels-of-marked* (*trail S*)
= *rev* ([*1*..<(*1*+*length* (*get-all-levels-of-marked* (*trail S*)))])
shows *atm-of L* ∉ *atm-of* ‘ *lits-of M1*
proof
let *?M* = *trail S*
assume *L-in-M1*: *atm-of L* ∈ *atm-of* ‘ *lits-of M1*
obtain *c* **where** *Mc*: *trail S* = *c* @ *M2* @ *Marked K* (*i* + *1*) # *M1* **using** *M1* **by** *blast*
have *atm-of L* ∉ *atm-of* ‘ *lits-of c*
using *L-in-M1 no-dup mk-disjoint-insert* **unfolding** *Mc lits-of-def* **by** *force*
have *g-M-eq-g-M1*: *get-level ?M L* = *get-level M1 L*
using *L-in-M1* **unfolding** *Mc* **by** *auto*
have *g*: *get-all-levels-of-marked M1* = *rev* [*1*..<*Suc i*]
using *order* **unfolding** *Mc*
by (*auto simp del*: *upt-simps dest*!: *append-cons-eq-upt-length-i*
simp add: *rev-swap*[*symmetric*])
then have *Max* (*set* (*0* # *get-all-levels-of-marked* (*rev M1*))) < *Suc i* **by** *auto*

then have *get-level M1 L* < *Suc i*
using *get-rev-level-less-max-get-all-levels-of-marked*[*of rev M1 0 L*] **by** *linarith*
moreover have *Suc i* ≤ *backtrack-lvl S* **using** *bt-l* **by** (*simp add*: *Mc g*)
ultimately show *False* **using** *L g-M-eq-g-M1* **by** *auto*
qed

lemma $cdcl_W$*-distinctinv-1*:
assumes
$cdcl_W$ *S S'* **and**
no-dup (*trail S*) **and**
backtrack-lvl S = *length* (*get-all-levels-of-marked* (*trail S*)) **and**
get-all-levels-of-marked (*trail S*) = *rev* [*1..<1+length* (*get-all-levels-of-marked* (*trail S*))]
shows *no-dup* (*trail S'*)
using *assms*
proof (*induct rule*: $cdcl_W$*-all-induct*)
case (*backtrack K i M1 M2 L D T*) **note** *decomp* = *this*(*1*) **and** *L* = *this*(*2*) **and** *T* = *this*(*6*) **and**
n-d = *this*(*7*)
obtain *c* **where** *Mc*: *trail S* = *c* @ *M2* @ *Marked K* (*i* + *1*) # *M1*
using *decomp* **by** *auto*
have *no-dup* (*M2* @ *Marked K* (*i* + *1*) # *M1*)
using *Mc n-d* **by** *fastforce*
moreover have *atm-of L* ∉ (λ*l. atm-of* (*lit-of l*)) ‘ *set M1*
using *backtrack-lit-skiped*[*of S L K i M1 M2*] *L decomp backtrack.prems*
by (*fastforce simp*: *lits-of-def*)
moreover then have *undefined-lit M1 L*
by (*simp add*: *defined-lit-map*)
ultimately show *?case* **using** *decomp T n-d* **by** *simp*
qed (*auto simp*: *defined-lit-map*)

lemma $cdcl_W$*-consistent-inv-2*:
assumes
$cdcl_W$ *S S'* **and**
no-dup (*trail S*) **and**
backtrack-lvl S = *length* (*get-all-levels-of-marked* (*trail S*)) **and**
get-all-levels-of-marked (*trail S*) = *rev* [*1..<1+length* (*get-all-levels-of-marked* (*trail S*))]
shows *consistent-interp* (*lits-of* (*trail S'*))
using $cdcl_W$*-distinctinv-1*[*OF assms*] *distinctconsistent-interp* **by** *fast*

lemma $cdcl_W$*-o-bt*:
assumes
$cdcl_W$*-o S S'* **and**
backtrack-lvl S = *length* (*get-all-levels-of-marked* (*trail S*)) **and**
get-all-levels-of-marked (*trail S*) =
rev ([*1..<*(*1+length* (*get-all-levels-of-marked* (*trail S*)))]) **and**
n-d[*simp*]: *no-dup* (*trail S*)
shows *backtrack-lvl S'* = *length* (*get-all-levels-of-marked* (*trail S'*))
using *assms*
proof (*induct rule*: $cdcl_W$*-o-induct*)
case (*backtrack K i M1 M2 L D T*) **note** *decomp* = *this*(*1*) **and** *T* = *this*(*6*) **and** *level* = *this*(*8*)
have [*simp*]: *trail* (*reduce-trail-to M1 S*) = *M1*
using *decomp* **by** *auto*
obtain *c* **where** *M*: *trail S* = *c* @ *M2* @ *Marked K* (*i* + *1*) # *M1* **using** *decomp* **by** *auto*
have *rev* (*get-all-levels-of-marked* (*trail S*))
= [*1..<1+* (*length* (*get-all-levels-of-marked* (*trail S*)))]
using *level* **by** (*auto simp*: *rev-swap*[*symmetric*])

moreover have *atm-of L* $\notin$ ($\lambda l.$ *atm-of* (*lit-of l*)) ' *set M1*
  using *backtrack-lit-skiped*[*of S L K i M1 M2*] *backtrack*(*2*,*7*,*8*,*9*) *decomp*
  by (*fastforce simp add*: *lits-of-def*)
moreover then have *undefined-lit M1 L*
  by (*simp add*: *defined-lit-map*)
moreover then have *no-dup* (*trail T*)
  using *T decomp n-d* **by** (*auto simp*: *defined-lit-map M*)
ultimately show *?case*
  using *T n-d* **unfolding** *M* **by** (*auto dest*!: *append-cons-eq-upt-length simp del*: *upt-simps*)
qed *auto*

lemma $cdcl_W$*-rf-bt*:
assumes
  $cdcl_W$*-rf S S'* **and**
  backtrack-lvl S = *length* (*get-all-levels-of-marked* (*trail S*)) **and**
  get-all-levels-of-marked (*trail S*) = *rev* [*1*..<(*1*+*length* (*get-all-levels-of-marked* (*trail S*)))]
shows *backtrack-lvl S'* = *length* (*get-all-levels-of-marked* (*trail S'*))
using *assms* **by** (*induct rule*: $cdcl_W$*-rf.induct*) *auto*

lemma $cdcl_W$*-bt*:
assumes
  $cdcl_W$ *S S'* **and**
  backtrack-lvl S = *length* (*get-all-levels-of-marked* (*trail S*)) **and**
  get-all-levels-of-marked (*trail S*)
  = *rev* ([*1*..<(*1*+*length* (*get-all-levels-of-marked* (*trail S*)))]) **and**
  no-dup (*trail S*)
shows *backtrack-lvl S'* = *length* (*get-all-levels-of-marked* (*trail S'*))
using *assms* **by** (*induct rule*: $cdcl_W$*.induct*) (*auto simp add*: $cdcl_W$*-o-bt* $cdcl_W$*-rf-bt*)

lemma $cdcl_W$*-bt-level'*:
assumes
  $cdcl_W$ *S S'* **and**
  backtrack-lvl S = *length* (*get-all-levels-of-marked* (*trail S*)) **and**
  get-all-levels-of-marked (*trail S*)
    = *rev* ([*1*..<(*1*+*length* (*get-all-levels-of-marked* (*trail S*)))]) **and**
  n-d: *no-dup* (*trail S*)
shows *get-all-levels-of-marked* (*trail S'*)
  = *rev* ([*1*..<(*1*+*length* (*get-all-levels-of-marked* (*trail S'*)))])
using *assms*
proof (*induct rule*: $cdcl_W$*-all-induct*)
case (*decide L T*) **note** *undef* = *this*(*2*) **and** *T* = *this*(*4*)
let *?k* = *backtrack-lvl S*
let *?M* = *trail S*
let *?M'* = *Marked L* (*?k* + *1*) # *trail S*
have *H*: *get-all-levels-of-marked ?M* = *rev* [*Suc 0*..<*1*+*length* (*get-all-levels-of-marked ?M*)]
  using *decide.prems* **by** *simp*
have *k*: *?k* = *length* (*get-all-levels-of-marked ?M*)
  using *decide.prems* **by** *auto*
have *get-all-levels-of-marked ?M'* = *Suc ?k* # *get-all-levels-of-marked ?M* **by** *simp*
then have *get-all-levels-of-marked ?M'* = *Suc ?k* #
    rev [*Suc 0*..<*1*+*length* (*get-all-levels-of-marked ?M*)]
  using *H* **by** *auto*
moreover have ... = *rev* [*Suc 0*..< *Suc* (*1*+*length* (*get-all-levels-of-marked ?M*))]
  unfolding *k* **by** *simp*
finally show *?case* **using** *T undef* **by** (*auto simp add*: *defined-lit-map*)

next
case (*backtrack K i M1 M2 L D T*) **note** *decomp* = *this*(*1*) **and** *confli* = *this*(*2*) **and** *T* =*this*(*6*) **and**
all-marked = *this*(*8*) **and** *bt-lvl* = *this*(*7*)
have *atm-of L* ∉ (λ*l*. *atm-of* (*lit-of l*)) ‘ *set M1*
using *backtrack-lit-skiped*[*of S L K i M1 M2*] *backtrack*(*2*,*7*,*8*,*9*) *decomp*
by (*fastforce simp add*: *lits-of-def*)
moreover then have *undefined-lit M1 L*
by (*simp add*: *defined-lit-map*)
then have [*simp*]: *trail T* = *Propagated L* (*D* + {#*L*#}) # *M1*
using *T decomp n-d* **by** *auto*
obtain *c* **where** *M*: *trail S* = *c* @ *M2* @ *Marked K* (*i* + *1*) # *M1* **using** *decomp* **by** *auto*
have *get-all-levels-of-marked* (*rev* (*trail S*))
= [*Suc 0*..<*2*+*length* (*get-all-levels-of-marked c*) + (*length* (*get-all-levels-of-marked M2*)
+ *length* (*get-all-levels-of-marked M1*))]
using *all-marked bt-lvl* **unfolding** *M* **by** (*auto simp add*: *rev-swap*[*symmetric*] *simp del*: *upt-simps*)
then show *?case*
using *T* **by** (*auto simp add*: *rev-swap M dest*!: *append-cons-eq-upt*(*1*) *simp del*: *upt-simps*)
qed *auto*

We write *1* + *length* (*get-all-levels-of-marked* (*trail S*)) instead of *backtrack-lvl S* to avoid non termination of rewriting.

definition $cdcl_W$*-M-level-inv* (*S*:: *'st*) ⟷
consistent-interp (*lits-of* (*trail S*))
∧ *no-dup* (*trail S*)
∧ *backtrack-lvl S* = *length* (*get-all-levels-of-marked* (*trail S*))
∧ *get-all-levels-of-marked* (*trail S*)
= *rev* ([*1*..<*1*+*length* (*get-all-levels-of-marked* (*trail S*))])

lemma $cdcl_W$*-M-level-inv-decomp*:
assumes $cdcl_W$*-M-level-inv S*
shows *consistent-interp* (*lits-of* (*trail S*))
and *no-dup* (*trail S*)
using *assms* **unfolding** $cdcl_W$*-M-level-inv-def* **by** *fastforce*+

lemma $cdcl_W$*-consistent-inv*:
fixes *S S'* :: *'st*
assumes
$cdcl_W$ *S S'* **and**
$cdcl_W$*-M-level-inv S*
shows $cdcl_W$*-M-level-inv S'*
using *assms* $cdcl_W$*-consistent-inv-2* $cdcl_W$*-distinctinv-1* $cdcl_W$*-bt* $cdcl_W$*-bt-level'*
unfolding $cdcl_W$*-M-level-inv-def* **by** *meson*+

lemma *rtranclp-*$cdcl_W$*-consistent-inv*:
assumes $cdcl_W^{**}$ *S S'*
and $cdcl_W$*-M-level-inv S*
shows $cdcl_W$*-M-level-inv S'*
using *assms* **by** (*induct rule*: *rtranclp-induct*)
(*auto intro*: $cdcl_W$*-consistent-inv*)

lemma *tranclp-*$cdcl_W$*-consistent-inv*:
assumes $cdcl_W^{++}$ *S S'*
and $cdcl_W$*-M-level-inv S*
shows $cdcl_W$*-M-level-inv S'*

using *assms* **by** (*induct rule*: *tranclp-induct*)
(*auto intro*: $cdcl_W$*-consistent-inv*)

lemma $cdcl_W$*-M-level-inv-S0-*$cdcl_W$[*simp*]:
$cdcl_W$*-M-level-inv* (*init-state N*)
unfolding $cdcl_W$*-M-level-inv-def* **by** *auto*

lemma $cdcl_W$*-M-level-inv-get-level-le-backtrack-lvl*:
assumes *inv*: $cdcl_W$*-M-level-inv S*
shows *get-level* (*trail S*) $L \le$ *backtrack-lvl S*
proof −
have *get-all-levels-of-marked* (*trail S*) = *rev* [*1*..<*1* + *backtrack-lvl S*]
using *inv* **unfolding** $cdcl_W$*-M-level-inv-def* **by** *auto*
then show *?thesis*
using *get-rev-level-less-max-get-all-levels-of-marked*[*of rev* (*trail S*) *0 L*]
by (*auto simp*: *Max-n-upt*)
qed

lemma *backtrack-ex-decomp*:
assumes *M-l*: $cdcl_W$*-M-level-inv S*
and *i-S*: *i* < *backtrack-lvl S*
shows ∃ *K M1 M2*. (*Marked K* (*i* + *1*) # *M1*, *M2*) ∈ *set* (*get-all-marked-decomposition* (*trail S*))
proof −
let *?M* = *trail S*
have
g: *get-all-levels-of-marked* (*trail S*) = *rev* [*Suc 0*..<*Suc* (*backtrack-lvl S*)]
using *M-l* **unfolding** $cdcl_W$*-M-level-inv-def* **by** *simp-all*
then have *i*+*1* ∈ *set* (*get-all-levels-of-marked* (*trail S*))
using *i-S* **by** *auto*

then obtain *c K c'* **where** *tr-S*: *trail S* = *c* @ *Marked K* (*i* + *1*) # *c'*
using *in-get-all-levels-of-marked-iff-decomp*[*of i*+*1 trail S*] **by** *auto*

obtain *M1 M2* **where** (*Marked K* (*i* + *1*) # *M1*, *M2*) ∈ *set* (*get-all-marked-decomposition* (*trail S*))
unfolding *tr-S* **apply** (*induct c rule*: *ann-literal-list-induct*)
apply *auto*[*2*]
apply (*rename-tac L m xs*,
case-tac hd (*get-all-marked-decomposition* (*xs* @ *Marked K* (*Suc i*) # *c'*)))
apply (*case-tac get-all-marked-decomposition* (*xs* @ *Marked K* (*Suc i*) # *c'*))
by *auto*
then show *?thesis* **by** *blast*
qed

5.4.2 Better-Suited Induction Principle

We generalise the induction principle defined previously: the induction case for *backtrack* now includes the assumption that *undefined-lit M1 L*. This helps the simplifier and thus the automation.

lemma *backtrack-induction-lev*[*consumes 1*, *case-names M-devel-inv backtrack*]:
assumes
bt: *backtrack S T* **and**
inv: $cdcl_W$*-M-level-inv S* **and**
backtrackH: ⋀*K i M1 M2 L D T*.
(*Marked K* (*Suc i*) # *M1*, *M2*) ∈ *set* (*get-all-marked-decomposition* (*trail S*))
⟹ *get-level* (*trail S*) *L* = *backtrack-lvl S*

$\Longrightarrow$ *conflicting S = Some (D + {#L#})*
$\Longrightarrow$ *get-level (trail S) L = get-maximum-level (trail S) (D+{#L#})*
$\Longrightarrow$ *get-maximum-level (trail S) D* $\equiv$ *i*
$\Longrightarrow$ *undefined-lit M1 L*
$\Longrightarrow$ *T* $\sim$ *cons-trail (Propagated L (D+{#L#}))*
(reduce-trail-to M1
(add-learned-cls (D + {#L#})
(update-backtrack-lvl i
(update-conflicting None S))))
$\Longrightarrow$ *P S T*
shows *P S T*
proof −
obtain *K i M1 M2 L D* **where**
decomp: *(Marked K (Suc i) # M1, M2)* $\in$ *set (get-all-marked-decomposition (trail S))* **and**
L: *get-level (trail S) L = backtrack-lvl S* **and**
confl: *conflicting S = Some (D + {#L#})* **and**
lev-L: *get-level (trail S) L = get-maximum-level (trail S) (D+{#L#})* **and**
lev-D: *get-maximum-level (trail S) D* $\equiv$ *i* **and**
T: *T* $\sim$ *cons-trail (Propagated L (D+{#L#}))*
(reduce-trail-to M1
(add-learned-cls (D + {#L#})
(update-backtrack-lvl i
(update-conflicting None S))))
using *bt* **by** *(elim backtrackE) metis*

have *atm-of L* $\notin$ *(λl. atm-of (lit-of l)) ' set M1*
using *backtrack-lit-skiped[of S L K i M1 M2] L decomp bt confl lev-L lev-D inv*
unfolding *$cdcl_W$-M-level-inv-def*
by *(fastforce simp add: lits-of-def)*
then have *undefined-lit M1 L*
by *(auto simp: defined-lit-map)*
then show *?thesis*
using *backtrackH[OF decomp L confl lev-L lev-D - T]* **by** *simp*
qed

lemmas *backtrack-induction-lev2 = backtrack-induction-lev[consumes 2, case-names backtrack]*

lemma *$cdcl_W$-all-induct-lev-full*:
fixes *S :: 'st*
assumes
$cdcl_W$: *$cdcl_W$ S S'* **and**
inv[simp]: *$cdcl_W$-M-level-inv S* **and**
propagateH: $\bigwedge$*C L T. C + {#L#}* $\in$# *clauses S* $\Longrightarrow$ *trail S* $\models$*as CNot C*
$\Longrightarrow$ *undefined-lit (trail S) L* $\Longrightarrow$ *conflicting S = None*
$\Longrightarrow$ *T* $\sim$ *cons-trail (Propagated L (C + {#L#})) S*
$\Longrightarrow$ *$cdcl_W$-M-level-inv S*
$\Longrightarrow$ *P S T* **and**
conflictH: $\bigwedge$*D T. D* $\in$# *clauses S* $\Longrightarrow$ *conflicting S = None* $\Longrightarrow$ *trail S* $\models$*as CNot D*
$\Longrightarrow$ *T* $\sim$ *update-conflicting (Some D) S*
$\Longrightarrow$ *P S T* **and**
forgetH: $\bigwedge$*C T.* $\neg$*trail S* $\models$*asm clauses S*
$\Longrightarrow$ *C* $\notin$ *set (get-all-mark-of-propagated (trail S))*
$\Longrightarrow$ *C* $\notin$# *init-clss S*
$\Longrightarrow$ *C* $\in$# *learned-clss S*
$\Longrightarrow$ *conflicting S = None*

$\Longrightarrow T \sim$ *remove-cls C S*
$\Longrightarrow$ *cdcl*$_W$*-M-level-inv S*
$\Longrightarrow P\ S\ T$ **and**
restartH: $\bigwedge T$. $\neg$*trail S* $\models$*asm clauses S*
$\Longrightarrow$ *conflicting S = None*
$\Longrightarrow T \sim$ *restart-state S*
$\Longrightarrow$ *cdcl*$_W$*-M-level-inv S*
$\Longrightarrow P\ S\ T$ **and**
decideH: $\bigwedge L\ T$. *conflicting S = None* $\Longrightarrow$ *undefined-lit (trail S) L*
$\Longrightarrow$ *atm-of L* $\in$ *atms-of-msu (init-clss S)*
$\Longrightarrow T \sim$ *cons-trail (Marked L (backtrack-lvl S +1)) (incr-lvl S)*
$\Longrightarrow$ *cdcl*$_W$*-M-level-inv S*
$\Longrightarrow P\ S\ T$ **and**
skipH: $\bigwedge L\ C'\ M\ D\ T$. *trail S = Propagated L C' # M*
$\Longrightarrow$ *conflicting S = Some D* $\Longrightarrow -L \notin\#\ D \Longrightarrow D \neq \{\#\}$
$\Longrightarrow T \sim$ *tl-trail S*
$\Longrightarrow$ *cdcl*$_W$*-M-level-inv S*
$\Longrightarrow P\ S\ T$ **and**
resolveH: $\bigwedge L\ C\ M\ D\ T$.
trail S = Propagated L ((C + {#L#})) # M
$\Longrightarrow$ *conflicting S = Some (D + {#−L#})*
$\Longrightarrow$ *get-maximum-level (Propagated L (C + {#L#}) # M) D = backtrack-lvl S*
$\Longrightarrow T \sim$ *(update-conflicting (Some (D #∪ C)) (tl-trail S))*
$\Longrightarrow$ *cdcl*$_W$*-M-level-inv S*
$\Longrightarrow P\ S\ T$ **and**
backtrackH: $\bigwedge K\ i\ M1\ M2\ L\ D\ T$.
(Marked K (Suc i) # M1, M2) $\in$ *set (get-all-marked-decomposition (trail S))*
$\Longrightarrow$ *get-level (trail S) L = backtrack-lvl S*
$\Longrightarrow$ *conflicting S = Some (D + {#L#})*
$\Longrightarrow$ *get-maximum-level (trail S) (D+{#L#}) = get-level (trail S) L*
$\Longrightarrow$ *get-maximum-level (trail S) D* $\equiv$ *i*
$\Longrightarrow$ *undefined-lit M1 L*
$\Longrightarrow T \sim$ *cons-trail (Propagated L (D+{#L#}))*
(reduce-trail-to M1
(add-learned-cls (D + {#L#})
(update-backtrack-lvl i
(update-conflicting None S))))
$\Longrightarrow$ *cdcl*$_W$*-M-level-inv S*
$\Longrightarrow P\ S\ T$
shows $P\ S\ S'$
using *cdcl*$_W$
proof *(induct S' rule: cdcl*$_W$*-all-rules-induct)*
case *(propagate S')*
then show *?case* **by** *(elim propagateE) (frule propagateH; simp)*
next
case *(conflict S')*
then show *?case* **by** *(elim conflictE) (frule conflictH; simp)*
next
case *(restart S')*
then show *?case* **by** *(elim restartE) (frule restartH; simp)*
next
case *(decide T)*
then show *?case* **by** *(elim decideE) (frule decideH; simp)*
next
case *(backtrack S')*

then show *?case*
  apply (*induction rule*: *backtrack-induction-lev*)
   apply (*rule inv*)
  by (*rule backtrackH*;
    fastforce simp del: *state-simp simp add*: *state-eq-def dest*!: *HOL.meta-eq-to-obj-eq*)
next
  case (*forget* S')
  then show *?case* **using** *forgetH* **by** *auto*
next
  case (*skip* S')
  then show *?case* **using** *skipH* **by** *auto*
next
  case (*resolve* S')
  then show *?case* **by** (*elim resolveE*) (*frule resolveH*; *simp*)
qed

lemmas $cdcl_W$*-all-induct-lev2* = $cdcl_W$*-all-induct-lev-full*[*consumes 2*, *case-names propagate conflict forget restart decide skip resolve backtrack*]

lemmas $cdcl_W$*-all-induct-lev* = $cdcl_W$*-all-induct-lev-full*[*consumes 1*, *case-names lev-inv propagate conflict forget restart decide skip resolve backtrack*]

thm $cdcl_W$*-o-induct*
lemma $cdcl_W$*-o-induct-lev*[*consumes 1*, *case-names M-lev decide skip resolve backtrack*]:
  fixes S :: *'st*
  assumes
    $cdcl_W$: $cdcl_W$*-o* S T **and**
    inv[*simp*]: $cdcl_W$*-M-level-inv* S **and**
    decideH: $\bigwedge L\ T.$ *conflicting* S = *None* $\Longrightarrow$ *undefined-lit* (*trail* S) L
      $\Longrightarrow$ *atm-of* $L \in$ *atms-of-msu* (*init-clss* S)
      $\Longrightarrow$ $T \sim$ *cons-trail* (*Marked* L (*backtrack-lvl* S +*1*)) (*incr-lvl* S)
      $\Longrightarrow$ $cdcl_W$*-M-level-inv* S
      $\Longrightarrow$ $P\ S\ T$ **and**
    skipH: $\bigwedge L\ C'\ M\ D\ T.$ *trail* S = *Propagated* L C' # M
      $\Longrightarrow$ *conflicting* S = *Some* D $\Longrightarrow$ $-L \notin\#$ D $\Longrightarrow$ $D \neq \{\#\}$
      $\Longrightarrow$ $T \sim$ *tl-trail* S
      $\Longrightarrow$ $cdcl_W$*-M-level-inv* S
      $\Longrightarrow$ $P\ S\ T$ **and**
    resolveH: $\bigwedge L\ C\ M\ D\ T.$
      trail S = *Propagated* L ((C + {#L#})) # M
      $\Longrightarrow$ *conflicting* S = *Some* (D + {#$-L$#})
      $\Longrightarrow$ *get-maximum-level* (*Propagated* L (C + {#L#}) # M) D = *backtrack-lvl* S
      $\Longrightarrow$ $T \sim$ *update-conflicting* (*Some* (D #∪ C)) (*tl-trail* S)
      $\Longrightarrow$ $cdcl_W$*-M-level-inv* S
      $\Longrightarrow$ $P\ S\ T$ **and**
    backtrackH: $\bigwedge K\ i\ M1\ M2\ L\ D\ T.$
      (*Marked* K (*Suc* i) # *M1*, *M2*) $\in$ *set* (*get-all-marked-decomposition* (*trail* S))
      $\Longrightarrow$ *get-level* (*trail* S) L = *backtrack-lvl* S
      $\Longrightarrow$ *conflicting* S = *Some* (D + {#L#})
      $\Longrightarrow$ *get-level* (*trail* S) L = *get-maximum-level* (*trail* S) (D+{#L#})
      $\Longrightarrow$ *get-maximum-level* (*trail* S) $D \equiv i$
      $\Longrightarrow$ *undefined-lit M1* L
      $\Longrightarrow$ $T \sim$ *cons-trail* (*Propagated* L (D+{#L#}))
          (*reduce-trail-to M1*
            (*add-learned-cls* (D + {#L#})

(*update-backtrack-lvl i*
(*update-conflicting None S*))))
$\Longrightarrow$ $cdcl_W$*-M-level-inv S*
$\Longrightarrow$ *P S T*
shows *P S T*
using $cdcl_W$
proof (*induct S T rule*: $cdcl_W$*-o-all-rules-induct*)
case (*decide T*)
then show *?case* **by** (*elim decideE*) (*frule decideH*; *simp*)
next
case (*backtrack S'*)
then show *?case*
using *inv* **apply** (*induction rule*: *backtrack-induction-lev2*)
by (*rule backtrackH*)
(*fastforce simp del*: *state-simp simp add*: *state-eq-def dest*!: *HOL.meta-eq-to-obj-eq*)+
next
case (*skip S'*)
then show *?case* **using** *skipH* **by** *auto*
next
case (*resolve S'*)
then show *?case* **by** (*elim resolveE*) (*frule resolveH*; *simp*)
qed

lemmas $cdcl_W$*-o-induct-lev2* = $cdcl_W$*-o-induct-lev*[*consumes 2*, *case-names decide skip resolve backtrack*]

5.4.3 Compatibility with *op* ∼

lemma *propagate-state-eq-compatible*:
assumes
propagate S T **and**
S ∼ *S'* **and**
T ∼ *T'*
shows *propagate S' T'*
using *assms* **apply** (*elim propagateE*)
apply (*rule propagate-rule*)
by (*auto simp*: *state-eq-def clauses-def simp del*: *state-simp*)

lemma *conflict-state-eq-compatible*:
assumes
conflict S T **and**
S ∼ *S'* **and**
T ∼ *T'*
shows *conflict S' T'*
using *assms* **apply** (*elim conflictE*)
apply (*rule conflict-rule*)
by (*auto simp*: *state-eq-def clauses-def simp del*: *state-simp*)

lemma *backtrack-state-eq-compatible*:
assumes
backtrack S T **and**
S ∼ *S'* **and**
T ∼ *T'* **and**
inv: $cdcl_W$*-M-level-inv S*
shows *backtrack S' T'*
using *assms* **apply** (*induction rule*: *backtrack-induction-lev*)

using *inv* **apply** *simp*
apply (*rule backtrack-rule*)
apply *auto*[*5*]
by (*auto simp*: *state-eq-def clauses-def* $cdcl_W$*-M-level-inv-def simp del*: *state-simp*)

lemma *decide-state-eq-compatible*:
assumes
decide S T **and**
$S \sim S'$ **and**
$T \sim T'$
shows *decide* S' T'
using *assms* **apply** (*elim decideE*)
apply (*rule decide-rule*)
by (*auto simp*: *state-eq-def clauses-def simp del*: *state-simp*)

lemma *skip-state-eq-compatible*:
assumes
skip S T **and**
$S \sim S'$ **and**
$T \sim T'$
shows *skip* S' T'
using *assms* **apply** (*elim skipE*)
apply (*rule skip-rule*)
by (*auto simp*: *state-eq-def clauses-def HOL.eq-sym-conv*[*of* - # - *trail* -]
simp del: *state-simp dest*: *arg-cong*[*of* - # *trail* - *trail* - *tl*])

lemma *resolve-state-eq-compatible*:
assumes
resolve S T **and**
$S \sim S'$ **and**
$T \sim T'$
shows *resolve* S' T'
using *assms* **apply** (*elim resolveE*)
apply (*rule resolve-rule*)
by (*auto simp*: *state-eq-def clauses-def HOL.eq-sym-conv*[*of* - # - *trail* -]
simp del: *state-simp dest*: *arg-cong*[*of* - # *trail* - *trail* - *tl*])

lemma *forget-state-eq-compatible*:
assumes
forget S T **and**
$S \sim S'$ **and**
$T \sim T'$
shows *forget* S' T'
using *assms* **apply** (*elim forgetE*)
apply (*rule forget-rule*)
by (*auto simp*: *state-eq-def clauses-def HOL.eq-sym-conv*[*of* {#-#} + - -]
simp del: *state-simp dest*: *arg-cong*[*of* - # *trail* - *trail* - *tl*])

lemma $cdcl_W$*-state-eq-compatible*:
assumes
$cdcl_W$ *S T* **and** ¬*restart S T* **and**
$S \sim S'$ **and**
$T \sim T'$ **and**
inv: $cdcl_W$*-M-level-inv S*
shows $cdcl_W$ S' T'

using *assms* **by** (*meson assms backtrack-state-eq-compatible bj* $cdcl_W$*.simps* $cdcl_W$*-bj.simps* $cdcl_W$*-o-rule-cases* $cdcl_W$*-rf.cases* $cdcl_W$*-rf.restart conflict-state-eq-compatible decide decide-state-eq-compatible forget forget-state-eq-compatible propagate-state-eq-compatible resolve-state-eq-compatible skip-state-eq-compatible*)

lemma $cdcl_W$*-bj-state-eq-compatible*:
assumes
$cdcl_W$*-bj* S T **and** $cdcl_W$*-M-level-inv* S
$S \sim S'$ **and**
$T \sim T'$
shows $cdcl_W$*-bj* S' T'
using *assms*
by *induction* (*auto*
intro: *skip-state-eq-compatible backtrack-state-eq-compatible resolve-state-eq-compatible*)

lemma *tranclp-*$cdcl_W$*-bj-state-eq-compatible*:
assumes
$cdcl_W$*-bj*$^{++}$ S T **and** *inv*: $cdcl_W$*-M-level-inv* S **and**
$S \sim S'$ **and**
$T \sim T'$
shows $cdcl_W$*-bj*$^{++}$ S' T'
using *assms*
proof (*induction arbitrary*: S' T')
case *base*
then show *?case*
using $cdcl_W$*-bj-state-eq-compatible* **by** *blast*
next
case (*step* T U) **note** $IH = this(3)[OF\ this(4-5)]$
have $cdcl_W{}^{++}$ S T
using *tranclp-mono*[*of* $cdcl_W$*-bj* $cdcl_W$] *other step.hyps*(*1*) **by** *blast*
then have $cdcl_W$*-M-level-inv* T
using *inv tranclp-*$cdcl_W$*-consistent-inv* **by** *blast*
then have $cdcl_W$*-bj*$^{++}$ T T'
using ‹$U \sim T'$› $cdcl_W$*-bj-state-eq-compatible*[*of* T U] ‹$cdcl_W$*-bj* T U› **by** *auto*
then show *?case*
using *IH*[*of* T] **by** *auto*
qed

5.4.4 Conservation of some Properties

lemma *level-of-marked-ge-1*:
assumes
$cdcl_W$ S S' **and**
inv: $cdcl_W$*-M-level-inv* S **and**
$\forall L\ l.\ Marked\ L\ l \in set\ (trail\ S) \longrightarrow l > 0$
shows $\forall L\ l.\ Marked\ L\ l \in set\ (trail\ S') \longrightarrow l > 0$
using *assms* **apply** (*induct rule*: $cdcl_W$*-all-induct-lev2*)
by (*auto dest*: *union-in-get-all-marked-decomposition-is-subset simp*: $cdcl_W$*-M-level-inv-decomp*)

lemma $cdcl_W$*-o-no-more-init-clss*:
assumes
$cdcl_W$*-o* S S' **and**
inv: $cdcl_W$*-M-level-inv* S
shows *init-clss* S = *init-clss* S'
using *assms* **by** (*induct rule*: $cdcl_W$*-o-induct-lev2*) (*auto simp*: $cdcl_W$*-M-level-inv-decomp*)

lemma *tranclp-cdcl*$_W$*-o-no-more-init-clss*:
assumes
cdcl$_W$*-o*$^{++}$ S S' **and**
inv: *cdcl*$_W$*-M-level-inv* S
shows *init-clss* S = *init-clss* S'
using *assms* **apply** (*induct rule*: *tranclp.induct*)
by (*auto dest*: *cdcl*$_W$*-o-no-more-init-clss*
dest!: *tranclp-cdcl*$_W$*-consistent-inv dest*: *tranclp-mono-explicit*[*of cdcl*$_W$*-o - - cdcl*$_W$]
simp: *other*)

lemma *rtranclp-cdcl*$_W$*-o-no-more-init-clss*:
assumes
cdcl$_W$*-o*** S S' **and**
inv: *cdcl*$_W$*-M-level-inv* S
shows *init-clss* S = *init-clss* S'
using *assms* **unfolding** *rtranclp-unfold* **by** (*auto intro*: *tranclp-cdcl*$_W$*-o-no-more-init-clss*)

lemma *cdcl*$_W$*-init-clss*:
cdcl$_W$ S T $\Longrightarrow$ *cdcl*$_W$*-M-level-inv* S $\Longrightarrow$ *init-clss* S = *init-clss* T
by (*induct rule*: *cdcl*$_W$*-all-induct-lev2*) (*auto simp*: *cdcl*$_W$*-M-level-inv-def*)

lemma *rtranclp-cdcl*$_W$*-init-clss*:
cdcl$_W$** S T $\Longrightarrow$ *cdcl*$_W$*-M-level-inv* S $\Longrightarrow$ *init-clss* S = *init-clss* T
by (*induct rule*: *rtranclp-induct*) (*auto dest*: *cdcl*$_W$*-init-clss rtranclp-cdcl*$_W$*-consistent-inv*)

lemma *tranclp-cdcl*$_W$*-init-clss*:
cdcl$_W$$^{++}$ S T $\Longrightarrow$ *cdcl*$_W$*-M-level-inv* S $\Longrightarrow$ *init-clss* S = *init-clss* T
using *rtranclp-cdcl*$_W$*-init-clss*[*of* S T] **unfolding** *rtranclp-unfold* **by** *auto*

5.4.5 Learned Clause

This invariant shows that:

- the learned clauses are entailed by the initial set of clauses.
- the conflicting clause is entailed by the initial set of clauses.
- the marks are entailed by the clauses. A more precise version would be to show that either these marked are learned or are in the set of clauses

definition *cdcl*$_W$*-learned-clause* (S:: *'st*) $\longleftrightarrow$
(*init-clss* S $\models$*psm learned-clss* S
$\wedge$ ($\forall$ T. *conflicting* S = *Some* T $\longrightarrow$ *init-clss* S $\models$*pm* T)
$\wedge$ *set* (*get-all-mark-of-propagated* (*trail* S)) $\subseteq$ *set-mset* (*clauses* S))

lemma *cdcl*$_W$*-learned-clause-S0-cdcl*$_W$[*simp*]:
cdcl$_W$*-learned-clause* (*init-state* N)
unfolding *cdcl*$_W$*-learned-clause-def* **by** *auto*

lemma *cdcl*$_W$*-learned-clss*:
assumes
cdcl$_W$ S S' **and**
learned: *cdcl*$_W$*-learned-clause* S **and**

lev-inv: $cdcl_W$*-M-level-inv S*
shows $cdcl_W$*-learned-clause S'*
using *assms(1) lev-inv learned*
proof (*induct rule*: $cdcl_W$*-all-induct-lev2*)
case (*backtrack K i M1 M2 L D T*) **note** *decomp* = *this(1)* **and** *confl* = *this(3)* **and** *undef* = *this(6)*
and *T* =*this(7)*
show *?case*
using *decomp confl learned undef T lev-inv* **unfolding** $cdcl_W$*-learned-clause-def*
by (*auto dest!*: *get-all-marked-decomposition-exists-prepend*
simp: *clauses-def* $cdcl_W$*-M-level-inv-decomp dest*: *true-clss-clss-left-right*)
next
case (*resolve L C M D*) **note** *trail* = *this(1)* **and** *confl* = *this(2)* **and** *lvl* = *this(3)* **and**
T =*this(4)*
moreover
have *init-clss S* ⊨*psm learned-clss S*
using *learned trail* **unfolding** $cdcl_W$*-learned-clause-def clauses-def* **by** *auto*
then have *init-clss S* ⊨*pm C* + {#*L*#}
using *trail learned* **unfolding** $cdcl_W$*-learned-clause-def clauses-def*
by (*auto dest*: *true-clss-clss-in-imp-true-clss-cls*)
ultimately show *?case*
using *learned*
by (*auto dest*: *mk-disjoint-insert true-clss-clss-left-right*
simp add: $cdcl_W$*-learned-clause-def clauses-def*
intro: *true-clss-cls-union-mset-true-clss-cls-or-not-true-clss-cls-or*)
next
case (*restart T*)
then show *?case*
using *learned-clss-restart-state[of T]*
by (*auto dest!*: *get-all-marked-decomposition-exists-prepend*
simp: *clauses-def state-eq-def* $cdcl_W$*-learned-clause-def*
simp del: *state-simp*
dest: *true-clss-clssm-subsetE*)
next
case *propagate*
then show *?case* **using** *learned* **by** (*auto simp*: $cdcl_W$*-learned-clause-def clauses-def*)
next
case *conflict*
then show *?case* **using** *learned*
by (*auto simp*: $cdcl_W$*-learned-clause-def clauses-def true-clss-clss-in-imp-true-clss-cls*)
next
case *forget*
then show *?case*
using *learned* **by** (*auto simp*: $cdcl_W$*-learned-clause-def clauses-def split*: *split-if-asm*)
qed (*auto simp*: $cdcl_W$*-learned-clause-def clauses-def*)

lemma *rtranclp-*$cdcl_W$*-learned-clss*:
assumes
$cdcl_W^{**}$ *S S'* **and**
$cdcl_W$*-M-level-inv S*
$cdcl_W$*-learned-clause S*
shows $cdcl_W$*-learned-clause S'*
using *assms* **by** *induction* (*auto dest*: $cdcl_W$*-learned-clss intro*: *rtranclp-*$cdcl_W$*-consistent-inv*)

5.4.6 No alien atom in the state

This invariant means that all the literals are in the set of clauses.

```
definition no-strange-atm S′ ⟷ (
    (∀ T. conflicting S′ = Some T ⟶ atms-of T ⊆ atms-of-msu (init-clss S′))
  ∧ (∀ L mark. Propagated L mark ∈ set (trail S′)
      ⟶ atms-of ( mark) ⊆ atms-of-msu (init-clss S′))
  ∧ atms-of-msu (learned-clss S′) ⊆ atms-of-msu (init-clss S′)
  ∧ atm-of ‘ (lits-of (trail S′)) ⊆ atms-of-msu (init-clss S′))

lemma no-strange-atm-decomp:
  assumes no-strange-atm S
  shows conflicting S = Some T ⟹ atms-of T ⊆ atms-of-msu (init-clss S)
  and (∀ L mark. Propagated L mark ∈ set (trail S)
    ⟶ atms-of ( mark) ⊆ atms-of-msu (init-clss S))
  and atms-of-msu (learned-clss S) ⊆ atms-of-msu (init-clss S)
  and atm-of ‘ (lits-of (trail S)) ⊆ atms-of-msu (init-clss S)
  using assms unfolding no-strange-atm-def by blast+

lemma no-strange-atm-S0 [simp]: no-strange-atm (init-state N)
  unfolding no-strange-atm-def by auto

lemma cdcl_W-no-strange-atm-explicit:
  assumes
    cdcl_W S S′ and
    lev: cdcl_W-M-level-inv S and
    conf: ∀ T. conflicting S = Some T ⟶ atms-of T ⊆ atms-of-msu (init-clss S) and
    marked: ∀ L mark. Propagated L mark ∈ set (trail S)
      ⟶ atms-of mark ⊆ atms-of-msu (init-clss S) and
    learned: atms-of-msu (learned-clss S) ⊆ atms-of-msu (init-clss S) and
    trail: atm-of ‘ (lits-of (trail S)) ⊆ atms-of-msu (init-clss S)
  shows (∀ T. conflicting S′ = Some T ⟶ atms-of T ⊆ atms-of-msu (init-clss S′)) ∧
   (∀ L mark. Propagated L mark ∈ set (trail S′)
     ⟶ atms-of ( mark) ⊆ atms-of-msu (init-clss S′)) ∧
   atms-of-msu (learned-clss S′) ⊆ atms-of-msu (init-clss S′) ∧
   atm-of ‘ (lits-of (trail S′)) ⊆ atms-of-msu (init-clss S′) (is ?C S′ ∧ ?M S′ ∧ ?U S′ ∧ ?V S′)
  using assms(1,2)
proof (induct rule: cdcl_W-all-induct-lev2)
 case (propagate C L T) note C-L = this(1) and undef = this(3) and confl = this(4) and T =this(5)
  have ?C (cons-trail (Propagated L (C + {#L#})) S) using confl undef by auto
  moreover
    have atms-of (C + {#L#}) ⊆ atms-of-msu (init-clss S)
      by (metis (no-types) atms-of-atms-of-ms-mono atms-of-ms-union clauses-def mem-set-mset-iff
        C-L learned set-mset-union sup.orderE)
    then have ?M (cons-trail (Propagated L (C + {#L#})) S) using undef
      by (simp add: marked)
  moreover have ?U (cons-trail (Propagated L (C + {#L#})) S)
    using learned undef by auto
  moreover have ?V (cons-trail (Propagated L (C + {#L#})) S)
    using C-L learned trail undef unfolding clauses-def
    by (auto simp: in-plus-implies-atm-of-on-atms-of-ms)
  ultimately show ?case using T by auto
next
  case (decide L)
  then show ?case using learned marked conf trail unfolding clauses-def by auto
```

next
  case (*skip L C M D*)
  then show *?case* **using** *learned marked conf trail* **by** *auto*
next
  case (*conflict D T*) **note** *T* =*this*(*4*)
  have *D*: *atm-of* ‘ *set-mset D* ⊆ ⋃(*atms-of* ‘ (*set-mset* (*clauses S*)))
    using ‹*D* ∈# *clauses S*› **by** (*auto simp add*: *atms-of-def atms-of-ms-def*)
  moreover {
    fix *xa* :: ′*v literal*
    assume *a1*: *atm-of* ‘ *set-mset D* ⊆ (⋃*x*∈*set-mset* (*init-clss S*). *atms-of x*)
      ∪ (⋃*x*∈*set-mset* (*learned-clss S*). *atms-of x*)
    assume *a2*: (⋃*x*∈*set-mset* (*learned-clss S*). *atms-of x*) ⊆ (⋃*x*∈*set-mset* (*init-clss S*). *atms-of x*)
    assume *xa* ∈# *D*
    then have *atm-of xa* ∈ *UNION* (*set-mset* (*init-clss S*)) *atms-of*
      using *a2 a1* **by** (*metis* (*no-types*) *Un-iff atm-of-lit-in-atms-of atms-of-def subset-Un-eq*)
    then have ∃ *m*∈*set-mset* (*init-clss S*). *atm-of xa* ∈ *atms-of m*
      by *blast*
    } **note** *H* = *this*
  ultimately show *?case* **using** *conflict.prems T learned marked conf trail*
    unfolding *atms-of-def atms-of-ms-def clauses-def*
     by (*auto simp add*: *H*)
next
  case (*restart T*)
  then show *?case* **using** *learned marked conf trail* **by** *auto*
next
  case (*forget C T*) **note** *C* = *this*(*3*) **and** *C-le* = *this*(*4*) **and** *confl* = *this*(*5*) **and**
    T = *this*(*6*)
  have *H*: ⋀*L mark. Propagated L mark* ∈ *set* (*trail S*) ⟹ *atms-of mark* ⊆ *atms-of-msu* (*init-clss S*)
    using *marked* **by** *simp*
  show *?case* **unfolding** *clauses-def* **apply** *standard*
    using *conf T trail C* **unfolding** *clauses-def* **apply** (*auto dest*!: *H*)[]
    apply *standard*
     using *T trail C* **apply** (*auto dest*!: *H*)[]
    apply *standard*
      using *T learned C C-le atms-of-ms-remove-subset*[*of set-mset* (*learned-clss S*)] **apply** (*auto*)[]
    using *T trail C* **apply** (*auto simp*: *clauses-def lits-of-def*)[]
  done
next
  case (*backtrack K i M1 M2 L D T*) **note** *decomp* = *this*(*1*) **and** *confl* = *this*(*3*) **and** *undef*= *this*(*6*)
    and *T* =*this*(*7*)
  have *?C T*
    using *conf T decomp undef lev* **by** (*auto simp*: $cdcl_W$*-M-level-inv-decomp*)
  moreover have *set M1* ⊆ *set* (*trail S*)
    using *backtrack.hyps*(*1*) **by** *auto*
  then have *M*: *?M T*
    using *marked conf undef confl T decomp lev*
    by (*auto simp*: *image-subset-iff clauses-def* $cdcl_W$*-M-level-inv-decomp*)
  moreover have *?U T*
    using *learned decomp conf confl T undef lev* **unfolding** *clauses-def*
    by (*auto simp*: $cdcl_W$*-M-level-inv-decomp*)
  moreover have *?V T*
    using *M conf confl trail T undef decomp lev* **by** (*force simp*: $cdcl_W$*-M-level-inv-decomp*)
  ultimately show *?case* **by** *blast*
next
  case (*resolve L C M D T*) **note** *trail-S* = *this*(*1*) **and** *confl* = *this*(*2*) **and** *T* = *this*(*4*)

let *?T* = *update-conflicting* (*Some* (*remdups-mset* (*D* + *C*))) (*tl-trail* *S*)
have *?C ?T*
using *confl trail-S conf marked* **by** *simp*
moreover have *?M ?T*
using *confl trail-S conf marked* **by** *auto*
moreover have *?U ?T*
using *trail learned* **by** *auto*
moreover have *?V ?T*
using *confl trail-S trail* **by** *auto*
ultimately show *?case* **using** *T* **by** *auto*
qed

lemma $cdcl_W$*-no-strange-atm-inv*:
assumes $cdcl_W$ *S S'* **and** *no-strange-atm S* **and** $cdcl_W$*-M-level-inv S*
shows *no-strange-atm S'*
using $cdcl_W$*-no-strange-atm-explicit*[*OF assms*(*1*)] *assms*(*2*,*3*) **unfolding** *no-strange-atm-def* **by** *fast*

lemma *rtranclp-*$cdcl_W$*-no-strange-atm-inv*:
assumes $cdcl_W^{**}$ *S S'* **and** *no-strange-atm S* **and** $cdcl_W$*-M-level-inv S*
shows *no-strange-atm S'*
using *assms* **by** *induction* (*auto intro*: $cdcl_W$*-no-strange-atm-inv rtranclp-*$cdcl_W$*-consistent-inv*)

5.4.7 No duplicates all around

This invariant shows that there is no duplicate (no literal appearing twice in the formula). The last part could be proven using the previous invariant moreover.

definition *distinct-*$cdcl_W$*-state* (*S*::*'st*)
$\longleftrightarrow$ ((∀ *T*. *conflicting S* = *Some T* $\longrightarrow$ *distinct-mset T*)
∧ *distinct-mset-mset* (*learned-clss S*)
∧ *distinct-mset-mset* (*init-clss S*)
∧ (∀ *L mark*. (*Propagated L mark* ∈ *set* (*trail S*) $\longrightarrow$ *distinct-mset* (*mark*))))

lemma *distinct-*$cdcl_W$*-state-decomp*:
assumes *distinct-*$cdcl_W$*-state* (*S*::*'st*)
shows ∀ *T*. *conflicting S* = *Some T* $\longrightarrow$ *distinct-mset T*
and *distinct-mset-mset* (*learned-clss S*)
and *distinct-mset-mset* (*init-clss S*)
and ∀ *L mark*. (*Propagated L mark* ∈ *set* (*trail S*) $\longrightarrow$ *distinct-mset* (*mark*))
using *assms* **unfolding** *distinct-*$cdcl_W$*-state-def* **by** *blast+*

lemma *distinct-*$cdcl_W$*-state-decomp-2*:
assumes *distinct-*$cdcl_W$*-state* (*S*::*'st*)
shows *conflicting S* = *Some T* $\Longrightarrow$ *distinct-mset T*
using *assms* **unfolding** *distinct-*$cdcl_W$*-state-def* **by** *auto*

lemma *distinct-*$cdcl_W$*-state-S0-*$cdcl_W$[*simp*]:
distinct-mset-mset N $\Longrightarrow$ *distinct-*$cdcl_W$*-state* (*init-state N*)
unfolding *distinct-*$cdcl_W$*-state-def* **by** *auto*

lemma *distinct-*$cdcl_W$*-state-inv*:
assumes
$cdcl_W$ *S S'* **and**
$cdcl_W$*-M-level-inv S* **and**
distinct-$cdcl_W$*-state S*
shows *distinct-*$cdcl_W$*-state S'*

using *assms*
proof (*induct rule*: $cdcl_W$*-all-induct-lev2*)
case (*backtrack K i M1 M2 L D*)
then show *?case*
unfolding *distinct-*$cdcl_W$*-state-def*
by (*fastforce dest*: *get-all-marked-decomposition-incl simp*: $cdcl_W$*-M-level-inv-decomp*)
next
case *restart*
then show *?case* **unfolding** *distinct-*$cdcl_W$*-state-def distinct-mset-set-def clauses-def*
using *learned-clss-restart-state*[*of S*] **by** *auto*
next
case *resolve*
then show *?case*
by (*auto simp add*: *distinct-*$cdcl_W$*-state-def distinct-mset-set-def clauses-def*
distinct-mset-single-add
intro!: *distinct-mset-union-mset*)
qed (*auto simp add*: *distinct-*$cdcl_W$*-state-def distinct-mset-set-def clauses-def*)

lemma *rtanclp-distinct-*$cdcl_W$*-state-inv*:
assumes
$cdcl_W{}^{**}$ S S' **and**
$cdcl_W$*-M-level-inv S* **and**
distinct-$cdcl_W$*-state S*
shows *distinct-*$cdcl_W$*-state* S'
using *assms* **apply** (*induct rule*: *rtranclp-induct*)
using *distinct-*$cdcl_W$*-state-inv rtranclp-*$cdcl_W$*-consistent-inv* **by** *blast+*

5.4.8 Conflicts and co

This invariant shows that each mark contains a contradiction only related to the previously defined variable.

abbreviation *every-mark-is-a-conflict* :: *'st* ⇒ *bool* **where**
every-mark-is-a-conflict S ≡
∀ *L mark a b. a* @ *Propagated L mark* # *b* = (*trail S*)
⟶ (*b* ⊨*as CNot* (*mark* − {#*L*#}) ∧ *L* ∈# *mark*)

definition $cdcl_W$*-conflicting S* ≡
(∀ *T*. *conflicting S* = *Some T* ⟶ *trail S* ⊨*as CNot T*)
∧ *every-mark-is-a-conflict S*

lemma *backtrack-atms-of-D-in-M1*:
fixes *M1* :: (*'v*, *nat*, *'v clause*) *ann-literals*
assumes
inv: $cdcl_W$*-M-level-inv S* **and**
undef: *undefined-lit M1 L* **and**
i: *get-maximum-level* (*trail S*) *D* = *i* **and**
decomp: (*Marked K* (*Suc i*) # *M1*, *M2*)
∈ *set* (*get-all-marked-decomposition* (*trail S*)) **and**
S-lvl: *backtrack-lvl S* = *get-maximum-level* (*trail S*) (*D* + {#*L*#}) **and**
S-confl: *conflicting S* = *Some* (*D* + {#*L*#}) **and**
undef: *undefined-lit M1 L* **and**
T: *T* ∼ (*cons-trail* (*Propagated L* (*D*+{#*L*#}))
(*reduce-trail-to M1*
(*add-learned-cls* (*D* + {#*L*#})
(*update-backtrack-lvl i*

(*update-conflicting None S*))))) **and**
confl: ∀ *T*. *conflicting S* = *Some T* ⟶ *trail S* ⊨*as CNot T*
shows *atms-of D* ⊆ *atm-of* ‘ *lits-of* (*tl* (*trail T*))
proof (*rule ccontr*)
let *?k* = *get-maximum-level* (*trail S*) (*D* + {#*L*#})
have *trail S* ⊨*as CNot D* **using** *confl S-confl* **by** *auto*
then have *vars-of-D*: *atms-of D* ⊆ *atm-of* ‘ *lits-of* (*trail S*) **unfolding** *atms-of-def*
by (*meson image-subsetI mem-set-mset-iff true-annots-CNot-all-atms-defined*)

obtain *M0* **where** *M*: *trail S* = *M0* @ *M2* @ *Marked K* (*Suc i*) # *M1*
using *decomp* **by** *auto*

have *max*: *get-maximum-level* (*trail S*) (*D* + {#*L*#})
= *length* (*get-all-levels-of-marked* (*M0* @ *M2* @ *Marked K* (*Suc i*) # *M1*))
using *inv* **unfolding** $cdcl_W$*-M-level-inv-def S-lvl M* **by** *simp*
assume *a*: ¬ *?thesis*
then obtain *L′* **where**
L′: *L′* ∈ *atms-of D* **and**
L′-notin-M1: *L′* ∉ *atm-of* ‘ *lits-of M1*
using *T undef decomp inv* **by** (*auto simp*: $cdcl_W$*-M-level-inv-decomp*)
then have *L′-in*: *L′* ∈ *atm-of* ‘ *lits-of* (*M0* @ *M2* @ *Marked K* (*i* + *1*) # [])
using *vars-of-D* **unfolding** *M* **by** *force*
then obtain *L″* **where**
L″ ∈# *D* **and**
L″: *L′* = *atm-of L″*
using *L′ L′-notin-M1* **unfolding** *atms-of-def* **by** *auto*
have *lev-L″*:
get-level (*trail S*) *L″* = *get-rev-level* (*Marked K* (*Suc i*) # *rev M2* @ *rev M0*) (*Suc i*) *L″*
using *L′-notin-M1 L″ M* **by** (*auto simp del*: *get-rev-level.simps*)
have *get-all-levels-of-marked* (*trail S*) = *rev* [*1*..<*1*+*?k*]
using *inv S-lvl* **unfolding** $cdcl_W$*-M-level-inv-def* **by** *auto*
then have *get-all-levels-of-marked* (*M0* @ *M2*)
= *rev* [*Suc* (*Suc i*)..<*Suc* (*get-maximum-level* (*trail S*) (*D* + {#*L*#}))]
unfolding *M* **by** (*auto simp*:*rev-swap*[*symmetric*] *dest*!: *append-cons-eq-upt-length-i-end*)

then have *M*: *get-all-levels-of-marked M0* @ *get-all-levels-of-marked M2*
= *rev* [*Suc* (*Suc i*)..<*Suc* (*length* (*get-all-levels-of-marked* (*M0* @ *M2* @ *Marked K* (*Suc i*) # *M1*)))]
unfolding *max* **unfolding** *M* **by** *simp*

have *get-rev-level* (*Marked K* (*Suc i*) # *rev* (*M0* @ *M2*)) (*Suc i*) *L″*
≥ *Min* (*set* ((*Suc i*) # *get-all-levels-of-marked* (*Marked K* (*Suc i*) # *rev* (*M0* @ *M2*))))
using *get-rev-level-ge-min-get-all-levels-of-marked*[*of L″*
rev (*M0* @ *M2* @ [*Marked K* (*Suc i*)]) *Suc i*] *L′-in*
unfolding *L″* **by** (*fastforce simp add*: *lits-of-def*)
also have *Min* (*set* ((*Suc i*) # *get-all-levels-of-marked* (*Marked K* (*Suc i*) # *rev* (*M0* @ *M2*))))
= *Min* (*set* ((*Suc i*) # *get-all-levels-of-marked* (*rev* (*M0* @ *M2*)))) **by** *auto*
also have . . . = *Min* (*set* ((*Suc i*) # *get-all-levels-of-marked M0* @ *get-all-levels-of-marked M2*))
by (*simp add*: *Un-commute*)
also have . . . = *Min* (*set* ((*Suc i*) # [*Suc* (*Suc i*)..<*2* + *length* (*get-all-levels-of-marked M0*)
+ (*length* (*get-all-levels-of-marked M2*) + *length* (*get-all-levels-of-marked M1*))]))
unfolding *M* **by** (*auto simp add*: *Un-commute*)
also have . . . = *Suc i* **by** (*auto intro*: *Min-eqI*)
finally have *get-rev-level* (*Marked K* (*Suc i*) # *rev* (*M0* @ *M2*)) (*Suc i*) *L″* ≥ *Suc i* **.**
then have *get-level* (*trail S*) *L″* ≥ *i* + *1*
using *lev-L″* **by** *simp*

then have *get-maximum-level* (*trail S*) $D \geq i + 1$
using *get-maximum-level-ge-get-level*[*OF* ‹$L'' \in\# D$›, *of trail S*] **by** *auto*
then show *False* **using** *i* **by** *auto*
qed

lemma *distinct-atms-of-incl-not-in-other*:
assumes
a1: *no-dup* (M @ M') **and** *a2*:
atms-of D $\subseteq$ *atm-of* ‘ *lits-of* M'
shows $\forall x \in$*atms-of D*. $x \notin$ *atm-of* ‘ *lits-of M*
proof −
{ fix *aa* :: *'a*
have *ff1*: $\bigwedge l$ *ms*. *undefined-lit ms l* $\vee$ *atm-of l*
$\in$ *set* (*map* (λm. *atm-of* (*lit-of* (*m*::(*'a*, *'b*, *'c*) *ann-literal*))) *ms*)
by (*simp add*: *defined-lit-map*)
have *ff2*: $\bigwedge a$. $a \notin$ *atms-of D* $\vee$ $a \in$ *atm-of* ‘ *lits-of* M'
using *a2* **by** (*meson subsetCE*)
have *ff3*: $\bigwedge a$. $a \notin$ *set* (*map* (λm. *atm-of* (*lit-of m*)) M')
$\vee$ $a \notin$ *set* (*map* (λm. *atm-of* (*lit-of m*)) *M*)
using *a1* **by** (*metis* (*lifting*) *IntI distinct-append empty-iff map-append*)
have $\forall L\ a\ f$. $\exists l$. ((*a*::*'a*) $\notin f$ ‘ L $\vee$ (*l*::*'a literal*) $\in L$) $\wedge$ ($a \notin f$ ‘ L $\vee$ $f\ l = a$)
by *blast*
then have *aa* $\notin$ *atms-of D* $\vee$ *aa* $\notin$ *atm-of* ‘ *lits-of M*
using *ff3 ff2 ff1* **by** (*metis* (*no-types*) *Marked-Propagated-in-iff-in-lits-of*) **}**
then show *?thesis*
by *blast*
qed

lemma $cdcl_W$*-propagate-is-conclusion*:
assumes
$cdcl_W$ S S' **and**
inv: $cdcl_W$*-M-level-inv S* **and**
decomp: *all-decomposition-implies-m* (*init-clss S*) (*get-all-marked-decomposition* (*trail S*)) **and**
learned: $cdcl_W$*-learned-clause S* **and**
confl: $\forall T$. *conflicting S* = *Some T* $\longrightarrow$ *trail S* $\models$*as CNot T* **and**
alien: *no-strange-atm S*
shows *all-decomposition-implies-m* (*init-clss* S') (*get-all-marked-decomposition* (*trail* S'))
using *assms*(*1*,*2*)
proof (*induct rule*: $cdcl_W$*-all-induct-lev2*)
case *restart*
then show *?case* **by** *auto*
next
case *forget*
then show *?case* **using** *decomp* **by** *auto*
next
case *conflict*
then show *?case* **using** *decomp* **by** *auto*
next
case (*resolve L C M D*) **note** *tr* = *this*(*1*) **and** T = *this*(*4*)
let *?decomp* = *get-all-marked-decomposition M*
have *M*: *set ?decomp* = *insert* (*hd ?decomp*) (*set* (*tl ?decomp*))
by (*cases ?decomp*) *auto*
show *?case*
using *decomp tr T* **unfolding** *all-decomposition-implies-def*
by (*cases hd* (*get-all-marked-decomposition M*))

```
      (auto simp: M)
next
  case (skip L C' M D) note tr = this(1) and T = this(5)
  have M: set (get-all-marked-decomposition M)
    = insert (hd (get-all-marked-decomposition M)) (set (tl (get-all-marked-decomposition M)))
    by (cases get-all-marked-decomposition M) auto
  show ?case
    using decomp tr T unfolding all-decomposition-implies-def
    by (cases hd (get-all-marked-decomposition M))
      (auto simp add: M)
next
  case decide note S = this(1) and undef = this(2) and T = this(4)
  show ?case using decomp T undef unfolding S all-decomposition-implies-def by auto
next
  case (propagate C L T) note propa = this(2) and undef = this(3) and T =this(5)
  obtain a y where ay: hd (get-all-marked-decomposition (trail S)) = (a, y)
    by (cases hd (get-all-marked-decomposition (trail S)))
  then have M: trail S = y @ a using get-all-marked-decomposition-decomp by blast
  have M': set (get-all-marked-decomposition (trail S))
    = insert (a, y) (set (tl (get-all-marked-decomposition (trail S))))
    using ay by (cases get-all-marked-decomposition (trail S)) auto
  have unmark a ∪ set-mset (init-clss S) ⊨ps unmark y
    using decomp ay unfolding all-decomposition-implies-def
    by (cases get-all-marked-decomposition (trail S)) fastforce+
  then have a-Un-N-M: unmark a ∪ set-mset (init-clss S)
    ⊨ps unmark (trail S)
    unfolding M by (auto simp add: all-in-true-clss-clss image-Un)

  have unmark a ∪ set-mset (init-clss S) ⊨p {#L#} (is ?I ⊨p -)
    proof (rule true-clss-cls-plus-CNot)
      show ?I ⊨p C + {#L#}
        using propa propagate.prems learned confl unfolding M
        by (metis Un-iff cdcl_W-learned-clause-def clauses-def mem-set-mset-iff propagate.hyps(1)
          set-mset-union true-clss-clss-in-imp-true-clss-cls true-clss-cs-mono-l2
          union-trus-clss-clss)
    next
      have (λm. {#lit-of m#}) ` set (trail S) ⊨ps CNot C
        using ‹(trail S) ⊨as CNot C› true-annots-true-clss-clss by blast
      then show ?I ⊨ps CNot C
        using a-Un-N-M true-clss-clss-left-right true-clss-clss-union-l-r by blast
    qed
  moreover have ⋀aa b.
      ∀ (Ls, seen)∈set (get-all-marked-decomposition (y @ a)).
        unmark Ls ∪ set-mset (init-clss S) ⊨ps unmark seen
    ⟹ (aa, b) ∈ set (tl (get-all-marked-decomposition (y @ a)))
    ⟹ unmark aa ∪ set-mset (init-clss S) ⊨ps unmark b
    by (metis (no-types, lifting) case-prod-conv get-all-marked-decomposition-never-empty-sym
      list.collapse list.set-intros(2))

  ultimately show ?case
    using decomp T undef unfolding ay all-decomposition-implies-def
    using M ‹unmark a ∪ set-mset (init-clss S) ⊨ps unmark y›
     ay by auto
next
  case (backtrack K i M1 M2 L D T) note decomp' = this(1) and lev-L = this(2) and conf = this(3)
```

```
and
    undef = this(6) and T = this(7)
  have ∀ l ∈ set M2. ¬is-marked l
    using get-all-marked-decomposition-snd-not-marked backtrack.hyps(1) by blast
  obtain M0 where M: trail S = M0 @ M2 @ Marked K (i + 1) # M1
    using decomp' by auto
  show ?case unfolding all-decomposition-implies-def
    proof
      fix x
      assume x ∈set (get-all-marked-decomposition (trail T))
      then have x: x ∈ set (get-all-marked-decomposition (Propagated L ((D + {#L#})) # M1))
        using T decomp' undef inv by (simp add: cdcl_W-M-level-inv-decomp)
      let ?m = get-all-marked-decomposition (Propagated L ((D + {#L#})) # M1)
      let ?hd = hd ?m
      let ?tl = tl ?m
      have x = ?hd ∨ x ∈ set ?tl
        using x by (cases ?m) auto
      moreover {
        assume x ∈ set ?tl
        then have x ∈ set (get-all-marked-decomposition (trail S))
          using tl-get-all-marked-decomposition-skip-some[of x] by (simp add: list.set-sel(2) M)
        then have case x of (Ls, seen) ⇒ unmark Ls
                ∪ set-mset (init-clss (T))
                ⊨ps unmark seen
          using decomp learned decomp confl alien inv T undef M
          unfolding all-decomposition-implies-def cdcl_W-M-level-inv-def
          by auto
      }
      moreover {
        assume x = ?hd
        obtain M1' M1'' where M1: hd (get-all-marked-decomposition M1) = (M1', M1'')
          by (cases hd (get-all-marked-decomposition M1))
        then have x': x = (M1', Propagated L ( (D + {#L#})) # M1'')
          using ‹x= ?hd› by auto
        have (M1', M1'') ∈ set (get-all-marked-decomposition (trail S))
          using M1[symmetric] hd-get-all-marked-decomposition-skip-some[OF M1[symmetric],
            of M0 @ M2 - i+1] unfolding M by fastforce
        then have 1: unmark M1' ∪ set-mset (init-clss S)
          ⊨ps unmark M1''
          using decomp unfolding all-decomposition-implies-def by auto
        moreover
          have trail S ⊨as CNot D using conf confl by auto
          then have vars-of-D: atms-of D ⊆ atm-of ‘ lits-of (trail S)
            unfolding atms-of-def
            by (meson image-subsetI mem-set-mset-iff true-annots-CNot-all-atms-defined)
          have vars-of-D: atms-of D ⊆ atm-of ‘ lits-of M1
            using backtrack-atms-of-D-in-M1[of S M1 L D i  K M2 T] backtrack inv conf confl
            by (auto simp: cdcl_W-M-level-inv-decomp)
          have no-dup (trail S) using inv by (auto simp: cdcl_W-M-level-inv-decomp)
          then have vars-in-M1:
            ∀ x ∈ atms-of D. x ∉ atm-of ‘ lits-of (M0 @ M2 @ Marked K (i + 1) # [])
            using vars-of-D distinct-atms-of-incl-not-in-other[of M0 @M2 @ Marked K (i + 1) # []
              M1]
            unfolding M by auto
          have M1 ⊨as CNot D
```

using *vars-in-M1 true-annots-remove-if-notin-vars*[*of M0* @ *M2* @ *Marked K* $(i + 1)$ # []
M1 CNot D] ‹*trail S* ⊨*as CNot D*› **unfolding** *M lits-of-def* **by** *simp*
have *M1* = *M1''* @ *M1'* **by** (*simp add*: *M1 get-all-marked-decomposition-decomp*)
have *TT*: *unmark M1'* ∪ *set-mset* (*init-clss S*) ⊨*ps CNot D*
using *true-annots-true-clss-cls*[*OF* ‹*M1* ⊨*as CNot D*›] *true-clss-clss-left-right*[*OF 1*,
of CNot D] **unfolding** ‹*M1* = *M1''* @ *M1'*› **by** (*auto simp add*: *inf-sup-aci*(*5*,*7*))
have *init-clss S* ⊨*pm D* + {#*L*#}
using *conf learned* $cdcl_W$*-learned-clause-def confl* **by** *blast*
then have *T'*: *unmark M1'* ∪ *set-mset* (*init-clss S*) ⊨*p D* + {#*L*#} **by** *auto*
have *atms-of* (*D* + {#*L*#}) ⊆ *atms-of-msu* (*clauses S*)
using *alien conf* **unfolding** *no-strange-atm-def clauses-def* **by** *auto*
then have *unmark M1'* ∪ *set-mset* (*init-clss S*) ⊨*p* {#*L*#}
using *true-clss-cls-plus-CNot*[*OF T' TT*] **by** *auto*
ultimately
have *case x of* (*Ls*, *seen*) ⇒ *unmark Ls*
∪ *set-mset* (*init-clss T*)
⊨*ps unmark seen* **using** *T' T decomp' undef inv* **unfolding** *x'*
by (*simp add*: $cdcl_W$*-M-level-inv-decomp*)
}
ultimately show *case x of* (*Ls*, *seen*) ⇒ *unmark Ls* ∪ *set-mset* (*init-clss T*)
⊨*ps unmark seen* **using** *T* **by** *auto*
qed
qed

lemma $cdcl_W$*-propagate-is-false*:
assumes
$cdcl_W$ *S S'* **and**
lev: $cdcl_W$*-M-level-inv S* **and**
learned: $cdcl_W$*-learned-clause S* **and**
decomp: *all-decomposition-implies-m* (*init-clss S*) (*get-all-marked-decomposition* (*trail S*)) **and**
confl: ∀ *T*. *conflicting S* = *Some T* ⟶ *trail S* ⊨*as CNot T* **and**
alien: *no-strange-atm S* **and**
mark-confl: *every-mark-is-a-conflict S*
shows *every-mark-is-a-conflict S'*
using *assms*(*1*,*2*)
proof (*induct rule*: $cdcl_W$*-all-induct-lev2*)
case (*propagate C L T*) **note** *undef* = *this*(*3*) **and** *T* =*this*(*5*)
show *?case*
proof (*intro allI impI*)
fix *L' mark a b*
assume *a* @ *Propagated L' mark* # *b* = *trail T*
then have (*a*=[] ∧ *L* = *L'* ∧ *mark* = *C* + {#*L*#} ∧ *b* = *trail S*)
∨ *tl a* @ *Propagated L' mark* # *b* = *trail S*
using *T undef* **by** (*cases a*) *fastforce+*
moreover {
assume *tl a* @ *Propagated L' mark* # *b* = *trail S*
then have *b* ⊨*as CNot* (*mark* − {#*L'*#}) ∧ *L'* ∈# *mark*
using *mark-confl* **by** *auto*
}
moreover {
assume *a*=[] **and** *L* = *L'* **and** *mark* = *C* + {#*L*#} **and** *b* = *trail S*
then have *b* ⊨*as CNot* (*mark* − {#*L*#}) ∧ *L* ∈# *mark*
using ‹*trail S* ⊨*as CNot C*› **by** *auto*
}
ultimately show *b* ⊨*as CNot* (*mark* − {#*L'*#}) ∧ *L'* ∈# *mark* **by** *blast*

qed
next
case (*decide L*) **note** *undef*[*simp*] = *this*(*2*) **and** *T* = *this*(*4*)
have $\bigwedge$*a La mark b. a* @ *Propagated La mark* # *b* = *Marked L* (*backtrack-lvl S*+*1*) # *trail S*
$\Longrightarrow$ *tl a* @ *Propagated La mark* # *b* = *trail S* **by** (*case-tac a, auto*)
then show *?case* **using** *mark-confl T* **unfolding** *decide.hyps*(*1*) **by** *fastforce*
next
case (*skip L C' M D T*) **note** *tr* = *this*(*1*) **and** *T* = *this*(*5*)
show *?case*
proof (*intro allI impI*)
fix *L' mark a b*
assume *a* @ *Propagated L' mark* # *b* = *trail T*
then have *a* @ *Propagated L' mark* # *b* = *M* **using** *tr T* **by** *simp*
then have (*Propagated L C'* # *a*) @ *Propagated L' mark* # *b* = *Propagated L C'* # *M* **by** *auto*
moreover have ∀ *La mark a b. a* @ *Propagated La mark* # *b* = *Propagated L C'* # *M*
⟶ *b* ⊨*as CNot* (*mark* − {#*La*#}) ∧ *La* ∈# *mark*
using *mark-confl* **unfolding** *skip.hyps*(*1*) **by** *simp*
ultimately show *b* ⊨*as CNot* (*mark* − {#*L'*#}) ∧ *L'* ∈# *mark* **by** *blast*
qed
next
case (*conflict D*)
then show *?case* **using** *mark-confl* **by** *simp*
next
case (*resolve L C M D T*) **note** *tr-S* = *this*(*1*) **and** *T* = *this*(*4*)
show *?case* **unfolding** *resolve.hyps*(*1*)
proof (*intro allI impI*)
fix *L' mark a b*
assume *a* @ *Propagated L' mark* # *b* = *trail T*
then have *Propagated L* ((*C* + {#*L*#})) # *M*
= (*Propagated L* ((*C* + {#*L*#})) # *a*) @ *Propagated L' mark* # *b*
using *T tr-S* **by** *auto*
then show *b* ⊨*as CNot* (*mark* − {#*L'*#}) ∧ *L'* ∈# *mark*
using *mark-confl* **unfolding** *resolve.hyps*(*1*) **by** *presburger*
qed
next
case *restart*
then show *?case* **by** *auto*
next
case *forget*
then show *?case* **using** *mark-confl* **by** *auto*
next
case (*backtrack K i M1 M2 L D T*) **note** *decomp* = *this*(*1*) **and** *conf* = *this*(*3*) **and** *undef* = *this*(*6*)
and
T =*this*(*7*)
have ∀ *l* ∈ *set M2*. ¬*is-marked l*
using *get-all-marked-decomposition-snd-not-marked backtrack.hyps*(*1*) **by** *blast*
obtain *M0* **where** *M*: *trail S* = *M0* @ *M2* @ *Marked K* (*i* + *1*) # *M1*
using *backtrack.hyps*(*1*) **by** *auto*
have [*simp*]: *trail* (*reduce-trail-to M1* (*add-learned-cls* (*D* + {#*L*#})
(*update-backtrack-lvl i* (*update-conflicting None S*)))) = *M1*
using *decomp lev* **by** (*auto simp*: $cdcl_W$*-M-level-inv-decomp*)
show *?case*
proof (*intro allI impI*)
fix *La mark a b*
assume *a* @ *Propagated La mark* # *b* = *trail T*

then have (a = [] ∧ *Propagated La mark* = *Propagated L* (D + {#L#}) ∧ b = *M1*)
∨ *tl a* @ *Propagated La mark* # b = *M1*
using *M T decomp undef* **by** (*cases a*) (*auto*)
moreover {
assume *A*: a = [] **and**
P: *Propagated La mark* = *Propagated L* ((D + {#L#})) **and**
b: b = *M1*
have *trail S* ⊨*as CNot D* **using** *conf confl* **by** *auto*
then have *vars-of-D*: *atms-of D* ⊆ *atm-of* ‘ *lits-of* (*trail S*)
unfolding *atms-of-def*
by (*meson image-subsetI mem-set-mset-iff true-annots-CNot-all-atms-defined*)
have *vars-of-D*: *atms-of D* ⊆ *atm-of* ‘ *lits-of M1*
using *backtrack-atms-of-D-in-M1*[*of S M1 L D i K M2 T*] *T backtrack lev confl* **by** *auto*
have *no-dup* (*trail S*) **using** *lev* **by** (*auto simp*: $cdcl_W$*-M-level-inv-decomp*)
then have *vars-in-M1*: ∀ x ∈ *atms-of D*. x ∉
atm-of ‘ *lits-of* (*M0* @ *M2* @ *Marked K* (i + *1*) # [])
using *vars-of-D distinct-atms-of-incl-not-in-other*[*of M0* @ *M2* @ *Marked K* (i + *1*) # []
M1] **unfolding** *M* **by** *auto*
have *M1* ⊨*as CNot D*
using *vars-in-M1 true-annots-remove-if-notin-vars*[*of M0* @ *M2* @ *Marked K* (i + *1*) # [] *M1*
CNot D] ‹*trail S* ⊨*as CNot D*› **unfolding** *M lits-of-def* **by** *simp*
then have b ⊨*as CNot* (*mark* − {#*La*#}) ∧ *La* ∈# *mark*
using *P b* **by** *auto*
}
moreover {
assume *tl a* @ *Propagated La mark* # b = *M1*
then obtain c' **where** c' @ *Propagated La mark* # b = *trail S* **unfolding** *M* **by** *auto*
then have b ⊨*as CNot* (*mark* − {#*La*#}) ∧ *La* ∈# *mark*
using *mark-confl* **by** *blast*
}
ultimately show b ⊨*as CNot* (*mark* − {#*La*#}) ∧ *La* ∈# *mark* **by** *fast*
qed
qed

lemma $cdcl_W$*-conflicting-is-false*:
assumes
$cdcl_W$ S S' **and**
M-lev: $cdcl_W$*-M-level-inv S* **and**
confl-inv: ∀ T. *conflicting S* = *Some T* ⟶ *trail S* ⊨*as CNot T* **and**
marked-confl: ∀ *L mark a b*. *a* @ *Propagated L mark* # *b* = (*trail S*)
⟶ (*b* ⊨*as CNot* (*mark* − {#*L*#}) ∧ *L* ∈# *mark*) **and**
dist: *distinct*-$cdcl_W$*-state S*
shows ∀ T. *conflicting* S' = *Some T* ⟶ *trail* S' ⊨*as CNot T*
using *assms*(*1*,*2*)
proof (*induct rule*: $cdcl_W$*-all-induct-lev2*)
case (*skip L* C' *M D*) **note** *tr-S* = *this*(*1*) **and** T =*this*(*5*)
then have *Propagated L* C' # *M* ⊨*as CNot D* **using** *assms skip* **by** *auto*
moreover
have L ∉# D
proof (*rule ccontr*)
assume ¬ *?thesis*
then have − L ∈ *lits-of M*
using *in-CNot-implies-uminus*(*2*)[*of D L Propagated L* C' # *M*]
‹*Propagated L* C' # *M* ⊨*as CNot D*› **by** *simp*
then show *False*

by (*metis M-lev* $cdcl_W$*-M-level-inv-decomp*(*1*) *consistent-interp-def insert-iff lits-of-cons ann-literal.sel*(*2*) *skip.hyps*(*1*)))
qed
ultimately show *?case*
using *skip.hyps*(*1−3*) *true-annots-CNot-lit-of-notin-skip T* **unfolding** $cdcl_W$*-M-level-inv-def*
by *fastforce*
next
case (*resolve L C M D T*) **note** *tr* = *this*(*1*) **and** *confl* = *this*(*2*) **and** *T* = *this*(*4*)
show *?case*
proof (*intro allI impI*)
fix T'
have *tl* (*trail S*) $\models$*as CNot C* **using** *tr assms*(*4*) **by** *fastforce*
moreover
have *distinct-mset* (*D* + {#− *L*#}) **using** *confl dist*
unfolding *distinct-cdcl*$_W$*-state-def* **by** *auto*
then have −*L* ∉# *D* **unfolding** *distinct-mset-def* **by** *auto*
have *M* $\models$*as CNot D*
proof −
have *Propagated L* ((*C* + {#*L*#})) # *M* $\models$*as CNot D* ∪ *CNot* {#− *L*#}
using *confl tr confl-inv* **by** *force*
then show *?thesis*
using *M-lev* ‹− *L* ∉# *D*› *tr true-annots-lit-of-notin-skip*
unfolding $cdcl_W$*-M-level-inv-def* **by** *force*
qed
moreover assume *conflicting T* = *Some* T'
ultimately
show *trail T* $\models$*as CNot* T'
using *tr T* **by** *auto*
qed
qed (*auto simp*: *assms*(*2*) $cdcl_W$*-M-level-inv-decomp*)

lemma $cdcl_W$*-conflicting-decomp*:
assumes $cdcl_W$*-conflicting S*
shows ∀ *T*. *conflicting S* = *Some T* ⟶ *trail S* $\models$*as CNot T*
and ∀ *L mark a b*. *a* @ *Propagated L mark* # *b* = (*trail S*)
⟶ (*b* $\models$*as CNot* (*mark* − {#*L*#}) ∧ *L* ∈# *mark*)
using *assms* **unfolding** $cdcl_W$*-conflicting-def* **by** *blast+*

lemma $cdcl_W$*-conflicting-decomp2*:
assumes $cdcl_W$*-conflicting S* **and** *conflicting S* = *Some T*
shows *trail S* $\models$*as CNot T*
using *assms* **unfolding** $cdcl_W$*-conflicting-def* **by** *blast+*

lemma $cdcl_W$*-conflicting-decomp2'*:
assumes
$cdcl_W$*-conflicting S* **and**
conflicting S = *Some D*
shows *trail S* $\models$*as CNot D*
using *assms* **unfolding** $cdcl_W$*-conflicting-def* **by** *auto*

lemma $cdcl_W$*-conflicting-S0-cdcl*$_W$[*simp*]:
$cdcl_W$*-conflicting* (*init-state N*)
unfolding $cdcl_W$*-conflicting-def* **by** *auto*

5.4.9 Putting all the invariants together

lemma *cdcl$_W$-all-inv*:
  assumes *cdcl$_W$*: *cdcl$_W$ S S'* **and**
  1: *all-decomposition-implies-m* (*init-clss S*) (*get-all-marked-decomposition* (*trail S*)) **and**
  2: *cdcl$_W$-learned-clause S* **and**
  4: *cdcl$_W$-M-level-inv S* **and**
  5: *no-strange-atm S* **and**
  7: *distinct-cdcl$_W$-state S* **and**
  8: *cdcl$_W$-conflicting S*
  shows *all-decomposition-implies-m* (*init-clss S'*) (*get-all-marked-decomposition* (*trail S'*))
  and *cdcl$_W$-learned-clause S'*
  and *cdcl$_W$-M-level-inv S'*
  and *no-strange-atm S'*
  and *distinct-cdcl$_W$-state S'*
  and *cdcl$_W$-conflicting S'*
proof −
  show *S1*: *all-decomposition-implies-m* (*init-clss S'*) (*get-all-marked-decomposition* (*trail S'*))
    using *cdcl$_W$-propagate-is-conclusion*[*OF cdcl$_W$ 4 1 2 - 5*] *8* **unfolding** *cdcl$_W$-conflicting-def*
    by *blast*
  show *S2*: *cdcl$_W$-learned-clause S'* **using** *cdcl$_W$-learned-clss*[*OF cdcl$_W$ 2 4*] **.**
  show *S4*: *cdcl$_W$-M-level-inv S'* **using** *cdcl$_W$-consistent-inv*[*OF cdcl$_W$ 4*] **.**
  show *S5*: *no-strange-atm S'* **using** *cdcl$_W$-no-strange-atm-inv*[*OF cdcl$_W$ 5 4*] **.**
  show *S7*: *distinct-cdcl$_W$-state S'* **using** *distinct-cdcl$_W$-state-inv*[*OF cdcl$_W$ 4 7*] **.**
  show *S8*: *cdcl$_W$-conflicting S'*
    using *cdcl$_W$-conflicting-is-false*[*OF cdcl$_W$ 4 - - 7*] *8* *cdcl$_W$-propagate-is-false*[*OF cdcl$_W$ 4 2 1 - 5*]
    unfolding *cdcl$_W$-conflicting-def* **by** *fast*
qed

lemma *rtranclp-cdcl$_W$-all-inv*:
  assumes
    cdcl$_W$: *rtranclp cdcl$_W$ S S'* **and**
    1: *all-decomposition-implies-m* (*init-clss S*) (*get-all-marked-decomposition* (*trail S*)) **and**
    2: *cdcl$_W$-learned-clause S* **and**
    4: *cdcl$_W$-M-level-inv S* **and**
    5: *no-strange-atm S* **and**
    7: *distinct-cdcl$_W$-state S* **and**
    8: *cdcl$_W$-conflicting S*
  shows
    all-decomposition-implies-m (*init-clss S'*) (*get-all-marked-decomposition* (*trail S'*)) **and**
    cdcl$_W$-learned-clause S' **and**
    cdcl$_W$-M-level-inv S' **and**
    no-strange-atm S' **and**
    distinct-cdcl$_W$-state S' **and**
    cdcl$_W$-conflicting S'
   using *assms*
proof (*induct rule*: *rtranclp-induct*)
  case *base*
    case *1* **then show** *?case* **by** *blast*
    case *2* **then show** *?case* **by** *blast*
    case *3* **then show** *?case* **by** *blast*
    case *4* **then show** *?case* **by** *blast*
    case *5* **then show** *?case* **by** *blast*
    case *6* **then show** *?case* **by** *blast*
next

case (*step* S' S'') **note** H = *this*
case *1* **with** $H(3-7)$[*OF this*$(1-6)$] **show** *?case* **using** $cdcl_W$*-all-inv*[*OF* $H(2)$] H **by** *presburger*
case *2* **with** $H(3-7)$[*OF this*$(1-6)$] **show** *?case* **using** $cdcl_W$*-all-inv*[*OF* $H(2)$] H **by** *presburger*
case *3* **with** $H(3-7)$[*OF this*$(1-6)$] **show** *?case* **using** $cdcl_W$*-all-inv*[*OF* $H(2)$] H **by** *presburger*
case *4* **with** $H(3-7)$[*OF this*$(1-6)$] **show** *?case* **using** $cdcl_W$*-all-inv*[*OF* $H(2)$] H **by** *presburger*
case *5* **with** $H(3-7)$[*OF this*$(1-6)$] **show** *?case* **using** $cdcl_W$*-all-inv*[*OF* $H(2)$] H **by** *presburger*
case *6* **with** $H(3-7)$[*OF this*$(1-6)$] **show** *?case* **using** $cdcl_W$*-all-inv*[*OF* $H(2)$] H **by** *presburger*
qed

lemma *all-invariant-S0-*$cdcl_W$:
assumes *distinct-mset-mset N*
shows *all-decomposition-implies-m* (*init-clss* (*init-state N*)) (*get-all-marked-decomposition* (*trail* (*init-state N*)))
and $cdcl_W$*-learned-clause* (*init-state N*)
and $\forall$ *T*. *conflicting* (*init-state N*) = *Some T* $\longrightarrow$ (*trail* (*init-state N*))$\models$*as CNot T*
and *no-strange-atm* (*init-state N*)
and *consistent-interp* (*lits-of* (*trail* (*init-state N*)))
and $\forall$ *L mark a b*. *a* @ *Propagated L mark* # *b* = *trail* (*init-state N*) $\longrightarrow$ (*b* $\models$*as CNot* (*mark* − {#*L*#}) $\wedge$ *L* $\in$# *mark*)
and *distinct-*$cdcl_W$*-state* (*init-state N*)
using *assms* **by** *auto*

lemma $cdcl_W$*-only-propagated-vars-unsat*:
assumes
marked: $\forall x \in$ *set M*. $\neg$ *is-marked x* **and**
DN: *D* $\in$# *clauses S* **and**
D: *M* $\models$*as CNot D* **and**
inv: *all-decomposition-implies-m N* (*get-all-marked-decomposition M*) **and**
state: *state S* = (*M*, *N*, *U*, *k*, *C*) **and**
learned-cl: $cdcl_W$*-learned-clause S* **and**
atm-incl: *no-strange-atm S*
shows *unsatisfiable* (*set-mset N*)
proof (*rule ccontr*)
assume $\neg$ *unsatisfiable* (*set-mset N*)
then obtain *I* **where**
I: *I* $\models$*s set-mset N* **and**
cons: *consistent-interp I* **and**
tot: *total-over-m I* (*set-mset N*)
unfolding *satisfiable-def* **by** *auto*
have *atms-of-msu N* $\cup$ *atms-of-msu U* = *atms-of-msu N*
using *atm-incl state* **unfolding** *total-over-m-def no-strange-atm-def*
by (*auto simp add*: *clauses-def*)
then have *total-over-m I* (*set-mset N*) **using** *tot* **unfolding** *total-over-m-def* **by** *auto*
moreover have *N* $\models$*psm U* **using** *learned-cl state* **unfolding** $cdcl_W$*-learned-clause-def* **by** *auto*
ultimately have *I-D*: *I* $\models$ *D*
using *I DN cons state* **unfolding** *true-clss-clss-def true-clss-def Ball-def*
by (*metis Un-iff* ‹*atms-of-msu N* $\cup$ *atms-of-msu U* = *atms-of-msu N*› *atms-of-ms-union clauses-def mem-set-mset-iff prod.inject set-mset-union total-over-m-def*)

have *l0*: $\{\{\#$*lit-of L*$\#\}$ $|$*L. is-marked L* $\wedge$ *L* $\in$ *set M*$\}$ $= \{\}$ **using** *marked* **by** *auto*
have *atms-of-ms* (*set-mset N* $\cup$ *unmark M*) $=$ *atms-of-msu N*
using *atm-incl state* **unfolding** *no-strange-atm-def* **by** *auto*
then have *total-over-m I* (*set-mset N* $\cup$ ($\lambda a.$ $\{\#$*lit-of a*$\#\}$) ' (*set M*))
using *tot* **unfolding** *total-over-m-def* **by** *auto*
then have *I* $\models s$ ($\lambda a.$ $\{\#$*lit-of a*$\#\}$) ' (*set M*)
using *all-decomposition-implies-propagated-lits-are-implied*[*OF inv*] *cons I*
unfolding *true-clss-clss-def l0* **by** *auto*
then have *IM*: *I* $\models s$ *unmark M* **by** *auto*
{
fix *K*
assume *K* $\in\#$ *D*
then have $-K \in$ *lits-of M*
using *D* **unfolding** *true-annots-def Ball-def CNot-def true-annot-def true-cls-def true-lit-def Bex-mset-def* **by** (*metis* (*mono-tags*, *lifting*) *count-single less-not-refl mem-Collect-eq*)
then have $-K \in I$ **using** *IM true-clss-singleton-lit-of-implies-incl lits-of-def* **by** *fastforce*
}
then have $\neg$ *I* $\models$ *D* **using** *cons* **unfolding** *true-cls-def true-lit-def consistent-interp-def* **by** *auto*
then show *False* **using** *I-D* **by** *blast*
qed

We have actually a much stronger theorem, namely *all-decomposition-implies ?N* (*get-all-marked-decomposition ?M*) $\Longrightarrow$ *?N* $\cup$ $\{\{\#$*lit-of L*$\#\}$ $|$*L. is-marked L* $\wedge$ *L* $\in$ *set ?M*$\}$ $\models ps$ *unmark ?M*, that show that the only choices we made are marked in the formula

lemma
assumes *all-decomposition-implies-m N* (*get-all-marked-decomposition M*)
and $\forall m \in$ *set M*. $\neg$*is-marked m*
shows *set-mset N* $\models ps$ *unmark M*
proof $-$
have *T*: $\{\{\#$*lit-of L*$\#\}$ $|$*L. is-marked L* $\wedge$ *L* $\in$ *set M*$\}$ $= \{\}$ **using** *assms*(*2*) **by** *auto*
then show *?thesis*
using *all-decomposition-implies-propagated-lits-are-implied*[*OF assms*(*1*)] **unfolding** *T* **by** *simp*
qed

lemma *conflict-with-false-implies-unsat*:
assumes
$cdcl_W$: $cdcl_W$ *S S'* **and**
lev: $cdcl_W$*-M-level-inv S* **and**
[*simp*]: *conflicting S'* $=$ *Some* $\{\#\}$ **and**
learned: $cdcl_W$*-learned-clause S*
shows *unsatisfiable* (*set-mset* (*init-clss S*))
using *assms*
proof $-$
have $cdcl_W$*-learned-clause S'* **using** $cdcl_W$*-learned-clss* $cdcl_W$ *learned lev* **by** *auto*
then have *init-clss S'* $\models pm$ $\{\#\}$ **using** *assms*(*3*) **unfolding** $cdcl_W$*-learned-clause-def* **by** *auto*
then have *init-clss S* $\models pm$ $\{\#\}$
using $cdcl_W$*-init-clss*[*OF assms*(*1*) *lev*] **by** *auto*
then show *?thesis* **unfolding** *satisfiable-def true-clss-cls-def* **by** *auto*
qed

lemma *conflict-with-false-implies-terminated*:
assumes $cdcl_W$ *S S'*
and *conflicting S* $=$ *Some* $\{\#\}$

shows *False*
using *assms* **by** (*induct rule*: *cdcl$_W$-all-induct*) *auto*

5.4.10 No tautology is learned

This is a simple consequence of all we have shown previously. It is not strictly necessary, but helps finding a better bound on the number of learned clauses.

lemma *learned-clss-are-not-tautologies*:
  assumes
    cdcl$_W$ S S' **and**
    lev: *cdcl$_W$-M-level-inv S* **and**
    conflicting: *cdcl$_W$-conflicting S* **and**
    no-tauto: $\forall$ *s* $\in$# *learned-clss S*. $\neg$*tautology s*
  shows $\forall$ *s* $\in$# *learned-clss S'*. $\neg$*tautology s*
  using *assms*
proof (*induct rule*: *cdcl$_W$-all-induct-lev2*)
  case (*backtrack K i M1 M2 L D*) **note** *confl* = *this*(*3*)
  have *consistent-interp* (*lits-of* (*trail S*)) **using** *lev* **by** (*auto simp*: *cdcl$_W$-M-level-inv-decomp*)
  moreover
    have *trail S* $\models$*as CNot* (*D* + {#*L*#})
      using *conflicting confl* **unfolding** *cdcl$_W$-conflicting-def* **by** *auto*
    then have *lits-of* (*trail S*) $\models$*s CNot* (*D* + {#*L*#}) **using** *true-annots-true-cls* **by** *blast*
  ultimately have $\neg$*tautology* (*D* + {#*L*#}) **using** *consistent-CNot-not-tautology* **by** *blast*
  then show *?case* **using** *backtrack no-tauto*
    by (*auto simp*: *cdcl$_W$-M-level-inv-decomp split*: *split-if-asm*)
next
  case *restart*
  then show *?case* **using** *learned-clss-restart-state state-eq-learned-clss no-tauto*
    by (*metis* (*no-types*, *lifting*) *ball-msetE ball-msetI mem-set-mset-iff set-mset-mono subsetCE*)
qed *auto*

definition *final-cdcl$_W$-state* (*S*:: *'st*)
  $\longleftrightarrow$ (*trail S* $\models$*asm init-clss S*
    $\vee$ (($\forall$ *L* $\in$ *set* (*trail S*). $\neg$*is-marked L*) $\wedge$
      ($\exists$ *C* $\in$# *init-clss S*. *trail S* $\models$*as CNot C*)))

definition *termination-cdcl$_W$-state* (*S*:: *'st*)
   $\longleftrightarrow$ (*trail S* $\models$*asm init-clss S*
     $\vee$ (($\forall$ *L* $\in$ *atms-of-msu* (*init-clss S*). *L* $\in$ *atm-of* ‘ *lits-of* (*trail S*))
       $\wedge$ ($\exists$ *C* $\in$# *init-clss S*. *trail S* $\models$*as CNot C*)))

5.5 CDCL Strong Completeness

fun *mapi* :: (*'a* $\Rightarrow$ *nat* $\Rightarrow$ *'b*) $\Rightarrow$ *nat* $\Rightarrow$ *'a list* $\Rightarrow$ *'b list* **where**
mapi - - [] = [] |
mapi f n (*x* # *xs*) = *f x n* # *mapi f* (*n* − *1*) *xs*

lemma *mark-not-in-set-mapi*[*simp*]: *L* $\notin$ *set M* $\Longrightarrow$ *Marked L k* $\notin$ *set* (*mapi Marked i M*)
  by (*induct M arbitrary*: *i*) *auto*

lemma *propagated-not-in-set-mapi*[*simp*]: *L* $\notin$ *set M* $\Longrightarrow$ *Propagated L k* $\notin$ *set* (*mapi Marked i M*)
  by (*induct M arbitrary*: *i*) *auto*

lemma *image-set-mapi*:
  f ‘ *set* (*mapi g i M*) = *set* (*mapi* (λ*x i*. *f* (*g x i*)) *i M*)

by (*induction M arbitrary*: *i*) *auto*

lemma *mapi-map-convert*:
$\forall x\ i\ j.\ f\ x\ i = f\ x\ j \Longrightarrow mapi\ f\ i\ M = map\ (\lambda x.\ f\ x\ 0)\ M$
by (*induction M arbitrary*: *i*) *auto*

lemma *defined-lit-mapi*: *defined-lit* (*mapi Marked i M*) $L \longleftrightarrow$ *atm-of* $L \in$ *atm-of* ‘ *set M*
by (*induction M*) (*auto simp*: *defined-lit-map image-set-mapi mapi-map-convert*)

lemma $cdcl_W$*-can-do-step*:
assumes
consistent-interp (*set M*) **and**
distinct M **and**
atm-of ‘ (*set M*) $\subseteq$ *atms-of-msu N*
shows $\exists S.$ *rtranclp* $cdcl_W$ (*init-state N*) *S*
$\wedge$ *state S* = (*mapi Marked* (*length M*) *M*, *N*, {#}, *length M*, *None*)
using *assms*
proof (*induct M*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons L M*) **note** *IH* = *this*(*1*)
have *consistent-interp* (*set M*) **and** *distinct M* **and** *atm-of* ‘ *set M* $\subseteq$ *atms-of-msu N*
using *Cons.prems*(*1*−*3*) **unfolding** *consistent-interp-def* **by** *auto*
then obtain *S* **where**
st: $cdcl_W{}^{**}$ (*init-state N*) *S* **and**
S: *state S* = (*mapi Marked* (*length M*) *M*, *N*, {#}, *length M*, *None*)
using *IH* **by** *auto*
let $?S_0$ = *incr-lvl* (*cons-trail* (*Marked L* (*length M* +*1*)) *S*)
have *undefined-lit* (*mapi Marked* (*length M*) *M*) *L*
using *Cons.prems*(*1*,*2*) **unfolding** *defined-lit-def consistent-interp-def* **by** *fastforce*
moreover have *init-clss S* = *N*
using *S* **by** *blast*
moreover have *atm-of* $L \in$ *atms-of-msu N* **using** *Cons.prems*(*3*) **by** *auto*
moreover have *undef*: *undefined-lit* (*trail S*) *L*
using *S* ‹*distinct* (*L*#*M*)› *calculation*(*1*) **by** (*auto simp*: *defined-lit-mapi defined-lit-map*)
ultimately have $cdcl_W$ *S* $?S_0$
using $cdcl_W$*.other*[*OF* $cdcl_W$*-o.decide*[*OF decide-rule*[*OF S*,
of L $?S_0$]]] *S* **by** (*auto simp*: *state-eq-def simp del*: *state-simp*)
then show *?case*
using *st S undef* **by** (*auto intro*!: *exI*[*of - * $?S_0$])
qed

lemma $cdcl_W$*-strong-completeness*:
assumes
set M $\models$*s set-mset N* **and**
consistent-interp (*set M*) **and**
distinct M **and**
atm-of ‘ (*set M*) $\subseteq$ *atms-of-msu N*
obtains *S* **where**
state S = (*mapi Marked* (*length M*) *M*, *N*, {#}, *length M*, *None*) **and**
rtranclp $cdcl_W$ (*init-state N*) *S* **and**
final-$cdcl_W$*-state S*
proof −
obtain *S* **where**

st: *rtranclp* $cdcl_W$ (*init-state N*) *S* **and**
S: *state S* = (*mapi Marked* (*length M*) *M*, *N*, {#}, *length M*, *None*)
using $cdcl_W$*-can-do-step*[*OF assms*(*2*−*4*)] **by** *auto*
have *lits-of* (*mapi Marked* (*length M*) *M*) = *set M*
by (*induct M*, *auto*)
then have *mapi Marked* (*length M*) *M* ⊨*asm N* **using** *assms*(*1*) *true-annots-true-cls* **by** *metis*
then have *final-*$cdcl_W$*-state S*
using *S* **unfolding** *final-*$cdcl_W$*-state-def* **by** *auto*
then show *?thesis* **using** *that st S* **by** *blast*
qed

5.6 Higher level strategy

The rules described previously do not lead to a conclusive state. We have to add a strategy.

5.6.1 Definition

lemma *tranclp-conflict-iff*[*iff*]:
full1 conflict S S' ⟷ *conflict S S'*
proof −
have *tranclp conflict S S'* ⟹ *conflict S S'*
unfolding *full1-def* **by** (*induct rule*: *tranclp.induct*) *force+*
then have *tranclp conflict S S'* ⟹ *conflict S S'* **by** (*meson rtranclpD*)
then show *?thesis* **unfolding** *full1-def* **by** (*metis conflictE option.simps*(*3*)
conflicting-update-conflicting state-eq-conflicting tranclp.intros(*1*))
qed

inductive $cdcl_W$*-cp* :: *'st* ⇒ *'st* ⇒ *bool* **where**
conflict'[*intro*]: *conflict S S'* ⟹ $cdcl_W$*-cp S S'* |
propagate': *propagate S S'* ⟹ $cdcl_W$*-cp S S'*

lemma *rtranclp-*$cdcl_W$*-cp-rtranclp-*$cdcl_W$:
$cdcl_W$*-cp*** *S T* ⟹ $cdcl_W{}^{**}$ *S T*
by (*induction rule*: *rtranclp-induct*) (*auto simp*: $cdcl_W$*-cp.simps dest*: $cdcl_W$*.intros*)

lemma $cdcl_W$*-cp-state-eq-compatible*:
assumes
$cdcl_W$*-cp S T* **and**
S ∼ *S'* **and**
T ∼ *T'*
shows $cdcl_W$*-cp S' T'*
using *assms*
apply (*induction*)
using *conflict-state-eq-compatible* **apply** *auto*[*1*]
using *propagate' propagate-state-eq-compatible* **by** *auto*

lemma *tranclp-*$cdcl_W$*-cp-state-eq-compatible*:
assumes
$cdcl_W$*-cp*$^{++}$ *S T* **and**
S ∼ *S'* **and**
T ∼ *T'*
shows $cdcl_W$*-cp*$^{++}$ *S' T'*
using *assms*
proof *induction*
case *base*

then show *?case*
using $cdcl_W$*-cp-state-eq-compatible* **by** *blast*
next
case (*step U V*)
obtain *ss* :: *'st* **where**
$cdcl_W$*-cp S ss* ∧ $cdcl_W$*-cp*** *ss U*
by (*metis* (*no-types*) *step*(*1*) *tranclpD*)
then show *?case*
by (*meson* $cdcl_W$*-cp-state-eq-compatible rtranclp.rtrancl-into-rtrancl rtranclp-into-tranclp2 state-eq-ref step*(*2*) *step*(*4*) *step*(*5*))
qed

lemma *option-full-*$cdcl_W$*-cp*:
conflicting S ≠ *None* ⟹ *full* $cdcl_W$*-cp S S*
unfolding *full-def rtranclp-unfold tranclp-unfold* **by** (*auto simp add*: $cdcl_W$*-cp.simps*)

lemma *skip-unique*:
skip S T ⟹ *skip S T'* ⟹ *T* ∼ *T'*
by (*fastforce simp*: *state-eq-def simp del*: *state-simp*)

lemma *resolve-unique*:
resolve S T ⟹ *resolve S T'* ⟹ *T* ∼ *T'*
by (*fastforce simp*: *state-eq-def simp del*: *state-simp*)

lemma $cdcl_W$*-cp-no-more-clauses*:
assumes $cdcl_W$*-cp S S'*
shows *clauses S* = *clauses S'*
using *assms* **by** (*induct rule*: $cdcl_W$*-cp.induct*) (*auto elim*!: *conflictE propagateE*)

lemma *tranclp-*$cdcl_W$*-cp-no-more-clauses*:
assumes $cdcl_W$*-cp*$^{++}$ *S S'*
shows *clauses S* = *clauses S'*
using *assms* **by** (*induct rule*: *tranclp.induct*) (*auto dest*: $cdcl_W$*-cp-no-more-clauses*)

lemma *rtranclp-*$cdcl_W$*-cp-no-more-clauses*:
assumes $cdcl_W$*-cp*** *S S'*
shows *clauses S* = *clauses S'*
using *assms* **by** (*induct rule*: *rtranclp-induct*) (*fastforce dest*: $cdcl_W$*-cp-no-more-clauses*)+

lemma *no-conflict-after-conflict*:
conflict S T ⟹ ¬*conflict T U*
by *fastforce*

lemma *no-propagate-after-conflict*:
conflict S T ⟹ ¬*propagate T U*
by *fastforce*

lemma *tranclp-*$cdcl_W$*-cp-propagate-with-conflict-or-not*:
assumes $cdcl_W$*-cp*$^{++}$ *S U*
shows (*propagate*$^{++}$ *S U* ∧ *conflicting U* = *None*)
∨ (∃ *T D*. *propagate*** *S T* ∧ *conflict T U* ∧ *conflicting U* = *Some D*)
proof −
have *propagate*$^{++}$ *S U* ∨ (∃ *T*. *propagate*** *S T* ∧ *conflict T U*)
using *assms* **by** *induction*
(*force simp*: $cdcl_W$*-cp.simps tranclp-into-rtranclp dest*: *no-conflict-after-conflict*

no-propagate-after-conflict)+
moreover
have *propagate*$^{++}$ *S U* $\Longrightarrow$ *conflicting U* = *None*
unfolding *tranclp-unfold-end* **by** *auto*
moreover
have $\bigwedge$*T*. *conflict T U* $\Longrightarrow$ $\exists$ *D*. *conflicting U* = *Some D*
by *auto*
ultimately show *?thesis* **by** *meson*
qed

lemma $cdcl_W$*-cp-conflicting-not-empty*[*simp*]: *conflicting S* = *Some D* $\Longrightarrow$ $\neg$$cdcl_W$*-cp S S'*
proof
assume $cdcl_W$*-cp S S'* **and** *conflicting S* = *Some D*
then show *False* **by** (*induct rule*: $cdcl_W$*-cp.induct*) *auto*
qed

lemma *no-step-*$cdcl_W$*-cp-no-conflict-no-propagate*:
assumes *no-step* $cdcl_W$*-cp S*
shows *no-step conflict S* **and** *no-step propagate S*
using *assms conflict'* **apply** *blast*
by (*meson assms conflict' propagate'*)

CDCL with the reasonable strategy: we fully propagate the conflict and propagate, then we apply any other possible rule $cdcl_W$*-o S S'* and re-apply conflict and propagate *full* $cdcl_W$*-cp S' S''*

inductive $cdcl_W$*-stgy* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **for** *S* :: *'st* **where**
conflict': *full1* $cdcl_W$*-cp S S'* $\Longrightarrow$ $cdcl_W$*-stgy S S'* |
other': $cdcl_W$*-o S S'* $\Longrightarrow$ *no-step* $cdcl_W$*-cp S* $\Longrightarrow$ *full* $cdcl_W$*-cp S' S''* $\Longrightarrow$ $cdcl_W$*-stgy S S''*

5.6.2 Invariants

These are the same invariants as before, but lifted

lemma $cdcl_W$*-cp-learned-clause-inv*:
assumes $cdcl_W$*-cp S S'*
shows *learned-clss S* = *learned-clss S'*
using *assms* **by** (*induct rule*: $cdcl_W$*-cp.induct*) *fastforce*+

lemma *rtranclp-*$cdcl_W$*-cp-learned-clause-inv*:
assumes $cdcl_W$*-cp*** *S S'*
shows *learned-clss S* = *learned-clss S'*
using *assms* **by** (*induct rule*: *rtranclp-induct*) (*fastforce dest*: $cdcl_W$*-cp-learned-clause-inv*)+

lemma *tranclp-*$cdcl_W$*-cp-learned-clause-inv*:
assumes $cdcl_W$*-cp*$^{++}$ *S S'*
shows *learned-clss S* = *learned-clss S'*
using *assms* **by** (*simp add*: *rtranclp-*$cdcl_W$*-cp-learned-clause-inv tranclp-into-rtranclp*)

lemma $cdcl_W$*-cp-backtrack-lvl*:
assumes $cdcl_W$*-cp S S'*
shows *backtrack-lvl S* = *backtrack-lvl S'*
using *assms* **by** (*induct rule*: $cdcl_W$*-cp.induct*) *fastforce*+

lemma *rtranclp-*$cdcl_W$*-cp-backtrack-lvl*:
assumes $cdcl_W$*-cp*** *S S'*
shows *backtrack-lvl S* = *backtrack-lvl S'*

using *assms* **by** (*induct rule*: *rtranclp-induct*) (*fastforce dest*: $cdcl_W$*-cp-backtrack-lvl*)+

lemma $cdcl_W$*-cp-consistent-inv*:
assumes $cdcl_W$*-cp* S S'
and $cdcl_W$*-M-level-inv* S
shows $cdcl_W$*-M-level-inv* S'
using *assms*
proof (*induct rule*: $cdcl_W$*-cp.induct*)
case (*conflict'*)
then show *?case* **using** $cdcl_W$*-consistent-inv* $cdcl_W$*.conflict* **by** *blast*
next
case (*propagate'* S S')
have $cdcl_W$ S S'
using *propagate'.hyps*(*1*) *propagate* **by** *blast*
then show $cdcl_W$*-M-level-inv* S'
using *propagate'.prems*(*1*) $cdcl_W$*-consistent-inv* *propagate* **by** *blast*
qed

lemma *full1-*$cdcl_W$*-cp-consistent-inv*:
assumes *full1* $cdcl_W$*-cp* S S'
and $cdcl_W$*-M-level-inv* S
shows $cdcl_W$*-M-level-inv* S'
using *assms* **unfolding** *full1-def*
proof −
have $cdcl_W$*-cp*$^{++}$ S S' **and** $cdcl_W$*-M-level-inv* S **using** *assms* **unfolding** *full1-def* **by** *auto*
then show *?thesis* **by** (*induct rule*: *tranclp.induct*) (*blast intro*: $cdcl_W$*-cp-consistent-inv*)+
qed

lemma *rtranclp-*$cdcl_W$*-cp-consistent-inv*:
assumes *rtranclp* $cdcl_W$*-cp* S S'
and $cdcl_W$*-M-level-inv* S
shows $cdcl_W$*-M-level-inv* S'
using *assms* **unfolding** *full1-def*
by (*induction rule*: *rtranclp-induct*) (*blast intro*: $cdcl_W$*-cp-consistent-inv*)+

lemma $cdcl_W$*-stgy-consistent-inv*:
assumes $cdcl_W$*-stgy* S S'
and $cdcl_W$*-M-level-inv* S
shows $cdcl_W$*-M-level-inv* S'
using *assms* **apply** (*induct rule*: $cdcl_W$*-stgy.induct*)
unfolding *full-unfold* **by** (*blast intro*: $cdcl_W$*-consistent-inv* *full1-*$cdcl_W$*-cp-consistent-inv*
$cdcl_W$*.other*)+

lemma *rtranclp-*$cdcl_W$*-stgy-consistent-inv*:
assumes $cdcl_W$*-stgy*** S S'
and $cdcl_W$*-M-level-inv* S
shows $cdcl_W$*-M-level-inv* S'
using *assms* **by** *induction* (*auto dest*!: $cdcl_W$*-stgy-consistent-inv*)

lemma $cdcl_W$*-cp-no-more-init-clss*:
assumes $cdcl_W$*-cp* S S'
shows *init-clss* S = *init-clss* S'
using *assms* **by** (*induct rule*: $cdcl_W$*-cp.induct*) *auto*

lemma *tranclp-*$cdcl_W$*-cp-no-more-init-clss*:

assumes $cdcl_W$*-cp*$^{++}$ *S S'*
shows *init-clss S = init-clss S'*
using *assms* **by** (*induct rule*: *tranclp.induct*) (*auto dest*: $cdcl_W$*-cp-no-more-init-clss*)

lemma $cdcl_W$*-stgy-no-more-init-clss*:
assumes $cdcl_W$*-stgy S S'* **and** $cdcl_W$*-M-level-inv S*
shows *init-clss S = init-clss S'*
using *assms*
apply (*induct rule*: $cdcl_W$*-stgy.induct*)
unfolding *full1-def full-def* **apply** (*blast dest*: *tranclp-*$cdcl_W$*-cp-no-more-init-clss*
tranclp-$cdcl_W$*-o-no-more-init-clss*)
by (*metis* $cdcl_W$*-o-no-more-init-clss rtranclp-unfold tranclp-*$cdcl_W$*-cp-no-more-init-clss*)

lemma *rtranclp-*$cdcl_W$*-stgy-no-more-init-clss*:
assumes $cdcl_W$*-stgy*** *S S'* **and** $cdcl_W$*-M-level-inv S*
shows *init-clss S = init-clss S'*
using *assms*
apply (*induct rule*: *rtranclp-induct*, *simp*)
using $cdcl_W$*-stgy-no-more-init-clss* **by** (*simp add*: *rtranclp-*$cdcl_W$*-stgy-consistent-inv*)

lemma $cdcl_W$*-cp-dropWhile-trail'*:
assumes $cdcl_W$*-cp S S'*
obtains *M* **where** *trail S' = M @ trail S* **and** ($\forall$ *l* $\in$ *set M*. $\neg$*is-marked l*)
using *assms* **by** *induction fastforce+*

lemma *rtranclp-*$cdcl_W$*-cp-dropWhile-trail'*:
assumes $cdcl_W$*-cp*** *S S'*
obtains *M* :: (*'v*, *nat*, *'v clause*) *ann-literal list* **where**
trail S' = M @ trail S **and** $\forall$ *l* $\in$ *set M*. $\neg$*is-marked l*
using *assms* **by** *induction* (*fastforce dest*!: $cdcl_W$*-cp-dropWhile-trail'*)+

lemma $cdcl_W$*-cp-dropWhile-trail*:
assumes $cdcl_W$*-cp S S'*
shows $\exists$ *M*. *trail S' = M @ trail S* $\wedge$ ($\forall$ *l* $\in$ *set M*. $\neg$*is-marked l*)
using *assms* **by** *induction fastforce+*

lemma *rtranclp-*$cdcl_W$*-cp-dropWhile-trail*:
assumes $cdcl_W$*-cp*** *S S'*
shows $\exists$ *M*. *trail S' = M @ trail S* $\wedge$ ($\forall$ *l* $\in$ *set M*. $\neg$*is-marked l*)
using *assms* **by** *induction* (*fastforce dest*: $cdcl_W$*-cp-dropWhile-trail*)+

This theorem can be seen a a termination theorem for $cdcl_W$*-cp*.

lemma *length-model-le-vars*:
assumes
no-strange-atm S **and**
no-d: *no-dup* (*trail S*) **and**
finite (*atms-of-msu* (*init-clss S*))
shows *length* (*trail S*) $\leq$ *card* (*atms-of-msu* (*init-clss S*))
proof −
obtain *M N U k D* **where** *S*: *state S* = (*M*, *N*, *U*, *k*, *D*) **by** (*cases state S*, *auto*)
have *finite* (*atm-of* ‘ *lits-of* (*trail S*))
using *assms*(*1*,*3*) **unfolding** *S* **by** (*auto simp add*: *finite-subset*)
have *length* (*trail S*) = *card* (*atm-of* ‘ *lits-of* (*trail S*))
using *no-dup-length-eq-card-atm-of-lits-of no-d* **by** *blast*
then show *?thesis* **using** *assms*(*1*) **unfolding** *no-strange-atm-def*

by (*auto simp add*: *assms*(*3*) *card-mono*)
qed

lemma $cdcl_W$*-cp-decreasing-measure*:
assumes
$cdcl_W$: $cdcl_W$*-cp S T* **and**
M-lev: $cdcl_W$*-M-level-inv S* **and**
alien: *no-strange-atm S*
shows (λS. *card* (*atms-of-msu* (*init-clss S*)) − *length* (*trail S*)
+ (*if conflicting S = None then 1 else 0*)) *S*
> (λS. *card* (*atms-of-msu* (*init-clss S*)) − *length* (*trail S*)
+ (*if conflicting S = None then 1 else 0*)) *T*
using *assms*
proof −
have *length* (*trail T*) ≤ *card* (*atms-of-msu* (*init-clss T*))
apply (*rule length-model-le-vars*)
using $cdcl_W$*-no-strange-atm-inv alien M-lev* **apply** (*meson* $cdcl_W$ $cdcl_W$*.simps* $cdcl_W$*-cp.cases*)
using *M-lev* $cdcl_W$ $cdcl_W$*-cp-consistent-inv* $cdcl_W$*-M-level-inv-def* **apply** *blast*
using $cdcl_W$ **by** (*auto simp*: $cdcl_W$*-cp.simps*)
with *assms*
show *?thesis* **by** *induction* (*auto split*: *split-if-asm*)+
qed

lemma $cdcl_W$*-cp-wf*: *wf* {(*b*,*a*). ($cdcl_W$*-M-level-inv a* ∧ *no-strange-atm a*)
∧ $cdcl_W$*-cp a b*}
apply (*rule wf-wf-if-measure'*[*of less-than - -*
(λS. *card* (*atms-of-msu* (*init-clss S*)) − *length* (*trail S*)
+ (*if conflicting S = None then 1 else 0*))])
apply *simp*
using $cdcl_W$*-cp-decreasing-measure* **unfolding** *less-than-iff* **by** *blast*

lemma *rtranclp-*$cdcl_W$*-all-struct-inv-*$cdcl_W$*-cp-iff-rtranclp-*$cdcl_W$*-cp*:
assumes
lev: $cdcl_W$*-M-level-inv S* **and**
alien: *no-strange-atm S*
shows ($\lambda a\ b$. ($cdcl_W$*-M-level-inv a* ∧ *no-strange-atm a*) ∧ $cdcl_W$*-cp a b*)** *S T*
⟷ $cdcl_W$*-cp*** *S T*
(**is** *?I S T* ⟷ *?C S T*)
proof
assume
?I S T
then show *?C S T* **by** *induction auto*
next
assume
?C S T
then show *?I S T*
proof *induction*
case *base*
then show *?case* **by** *simp*
next
case (*step T U*) **note** *st* = *this*(*1*) **and** *cp* = *this*(*2*) **and** *IH* = *this*(*3*)
have $cdcl_W{}^{**}$ *S T*
by (*metis rtranclp-unfold* $cdcl_W$*-cp-conflicting-not-empty cp st*
rtranclp-propagate-is-rtranclp-$cdcl_W$ *tranclp-*$cdcl_W$*-cp-propagate-with-conflict-or-not*)
then have

$cdcl_W$*-M-level-inv T* **and**
no-strange-atm T
using ‹$cdcl_W^{**}$ *S T*› **apply** (*simp add*: *assms*(*1*) *rtranclp-*$cdcl_W$*-consistent-inv*)
using ‹$cdcl_W^{**}$ *S T*› *alien rtranclp-*$cdcl_W$*-no-strange-atm-inv lev* **by** *blast*
then have (λa *b*. ($cdcl_W$*-M-level-inv a* ∧ *no-strange-atm a*)
∧ $cdcl_W$*-cp a b*)** *T U*
using *cp* **by** *auto*
then show *?case* **using** *IH* **by** *auto*
qed
qed

lemma $cdcl_W$*-cp-normalized-element*:
assumes
lev: $cdcl_W$*-M-level-inv S* **and**
no-strange-atm S
obtains *T* **where** *full* $cdcl_W$*-cp S T*
proof −
let *?inv* = λa. ($cdcl_W$*-M-level-inv a* ∧ *no-strange-atm a*)
obtain *T* **where** *T*: *full* (λa *b*. *?inv a* ∧ $cdcl_W$*-cp a b*) *S T*
using $cdcl_W$*-cp-wf wf-exists-normal-form*[*of* λa *b*. *?inv a* ∧ $cdcl_W$*-cp a b*]
unfolding *full-def* **by** *blast*
then have $cdcl_W$*-cp*** *S T*
using *rtranclp-*$cdcl_W$*-all-struct-inv-*$cdcl_W$*-cp-iff-rtranclp-*$cdcl_W$*-cp assms* **unfolding** *full-def*
by *blast*
moreover
then have $cdcl_W^{**}$ *S T*
using *rtranclp-*$cdcl_W$*-cp-rtranclp-*$cdcl_W$ **by** *blast*
then have
$cdcl_W$*-M-level-inv T* **and**
no-strange-atm T
using ‹$cdcl_W^{**}$ *S T*› **apply** (*simp add*: *assms*(*1*) *rtranclp-*$cdcl_W$*-consistent-inv*)
using ‹$cdcl_W^{**}$ *S T*› *assms*(*2*) *rtranclp-*$cdcl_W$*-no-strange-atm-inv lev* **by** *blast*
then have *no-step* $cdcl_W$*-cp T*
using *T* **unfolding** *full-def* **by** *auto*
ultimately show *thesis* **using** *that* **unfolding** *full-def* **by** *blast*
qed

lemma *in-atms-of-implies-atm-of-on-atms-of-ms*:
C + {#*L*#} ∈# *A* ⟹ *x* ∈ *atms-of C* ⟹ *x* ∈ *atms-of-msu A*
by (*metis add.commute atm-iff-pos-or-neg-lit atms-of-atms-of-ms-mono contra-subsetD*
mem-set-mset-iff multi-member-skip)

lemma *propagate-no-stange-atm*:
assumes
propagate S S′ **and**
no-strange-atm S
shows *no-strange-atm S′*
using *assms* **by** *induction*
(*auto simp add*: *no-strange-atm-def clauses-def in-plus-implies-atm-of-on-atms-of-ms*
in-atms-of-implies-atm-of-on-atms-of-ms)

lemma *always-exists-full-*$cdcl_W$*-cp-step*:
assumes *no-strange-atm S*
shows ∃ *S″*. *full* $cdcl_W$*-cp S S″*
using *assms*

proof (*induct card* (*atms-of-msu* (*init-clss S*) − *atm-of* '*lits-of* (*trail S*)) *arbitrary*: *S*)
  case *0* **note** *card* = *this*(*1*) **and** *alien* = *this*(*2*)
  then have *atm*: *atms-of-msu* (*init-clss S*) = *atm-of* ' *lits-of* (*trail S*)
    unfolding *no-strange-atm-def* **by** *auto*
  { assume *a*: $\exists S'$. *conflict S S'*
    then obtain S' **where** S': *conflict S S'* **by** *metis*
    then have $\forall S''$. ¬$cdcl_W$*-cp S' S''* **by** *auto*
    then have *?case* **using** *a S'* $cdcl_W$*-cp.conflict'* **unfolding** *full-def* **by** *blast*
  }
  moreover {
    assume *a*: $\exists S'$. *propagate S S'*
    then obtain S' **where** *propagate S S'* **by** *blast*
    then obtain *M N U k C L* **where** *S*: *state S* = (*M*, *N*, *U*, *k*, *None*)
    and S': *state S'* = (*Propagated L* ((*C* + {#*L*#})) # *M*, *N*, *U*, *k*, *None*)
    and *C* + {#*L*#} ∈# *clauses S*
    and *M* ⊨*as CNot C*
    and *undefined-lit M L*
    using *propagate* **by** *auto*
    have *atms-of-msu U* ⊆ *atms-of-msu N* **using** *alien S* **unfolding** *no-strange-atm-def* **by** *auto*
    then have *atm-of L* ∈ *atms-of-msu* (*init-clss S*)
      using ‹*C* + {#*L*#} ∈# *clauses S*› *S* **unfolding** *atms-of-ms-def clauses-def* **by** *force+*
    then have *False* **using** ‹*undefined-lit M L*› *S* **unfolding** *atm* **unfolding** *lits-of-def*
      by (*auto simp add*: *defined-lit-map*)
  }
  ultimately show *?case* **by** (*metis* $cdcl_W$*-cp.cases full-def rtranclp.rtrancl-refl*)
next
  case (*Suc n*) **note** *IH* = *this*(*1*) **and** *card* = *this*(*2*) **and** *alien* = *this*(*3*)
  { assume *a*: $\exists S'$. *conflict S S'*
    then obtain S' **where** S': *conflict S S'* **by** *metis*
    then have $\forall S''$. ¬$cdcl_W$*-cp S' S''* **by** *auto*
    then have *?case* **unfolding** *full-def Ex-def* **using** S' $cdcl_W$*-cp.conflict'* **by** *blast*
  }
  moreover {
    assume *a*: $\exists S'$. *propagate S S'*
    then obtain S' **where** *propagate*: *propagate S S'* **by** *blast*
    then obtain *M N U k C L* **where**
      S: *state S* = (*M*, *N*, *U*, *k*, *None*) **and**
      S': *state S'* = (*Propagated L* ((*C* + {#*L*#})) # *M*, *N*, *U*, *k*, *None*) **and**
      C + {#*L*#} ∈# *clauses S* **and**
      M ⊨*as CNot C* **and**
      undefined-lit M L
      by *fastforce*
    then have *atm-of L* ∉ *atm-of* ' *lits-of M*
      unfolding *lits-of-def* **by** (*auto simp add*: *defined-lit-map*)
    moreover
      have *no-strange-atm S'* **using** *alien propagate propagate-no-stange-atm* **by** *blast*
      then have *atm-of L* ∈ *atms-of-msu N* **using** S' **unfolding** *no-strange-atm-def* **by** *auto*
      then have $\bigwedge$*A*. {*atm-of L*} ⊆ *atms-of-msu N* − *A* ∨ *atm-of L* ∈ *A* **by** *force*
    moreover have *Suc n* − *card* {*atm-of L*} = *n* **by** *simp*
    moreover have *card* (*atms-of-msu N* − *atm-of* ' *lits-of M*) = *Suc n*
     using *card S S'* **by** *simp*
    ultimately
      have *card* (*atms-of-msu N* − *atm-of* ' *insert L* (*lits-of M*)) = *n*
        by (*metis* (*no-types*) *Diff-insert card-Diff-subset finite.emptyI finite.insertI image-insert*)
      then have *n* = *card* (*atms-of-msu* (*init-clss S'*) − *atm-of* ' *lits-of* (*trail S'*))

using *card S S'* **by** *simp*
then have *a1*: *Ex* (*full* $cdcl_W$*-cp S'*) **using** *IH* ‹*no-strange-atm S'*› **by** *blast*
have *?case*
proof −
obtain *S''* :: *'st* **where**
ff1: $cdcl_W$*-cp*** *S' S''* ∧ *no-step* $cdcl_W$*-cp S''*
using *a1* **unfolding** *full-def* **by** *blast*
have $cdcl_W$*-cp*** *S S''*
using *ff1* $cdcl_W$*-cp.intros*(*2*)[*OF propagate*]
by (*metis* (*no-types*) *converse-rtranclp-into-rtranclp*)
then have ∃ *S''*. $cdcl_W$*-cp*** *S S''* ∧ (∀ *S'''*. ¬ $cdcl_W$*-cp S'' S'''*)
using *ff1* **by** *blast*
then show *?thesis* **unfolding** *full-def*
by *meson*
qed
}
ultimately show *?case* **unfolding** *full-def* **by** (*metis* $cdcl_W$*-cp.cases rtranclp.rtrancl-refl*)
qed

5.6.3 Literal of highest level in conflicting clauses

One important property of the *local.*$cdcl_W$ with strategy is that, whenever a conflict takes place, there is at least a literal of level k involved (except if we have derived the false clause). The reason is that we apply conflicts before a decision is taken.

abbreviation *no-clause-is-false* :: *'st* ⇒ *bool* **where**
no-clause-is-false ≡
λ*S*. (*conflicting S* = *None* ⟶ (∀ *D* ∈# *clauses S*. ¬*trail S* ⊨*as CNot D*))

abbreviation *conflict-is-false-with-level* :: *'st* ⇒ *bool* **where**
conflict-is-false-with-level S ≡ ∀ *D*. *conflicting S* = *Some D* ⟶ *D* ≠ {#}
⟶ (∃ *L* ∈# *D*. *get-level* (*trail S*) *L* = *backtrack-lvl S*)

lemma *not-conflict-not-any-negated-init-clss*:
assumes ∀ *S'*. ¬*conflict S S'*
shows *no-clause-is-false S*
using *assms state-eq-ref* **by** *blast*

lemma *full-*$cdcl_W$*-cp-not-any-negated-init-clss*:
assumes *full* $cdcl_W$*-cp S S'*
shows *no-clause-is-false S'*
using *assms not-conflict-not-any-negated-init-clss* **unfolding** *full-def* **by** *blast*

lemma *full1-*$cdcl_W$*-cp-not-any-negated-init-clss*:
assumes *full1* $cdcl_W$*-cp S S'*
shows *no-clause-is-false S'*
using *assms not-conflict-not-any-negated-init-clss* **unfolding** *full1-def* **by** *blast*

lemma $cdcl_W$*-stgy-not-non-negated-init-clss*:
assumes $cdcl_W$*-stgy S S'*
shows *no-clause-is-false S'*
using *assms* **apply** (*induct rule*: $cdcl_W$*-stgy.induct*)
using *full1-*$cdcl_W$*-cp-not-any-negated-init-clss full-*$cdcl_W$*-cp-not-any-negated-init-clss* **by** *metis+*

lemma *rtranclp-*$cdcl_W$*-stgy-not-non-negated-init-clss*:
assumes $cdcl_W$*-stgy*** *S S'* **and** *no-clause-is-false S*

shows *no-clause-is-false* S'
using *assms* **by** (*induct rule*: *rtranclp-induct*) (*auto simp*: $cdcl_W$*-stgy-not-non-negated-init-clss*)

lemma $cdcl_W$*-stgy-conflict-ex-lit-of-max-level*:
assumes $cdcl_W$*-cp* S S'
and *no-clause-is-false* S
and $cdcl_W$*-M-level-inv* S
shows *conflict-is-false-with-level* S'
using *assms*
proof (*induct rule*: $cdcl_W$*-cp.induct*)
case *conflict'*
then show *?case* **by** *auto*
next
case *propagate'*
then show *?case* **by** *auto*
qed

lemma *no-chained-conflict*:
assumes *conflict* S S'
and *conflict* S' S''
shows *False*
using *assms* **by** *fastforce*

lemma *rtranclp-*$cdcl_W$*-cp-propa-or-propa-confl*:
assumes $cdcl_W$*-cp*** S U
shows *propagate*** S U $\vee$ ($\exists$ T. *propagate*** S T $\wedge$ *conflict* T U)
using *assms*
proof *induction*
case *base*
then show *?case* **by** *auto*
next
case (*step* U V) **note** SU = *this*(*1*) **and** UV = *this*(*2*) **and** IH = *this*(*3*)
consider (*confl*) T **where** *propagate*** S T **and** *conflict* T U
| (*propa*) *propagate*** S U **using** IH **by** *auto*
then show *?case*
proof *cases*
case *confl*
then have *False* **using** UV **by** *auto*
then show *?thesis* **by** *fast*
next
case *propa*
also have *conflict* U V $\vee$ *propagate* U V **using** UV **by** (*auto simp add*: $cdcl_W$*-cp.simps*)
ultimately show *?thesis* **by** *force*
qed
qed

lemma *rtranclp-*$cdcl_W$*-co-conflict-ex-lit-of-max-level*:
assumes *full*: *full* $cdcl_W$*-cp* S U
and *cls-f*: *no-clause-is-false* S
and *conflict-is-false-with-level* S
and *lev*: $cdcl_W$*-M-level-inv* S
shows *conflict-is-false-with-level* U
proof (*intro allI impI*)
fix D
assume *confl*: *conflicting* U = *Some* D **and**

D: $D \neq \{\#\}$

consider (CT) *conflicting* $S = None \mid (SD)\ D'$ **where** *conflicting* $S = Some\ D'$

by (*cases conflicting S*) *auto*

then show $\exists L\in\#D.$ *get-level* (*trail U*) $L =$ *backtrack-lvl U*

proof *cases*

case *SD*

then have $S = U$

by (*metis* (*no-types*) *assms*(*1*) $cdcl_W$*-cp-conflicting-not-empty full-def rtranclpD tranclpD*)

then show *?thesis* **using** *assms*(*3*) *confl D* **by** *blast*−

next

case *CT*

have *init-clss U* = *init-clss S* **and** *learned-clss U* = *learned-clss S*

using *assms*(*1*) **unfolding** *full-def*

apply (*metis* (*no-types*) *rtranclpD tranclp-*$cdcl_W$*-cp-no-more-init-clss*)

by (*metis* (*mono-tags*, *lifting*) *assms*(*1*) *full-def rtranclp-*$cdcl_W$*-cp-learned-clause-inv*)

obtain *T* **where** $propagate^{**}$ *S T* **and** *TU*: *conflict T U*

proof −

have *f5*: $U \neq S$

using *confl CT* **by** *force*

then have $cdcl_W$*-cp*$^{++}$ *S U*

by (*metis full full-def rtranclpD*)

have $\bigwedge p\ pa.\ \neg$ *propagate p pa* $\vee$ *conflicting pa* =

(*None*::′*v literal multiset option*)

by *auto*

then show *?thesis*

using *f5 that tranclp-*$cdcl_W$*-cp-propagate-with-conflict-or-not*[*OF* ‹$cdcl_W$*-cp*$^{++}$ *S U*›]

full confl CT **unfolding** *full-def* **by** *auto*

qed

have *init-clss T* = *init-clss S* **and** *learned-clss T* = *learned-clss S*

using *TU* ‹*init-clss U* = *init-clss S*› ‹*learned-clss U* = *learned-clss S*› **by** *auto*

then have $D \in\#$ *clauses S*

using *TU confl* **by** (*fastforce simp*: *clauses-def*)

then have ¬ *trail S* ⊨*as CNot D*

using *cls-f CT* **by** *simp*

moreover

obtain *M* **where** *tr-U*: *trail U* = *M* @ *trail S* **and** *nm*: $\forall m\in$*set M*. ¬*is-marked m*

by (*metis* (*mono-tags*, *lifting*) *assms*(*1*) *full-def rtranclp-*$cdcl_W$*-cp-dropWhile-trail*)

have *trail U* ⊨*as CNot D*

using *TU confl* **by** *auto*

ultimately obtain *L* **where** $L \in\# D$ **and** $-L \in$ *lits-of M*

unfolding *tr-U CNot-def true-annots-def Ball-def true-annot-def true-cls-def* **by** *auto*

moreover have *inv-U*: $cdcl_W$*-M-level-inv U*

by (*metis* $cdcl_W$*-stgy.conflict*′ $cdcl_W$*-stgy-consistent-inv full full-unfold lev*)

moreover

have *backtrack-lvl U* = *backtrack-lvl S*

using *full* **unfolding** *full-def* **by** (*auto dest*: *rtranclp-*$cdcl_W$*-cp-backtrack-lvl*)

moreover

have *no-dup* (*trail U*)

using *inv-U* **unfolding** $cdcl_W$*-M-level-inv-def* **by** *auto*

{ **fix** *x* :: (′*v*, *nat*, ′*v literal multiset*) *ann-literal* **and**

xb :: (′*v*, *nat*, ′*v literal multiset*) *ann-literal*

assume *a1*: *atm-of L* = *atm-of* (*lit-of xb*)

moreover assume *a2*: − *L* = *lit-of x*

moreover assume *a3*: $(\lambda l.\ atm\text{-}of\ (lit\text{-}of\ l))$ ‘ *set M*
$\cap$ $(\lambda l.\ atm\text{-}of\ (lit\text{-}of\ l))$ ‘ *set* (*trail S*) = {}
moreover assume *a4*: $x \in$ *set M*
moreover assume *a5*: $xb \in$ *set* (*trail S*)
moreover have *atm-of* (− *L*) = *atm-of L*
by *auto*
ultimately have *False*
by *auto*
}
then have *LS*: *atm-of L* $\notin$ *atm-of* ‘ *lits-of* (*trail S*)
using ⟨−*L* $\in$ *lits-of M*⟩ ⟨*no-dup* (*trail U*)⟩ **unfolding** *tr-U lits-of-def* **by** *auto*
ultimately have *get-level* (*trail U*) *L* = *backtrack-lvl U*
proof (*cases get-all-levels-of-marked* (*trail S*) $\neq$ [], *goal-cases*)
case *2* **note** *LD* = *this*(*1*) **and** *LM* = *this*(*2*) **and** *inv-U* = *this*(*3*) **and** *US* = *this*(*4*) **and** *LS* = *this*(*5*) **and** *ne* = *this*(*6*)
have *backtrack-lvl S* = *0*
using *lev ne* **unfolding** $cdcl_W$*-M-level-inv-def* **by** *auto*
moreover have *get-rev-level* (*rev M*) *0 L* = *0*
using *nm* **by** *auto*
ultimately show *?thesis* **using** *LS ne US* **unfolding** *tr-U*
by (*simp add*: *get-all-levels-of-marked-nil-iff-not-is-marked lits-of-def*)
next
case *1* **note** *LD* = *this*(*1*) **and** *LM* = *this*(*2*) **and** *inv-U* = *this*(*3*) **and** *US* = *this*(*4*) **and** *LS* = *this*(*5*) **and** *ne* = *this*(*6*)

have *hd* (*get-all-levels-of-marked* (*trail S*)) = *backtrack-lvl S*
using *ne lev* **unfolding** $cdcl_W$*-M-level-inv-def*
by (*cases get-all-levels-of-marked* (*trail S*)) *auto*
moreover have *atm-of L* $\in$ *atm-of* ‘ *lits-of M*
using ⟨−*L* $\in$ *lits-of M*⟩ **by** (*simp add*: *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set lits-of-def*)
ultimately show *?thesis*
using *nm ne* **unfolding** *tr-U*
using *get-level-skip-beginning-hd-get-all-levels-of-marked*[*OF LS*, *of M*]
get-level-skip-in-all-not-marked[*of rev M L backtrack-lvl S*]
unfolding *lits-of-def US*
by *auto*
qed
then show $\exists$ *L*$\in$#*D*. *get-level* (*trail U*) *L* = *backtrack-lvl U*
using ⟨*L* $\in$# *D*⟩ **by** *blast*
qed
qed

5.6.4 Literal of highest level in marked literals

definition *mark-is-false-with-level* :: *'st* $\Rightarrow$ *bool* **where**
mark-is-false-with-level S' $\equiv$
$\forall$ *D M1 M2 L*. *M1* @ *Propagated L D* # *M2* = *trail S'* $\longrightarrow$ *D* − {#*L*#} $\neq$ {#}
$\longrightarrow$ ($\exists$ *L*. *L* $\in$# *D* $\wedge$ *get-level* (*trail S'*) *L* = *get-maximum-possible-level M1*)

definition *no-more-propagation-to-do*:: *'st* $\Rightarrow$ *bool* **where**
no-more-propagation-to-do S $\equiv$
$\forall$ *D M M' L*. *D* + {#*L*#} $\in$# *clauses S* $\longrightarrow$ *trail S* = *M'* @ *M* $\longrightarrow$ *M* $\models$*as CNot D*
$\longrightarrow$ *undefined-lit M L* $\longrightarrow$ *get-maximum-possible-level M* < *backtrack-lvl S*
$\longrightarrow$ ($\exists$ *L*. *L* $\in$# *D* $\wedge$ *get-level* (*trail S*) *L* = *get-maximum-possible-level M*)

```
lemma propagate-no-more-propagation-to-do:
  assumes propagate: propagate S S'
  and H: no-more-propagation-to-do S
  and M: cdcl_W-M-level-inv S
  shows no-more-propagation-to-do S'
  using assms
proof -
  obtain M N U k C L where
    S: state S = (M, N, U, k, None) and
    S': state S' = (Propagated L ( (C + {#L#})) # M, N, U, k, None) and
    C + {#L#} ∈# clauses S and
    M ⊨as CNot C and
    undefined-lit M L
    using propagate by auto
  let ?M' = Propagated L ( (C + {#L#})) # M
  show ?thesis unfolding no-more-propagation-to-do-def
    proof (intro allI impI)
      fix D M1 M2 L'
      assume D-L: D + {#L'#} ∈# clauses S'
      and trail S' = M2 @ M1
      and get-max: get-maximum-possible-level M1 < backtrack-lvl S'
      and M1 ⊨as CNot D
      and undef: undefined-lit M1 L'
      have tl M2 @ M1 = trail S ∨ (M2 = [] ∧ M1 = Propagated L ( (C + {#L#})) # M)
        using ‹trail S' = M2 @ M1› S' S by (cases M2) auto
      moreover {
        assume tl M2 @ M1 = trail S
        moreover have D + {#L'#} ∈# clauses S using D-L S S' unfolding clauses-def by auto
        moreover have get-maximum-possible-level M1 < backtrack-lvl S
          using get-max S S' by auto
        ultimately obtain L' where L' ∈# D and
          get-level (trail S) L' = get-maximum-possible-level M1
          using H ‹M1 ⊨as CNot D› undef unfolding no-more-propagation-to-do-def by metis
        moreover
          { have cdcl_W-M-level-inv S'
              using cdcl_W-consistent-inv[OF - M] cdcl_W.propagate[OF propagate] by blast
            then have no-dup ?M' using S' unfolding cdcl_W-M-level-inv-def by auto
            moreover
              have atm-of L' ∈ atm-of ` (lits-of M1)
                using ‹L' ∈# D› ‹M1 ⊨as CNot D› by (metis atm-of-uminus image-eqI
                  in-CNot-implies-uminus(2))
              then have atm-of L' ∈ atm-of ` (lits-of M)
                using ‹tl M2 @ M1 = trail S› S by auto
            ultimately have atm-of L ≠ atm-of L' unfolding lits-of-def by auto
        }
        ultimately have ∃ L' ∈# D. get-level (trail S') L' = get-maximum-possible-level M1
          using S S' by auto
      }
      moreover {
        assume M2 = [] and M1: M1 = Propagated L ( (C + {#L#})) # M
        have cdcl_W-M-level-inv S'
          using cdcl_W-consistent-inv[OF - M] cdcl_W.propagate[OF propagate] by blast
        then have get-all-levels-of-marked (trail S') = rev ([Suc 0..<(Suc 0+k)])
          using S' unfolding cdcl_W-M-level-inv-def by auto
        then have get-maximum-possible-level M1 = backtrack-lvl S'
```

using *get-maximum-possible-level-max-get-all-levels-of-marked*[*of M1*] *S′ M1*
by (*auto intro*: *Max-eqI*)
then have *False* **using** *get-max* **by** *auto*
}
ultimately show $\exists L.\ L \in\# D \wedge$ *get-level* (*trail S′*) *L* = *get-maximum-possible-level M1* **by** *fast*
qed
qed

lemma *conflict-no-more-propagation-to-do*:
assumes *conflict*: *conflict S S′*
and *H*: *no-more-propagation-to-do S*
and *M*: $cdcl_W$*-M-level-inv S*
shows *no-more-propagation-to-do S′*
using *assms* **unfolding** *no-more-propagation-to-do-def conflict.simps* **by** *force*

lemma $cdcl_W$*-cp-no-more-propagation-to-do*:
assumes *conflict*: $cdcl_W$*-cp S S′*
and *H*: *no-more-propagation-to-do S*
and *M*: $cdcl_W$*-M-level-inv S*
shows *no-more-propagation-to-do S′*
using *assms*
proof (*induct rule*: $cdcl_W$*-cp.induct*)
case (*conflict′ S S′*)
then show *?case* **using** *conflict-no-more-propagation-to-do*[*of S S′*] **by** *blast*
next
case (*propagate′ S S′*) **note** *S* = *this*
show *1*: *no-more-propagation-to-do S′*
using *propagate-no-more-propagation-to-do*[*of S S′*] *S* **by** *blast*
qed

lemma $cdcl_W$*-then-exists-*$cdcl_W$*-stgy-step*:
assumes
o: $cdcl_W$*-o S S′* **and**
alien: *no-strange-atm S* **and**
lev: $cdcl_W$*-M-level-inv S*
shows $\exists S'.$ $cdcl_W$*-stgy S S′*
proof −
obtain *S″* **where** *full* $cdcl_W$*-cp S′ S″*
using *always-exists-full-*$cdcl_W$*-cp-step alien* $cdcl_W$*-no-strange-atm-inv* $cdcl_W$*-o-no-more-init-clss*
o other lev **by** (*meson* $cdcl_W$*-consistent-inv*)
then show *?thesis*
using *assms* **by** (*metis always-exists-full-*$cdcl_W$*-cp-step* $cdcl_W$*-stgy.conflict′ full-unfold other′*)
qed

lemma *backtrack-no-decomp*:
assumes *S*: *state S* = (*M*, *N*, *U*, *k*, *Some* (*D* + {#*L*#}))
and *L*: *get-level M L* = *k*
and *D*: *get-maximum-level M D* < *k*
and *M-L*: $cdcl_W$*-M-level-inv S*
shows $\exists S'.$ $cdcl_W$*-o S S′*
proof −
have *L-D*: *get-level M L* = *get-maximum-level M* (*D* + {#*L*#})
using *L D* **by** (*simp add*: *get-maximum-level-plus*)
let *?i* = *get-maximum-level M D*
obtain *K M1 M2* **where** *K*: (*Marked K* (*?i* + *1*) # *M1*, *M2*) ∈ *set* (*get-all-marked-decomposition*

M)
  using *backtrack-ex-decomp*[*OF M-L*, *of ?i*] *D S* **by** *auto*
 show *?thesis* **using** *backtrack-rule*[*OF S K L L-D*] **by** (*meson bj* $cdcl_W$*-bj.simps state-eq-ref*)
qed

lemma $cdcl_W$*-stgy-final-state-conclusive*:
 assumes *termi*: $\forall S'. \neg cdcl_W\text{-}stgy\ S\ S'$
 and *decomp*: *all-decomposition-implies-m* (*init-clss S*) (*get-all-marked-decomposition* (*trail S*))
 and *learned*: $cdcl_W$*-learned-clause S*
 and *level-inv*: $cdcl_W$*-M-level-inv S*
 and *alien*: *no-strange-atm S*
 and *no-dup*: *distinct-*$cdcl_W$*-state S*
 and *confl*: $cdcl_W$*-conflicting S*
 and *confl-k*: *conflict-is-false-with-level S*
 shows (*conflicting S* = *Some* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss S*)))
    ∨ (*conflicting S* = *None* ∧ *trail S* ⊨*as set-mset* (*init-clss S*))
proof −
 let *?M* = *trail S*
 let *?N* = *init-clss S*
 let *?k* = *backtrack-lvl S*
 let *?U* = *learned-clss S*
 have *conflicting S* = *Some* {#}
   ∨ *conflicting S* = *None*
   ∨ (∃ *D L*. *conflicting S* = *Some* (*D* + {#*L*#}))
  apply (*cases conflicting S*, *auto*)
  by (*rename-tac C*, *case-tac C*, *auto*)
 moreover {
  assume *conflicting S* = *Some* {#}
  then have *unsatisfiable* (*set-mset* (*init-clss S*))
   using *assms*(*3*) **unfolding** $cdcl_W$*-learned-clause-def true-clss-cls-def*
   by (*metis* (*no-types*, *lifting*) *Un-insert-right atms-of-empty satisfiable-def*
    sup-bot.right-neutral total-over-m-insert total-over-set-empty true-cls-empty)
 }
 moreover {
  assume *conflicting S* = *None*
  { **assume** ¬*?M* ⊨*asm ?N*
   have *atm-of* ' (*lits-of ?M*) = *atms-of-msu ?N* (**is** *?A* = *?B*)
    proof
     show *?A* ⊆ *?B* **using** *alien* **unfolding** *no-strange-atm-def* **by** *auto*
     show *?B* ⊆ *?A*
      proof (*rule ccontr*)
       assume ¬*?B* ⊆ *?A*
       then obtain *l* **where** *l* ∈ *?B* **and** *l* ∉ *?A* **by** *auto*
       then have *undefined-lit ?M* (*Pos l*)
        using ‹*l* ∉ *?A*› **unfolding** *lits-of-def* **by** (*auto simp add*: *defined-lit-map*)
       then have $\exists S'. cdcl_W\text{-}o\ S\ S'$
        using $cdcl_W$*-o.decide decide.intros* ‹*l* ∈ *?B*› *no-strange-atm-def*
        by (*metis* ‹*conflicting S* = *None*› *literal.sel*(*1*) *state-eq-def*)
       then show *False*
        using *termi* $cdcl_W$*-then-exists-*$cdcl_W$*-stgy-step*[*OF - alien*] *level-inv* **by** *blast*
      qed
     qed
    obtain *D* **where** ¬ *?M* ⊨*a D* **and** *D* ∈# *?N*
     using ‹¬*?M* ⊨*asm ?N*› **unfolding** *lits-of-def true-annots-def Ball-def* **by** *auto*
    have *atms-of D* ⊆ *atm-of* ' (*lits-of ?M*)

```
        using ‹D ∈# ?N› unfolding ‹atm-of ‘ (lits-of ?M) = atms-of-msu ?N› atms-of-ms-def
        by (auto simp add: atms-of-def)
      then have a1: atm-of ‘ set-mset D ⊆ atm-of ‘ lits-of (trail S)
        by (auto simp add: atms-of-def lits-of-def)
      have total-over-m (lits-of ?M) {D}
        using ‹atms-of D ⊆ atm-of ‘ (lits-of ?M)› atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set
        by (fastforce simp: total-over-set-def)
      then have ?M ⊨as CNot D
        using total-not-true-cls-true-clss-CNot ‹¬ trail S ⊨a D› true-annot-def
        true-annots-true-cls by fastforce
      then have False
        proof −
          obtain S' where
            f2: full cdcl_W-cp S S'
            by (meson alien always-exists-full-cdcl_W-cp-step level-inv)
          then have S' = S
            using cdcl_W-stgy.conflict'[of S] by (metis (no-types) full-unfold termi)
          then show ?thesis
            using f2 ‹D ∈# init-clss S› ‹conflicting S = None› ‹trail S ⊨as CNot D›
            clauses-def full-cdcl_W-cp-not-any-negated-init-clss by auto
        qed
    }
    then have ?M ⊨asm ?N by blast
  }
  moreover {
    assume ∃D L. conflicting S = Some (D + {#L#})
    then obtain D L where LD: conflicting S = Some (D + {#L#}) and lev-L: get-level ?M L = ?k
      by (metis (mono-tags) bex-msetE confl-k insert-DiffM2 multi-self-add-other-not-self
        union-eq-empty)
    let ?D = D + {#L#}
    have ?D ≠ {#} by auto
    have ?M ⊨as CNot ?D using confl LD unfolding cdcl_W-conflicting-def by auto
    then have ?M ≠ [] unfolding true-annots-def Ball-def true-annot-def true-cls-def by force
    { have M: ?M = hd ?M # tl ?M using ‹?M ≠ []› list.collapse by fastforce
      assume marked: is-marked (hd ?M)
      then obtain k' where k': k' + 1 = ?k
        using level-inv M unfolding cdcl_W-M-level-inv-def
        by (cases hd (trail S); cases trail S) auto
      obtain L' l' where L': hd ?M = Marked L' l' using marked by (cases hd ?M) auto
      have marked-hd-tl: get-all-levels-of-marked (hd (trail S) # tl (trail S))
        = rev [1..<1 + length (get-all-levels-of-marked ?M)]
        using level-inv lev-L M unfolding cdcl_W-M-level-inv-def M[symmetric]
        by blast
      then have l'-tl: l' # get-all-levels-of-marked (tl ?M)
        = rev [1..<1 + length (get-all-levels-of-marked ?M)] unfolding L' by simp
      moreover have ... = length (get-all-levels-of-marked ?M)
        # rev [1..<length (get-all-levels-of-marked ?M)]
        using M Suc-le-mono calculation by (fastforce simp add: upt.simps(2))
      finally have
        l' = ?k and
        g-r: get-all-levels-of-marked (tl (trail S))
          = rev [1..<length (get-all-levels-of-marked (trail S))]
        using level-inv lev-L M unfolding cdcl_W-M-level-inv-def by auto
      have *: ⋀list. no-dup list ⟹
           − L ∈ lits-of list ⟹ atm-of L ∈ atm-of ‘ lits-of list
```

by (*metis atm-of-uminus imageI*)
have $L' = -L$
proof (*rule ccontr*)
assume ¬ *?thesis*
moreover have $-L \in$ *lits-of ?M* **using** *confl LD* **unfolding** *cdcl*$_W$*-conflicting-def* **by** *auto*
ultimately have *get-level* (*hd* (*trail S*) # *tl* (*trail S*)) *L* = *get-level* (*tl ?M*) *L*
using *cdcl*$_W$*-M-level-inv-decomp*(*1*)[*OF level-inv*] **unfolding** L' *consistent-interp-def*
by (*metis* (*no-types*, *lifting*) L' *M atm-of-eq-atm-of get-level-skip-beginning insert-iff*
lits-of-cons ann-literal.sel(*1*))

moreover
have *length* (*get-all-levels-of-marked* (*trail S*)) = *?k*
using *level-inv* **unfolding** *cdcl*$_W$*-M-level-inv-def* **by** *auto*
then have *Max* (*set* (*0#get-all-levels-of-marked* (*tl* (*trail S*)))) = *?k* − *1*
unfolding *g-r* **by** (*auto simp add*: *Max-n-upt*)
then have *get-level* (*tl ?M*) *L* < *?k*
using *get-maximum-possible-level-ge-get-level*[*of tl ?M L*]
by (*metis One-nat-def add.right-neutral add-Suc-right diff-add-inverse2*
get-maximum-possible-level-max-get-all-levels-of-marked k' *le-imp-less-Suc*
list.simps(*15*))
finally show *False* **using** *lev-L M* **by** *auto*
qed
have *L*: *hd ?M* = *Marked* (−*L*) *?k* **using** ‹l' = *?k*› ‹$L' = -L$› L' **by** *auto*

have *g-a-l*: *get-all-levels-of-marked ?M* = *rev* [*1*..<*1* + *?k*]
using *level-inv lev-L M* **unfolding** *cdcl*$_W$*-M-level-inv-def* **by** *auto*
have *g-k*: *get-maximum-level* (*trail S*) *D* ≤ *?k*
using *get-maximum-possible-level-ge-get-maximum-level*[*of ?M*]
get-maximum-possible-level-max-get-all-levels-of-marked[*of ?M*]
by (*auto simp add*: *Max-n-upt g-a-l*)
have *get-maximum-level* (*trail S*) *D* < *?k*
proof (*rule ccontr*)
assume ¬ *?thesis*
then have *get-maximum-level* (*trail S*) *D* = *?k* **using** *M g-k* **unfolding** *L* **by** *auto*
then obtain L' **where** $L' \in\#$ *D* **and** *L-k*: *get-level ?M* L' = *?k*
using *get-maximum-level-exists-lit*[*of ?k ?M D*] **unfolding** k'[*symmetric*] **by** *auto*
have $L \neq L'$ **using** *no-dup* ‹$L' \in\#$ *D*›
unfolding *distinct-cdcl*$_W$*-state-def LD* **by** (*metis add.commute add-eq-self-zero*
count-single count-union less-not-refl3 distinct-mset-def union-single-eq-member)
have $L' = -L$
proof (*rule ccontr*)
assume ¬ *?thesis*
then have *get-level ?M* L' = *get-level* (*tl ?M*) L'
using *M* ‹$L \neq L'$› *get-level-skip-beginning*[*of* L' *hd ?M tl ?M*] **unfolding** *L*
by (*auto simp*: *atm-of-eq-atm-of*)
moreover have . . . < *?k*
proof −
{ **assume** *a1*: *get-level* (*tl* (*trail S*)) L' = *backtrack-lvl S*
assume *a2*: *rev* (*get-all-levels-of-marked* (*tl* (*trail S*))) =
[*Suc 0*..<*backtrack-lvl S*]
have k' + *Suc 0* = *backtrack-lvl S*
using k' **by** *presburger*
then have *False*
using *a2 a1* **by** (*metis* (*no-types*) *Max-n-upt Zero-neq-Suc add-diff-cancel-left'*
add-diff-cancel-right' *diff-is-0-eq*

get-all-levels-of-marked-rev-eq-rev-get-all-levels-of-marked
get-rev-level-less-max-get-all-levels-of-marked list.set(2) set-upt)
}
then show *?thesis*
using *g-r get-rev-level-less-max-get-all-levels-of-marked*[*of rev* (*tl ?M*) *0 L*]
l′-tl calculation[*symmetric*] *g-a-l L-k*
by (*auto simp*: *Max-n-upt* $cdcl_W$*-M-level-inv-def rev-swap*[*symmetric*])
qed
finally show *False* **using** *L-k* **by** *simp*
qed
then have *taut*: *tautology* (D + {#L#})
using ‹L' ∈# D› **by** (*metis add.commute mset-leD mset-le-add-left multi-member-this tautology-minus*)
have *consistent-interp* (*lits-of ?M*)
using *level-inv* **unfolding** $cdcl_W$*-M-level-inv-def* **by** *auto*
then have ¬*?M* ⊨*as CNot ?D*
using *taut* **by** (*metis* (*no-types*) ‹L' = − L› ‹L' ∈# D› *add.commute consistent-interp-def in-CNot-implies-uminus*(*2*) *mset-leD mset-le-add-left multi-member-this*)
moreover have *?M* ⊨*as CNot ?D*
using *confl no-dup LD* **unfolding** $cdcl_W$*-conflicting-def* **by** *auto*
ultimately show *False* **by** *blast*
qed
then have *False*
using *backtrack-no-decomp*[*OF* - ‹*get-level* (*trail S*) *L* = *backtrack-lvl S*› - *level-inv*]
LD alien termi **by** (*metis* $cdcl_W$*-then-exists-*$cdcl_W$*-stgy-step level-inv*)
}
moreover {
assume ¬*is-marked* (*hd ?M*)
then obtain L' C **where** $L'C$: *hd ?M* = *Propagated* L' C **by** (*cases hd ?M*, *auto*)
then have *M*: *?M* = *Propagated* L' C # *tl ?M* **using** ‹*?M* ≠ []› *list.collapse* **by** *fastforce*
then obtain C' **where** C': $C = C'$ + {#L'#}
using *confl* **unfolding** $cdcl_W$*-conflicting-def* **by** (*metis append-Nil diff-single-eq-union*)
{ **assume** −L' ∉# *?D*
then have *False*
using *bj*[*OF* $cdcl_W$*-bj.skip*[*OF skip-rule*[*OF* - ‹−L' ∉# *?D*› ‹*?D* ≠ {#}›, *of S C tl* (*trail S*) -]]]
termi M **by** (*metis LD alien* $cdcl_W$*-then-exists-*$cdcl_W$*-stgy-step state-eq-def level-inv*)
}
moreover {
assume −L' ∈# *?D*
then obtain D' **where** D': *?D* = D' + {#−L'#} **by** (*metis insert-DiffM2*)
have *g-r*: *get-all-levels-of-marked* (*Propagated* L' C # *tl* (*trail S*))
= *rev* [*Suc 0*..<*Suc* (*length* (*get-all-levels-of-marked* (*trail S*)))]
using *level-inv M* **unfolding** $cdcl_W$*-M-level-inv-def* **by** *auto*
have *Max* (*insert 0* (*set* (*get-all-levels-of-marked* (*Propagated* L' C # *tl* (*trail S*))))) = *?k*
using *level-inv M* **unfolding** *g-r* $cdcl_W$*-M-level-inv-def set-rev*
by (*auto simp add*:*Max-n-upt*)
then have *get-maximum-level* (*Propagated* L' C # *tl ?M*) D' ≤ *?k*
using *get-maximum-possible-level-ge-get-maximum-level*[*of Propagated* L' C # *tl ?M*]
unfolding *get-maximum-possible-level-max-get-all-levels-of-marked* **by** *auto*
then have *get-maximum-level* (*Propagated* L' C # *tl ?M*) D' = *?k*
∨ *get-maximum-level* (*Propagated* L' C # *tl ?M*) D' < *?k*
using *le-neq-implies-less* **by** *blast*
moreover {
assume *g-D′-k*: *get-maximum-level* (*Propagated* L' C # *tl ?M*) D' = *?k*

```
          have False
            proof –
              have f1: get-maximum-level (trail S) D′ = backtrack-lvl S
                using M g-D′-k by auto
              have (trail S, init-clss S, learned-clss S, backtrack-lvl S, Some (D + {#L#}))
                = state S
                by (metis (no-types) LD)
              then have cdclW-o S (update-conflicting (Some (D′ #∪ C′)) (tl-trail S))
                using f1 bj[OF cdclW-bj.resolve[OF resolve-rule[of S L′ C′ tl ?M ?N ?U ?k D′]]]
                C′ D′ M by (metis state-eq-def)
              then show ?thesis
                by (meson alien cdclW-then-exists-cdclW-stgy-step termi level-inv)
            qed
        }
        moreover {
          assume get-maximum-level (Propagated L′ C # tl ?M) D′ < ?k
          then have False
            proof –
              assume a1: get-maximum-level (Propagated L′ C # tl (trail S)) D′ < backtrack-lvl S
              obtain mm :: ′v literal multiset and ll :: ′v literal where
                f2: conflicting S = Some (mm + {#ll#})
                    get-level (trail S) ll = backtrack-lvl S
                using LD ‹get-level (trail S) L = backtrack-lvl S› by blast
              then have f3: get-maximum-level (trail S) D′ ≤ get-level (trail S) ll
                using M a1 by force
              have lev-neq: get-level (trail S) ll ≠ get-maximum-level (trail S) D′
                using f2 M calculation(2) by presburger
              have f1: trail S = Propagated L′ C # tl (trail S)
                  conflicting S = Some (D′ + {#− L′#})
                using D′ LD M by force+
              have f2: conflicting S = Some (mm + {#ll#})
                 get-level (trail S) ll = backtrack-lvl S
                using f2 by force+
              have ll = − L′
                by (metis (no-types) D′ LD lev-neq option.inject f2 f3 le-antisym
                  get-maximum-level-ge-get-level insert-noteq-member)
              then show ?thesis
                using f2 f1 M backtrack-no-decomp[of S]
                by (metis (no-types) a1 alien cdclW-then-exists-cdclW-stgy-step level-inv termi)
            qed
        }
        ultimately have False by blast
      }
      ultimately have False by blast
    }
    ultimately have False by blast
  }
  ultimately show ?thesis by blast
qed

lemma cdclW-cp-tranclp-cdclW:
  cdclW-cp S S′ ⟹ cdclW⁺⁺ S S′
  apply (induct rule: cdclW-cp.induct)
  by (meson cdclW.conflict cdclW.propagate tranclp.r-into-trancl tranclp.trancl-into-trancl)+
```

lemma *tranclp-cdcl*$_W$*-cp-tranclp-cdcl*$_W$:
  cdcl$_W$*-cp*$^{++}$ *S S'* $\Longrightarrow$ *cdcl*$_W$$^{++}$ *S S'*
  apply (*induct rule*: *tranclp.induct*)
   apply (*simp add*: *cdcl*$_W$*-cp-tranclp-cdcl*$_W$)
   by (*meson cdcl*$_W$*-cp-tranclp-cdcl*$_W$ *tranclp-trans*)

lemma *cdcl*$_W$*-stgy-tranclp-cdcl*$_W$:
  cdcl$_W$*-stgy S S'* $\Longrightarrow$ *cdcl*$_W$$^{++}$ *S S'*
proof (*induct rule*: *cdcl*$_W$*-stgy.induct*)
 case *conflict'*
 then show *?case*
  unfolding *full1-def* **by** (*simp add*: *tranclp-cdcl*$_W$*-cp-tranclp-cdcl*$_W$)
next
 case (*other' S' S''*)
 then have $S' = S'' \vee$ *cdcl*$_W$*-cp*$^{++}$ *S' S''*
  by (*simp add*: *rtranclp-unfold full-def*)
 then show *?case*
  using *other'* **by** (*meson cdcl*$_W$*.other cdcl*$_W$*-axioms tranclp.r-into-trancl*
   tranclp-cdcl$_W$*-cp-tranclp-cdcl*$_W$ *tranclp-trans*)
qed

lemma *tranclp-cdcl*$_W$*-stgy-tranclp-cdcl*$_W$:
  cdcl$_W$*-stgy*$^{++}$ *S S'* $\Longrightarrow$ *cdcl*$_W$$^{++}$ *S S'*
  apply (*induct rule*: *tranclp.induct*)
  using *cdcl*$_W$*-stgy-tranclp-cdcl*$_W$ **apply** *blast*
  by (*meson cdcl*$_W$*-stgy-tranclp-cdcl*$_W$ *tranclp-trans*)

lemma *rtranclp-cdcl*$_W$*-stgy-rtranclp-cdcl*$_W$:
  cdcl$_W$*-stgy*** *S S'* $\Longrightarrow$ *cdcl*$_W$** *S S'*
 using *rtranclp-unfold*[*of cdcl*$_W$*-stgy S S'*] *tranclp-cdcl*$_W$*-stgy-tranclp-cdcl*$_W$[*of S S'*] **by** *auto*

lemma *cdcl*$_W$*-o-conflict-is-false-with-level-inv*:
 assumes
  cdcl$_W$*-o S S'* **and**
  lev: *cdcl*$_W$*-M-level-inv S* **and**
  confl-inv: *conflict-is-false-with-level S* **and**
  n-d: *distinct-cdcl*$_W$*-state S* **and**
  conflicting: *cdcl*$_W$*-conflicting S*
 shows *conflict-is-false-with-level S'*
 using *assms*(*1*,*2*)
proof (*induct rule*: *cdcl*$_W$*-o-induct-lev2*)
 case (*resolve L C M D T*) **note** *tr-S* = *this*(*1*) **and** *confl* = *this*(*2*) **and** *T* = *this*(*4*)
 have $-L \notin\#$ *D* **using** *n-d confl* **unfolding** *distinct-cdcl*$_W$*-state-def distinct-mset-def* **by** *auto*
 moreover have $L \notin\#$ *D*
  proof (*rule ccontr*)
   assume ¬ *?thesis*
   moreover have *Propagated L* (*C* + {#*L*#}) # *M* $\models$*as CNot D*
    using *conflicting confl tr-S* **unfolding** *cdcl*$_W$*-conflicting-def* **by** *auto*
   ultimately have $-L \in$ *lits-of* (*Propagated L* ((*C* + {#*L*#})) # *M*)
    using *in-CNot-implies-uminus*(*2*) **by** *blast*
   moreover have *no-dup* (*Propagated L* ((*C* + {#*L*#})) # *M*)
    using *lev tr-S* **unfolding** *cdcl*$_W$*-M-level-inv-def* **by** *auto*
   ultimately show *False* **unfolding** *lits-of-def* **by** (*metis consistent-interp-def image-eqI*
    list.set-intros(*1*) *lits-of-def ann-literal.sel*(*2*) *distinctconsistent-interp*)
  qed

```
  ultimately
    have g-D: get-maximum-level (Propagated L (C + {#L#}) # M) D
      = get-maximum-level M D
    proof –
      have ∀ a f L. ((a::'v) ∈ f ‘ L) = (∃ l. (l::'v literal) ∈ L ∧ a = f l)
        by blast
      then show ?thesis
        using get-maximum-level-skip-first[of L D  (C + {#L#}) M] unfolding atms-of-def
        by (metis (no-types) ‹– L ∉# D› ‹L ∉# D› atm-of-eq-atm-of mem-set-mset-iff)
    qed
  { assume
      get-maximum-level (Propagated L (C + {#L#}) # M) D = backtrack-lvl S and
      backtrack-lvl S > 0
    then have D: get-maximum-level M D = backtrack-lvl S unfolding g-D by blast
    then have ?case
      using tr-S ‹backtrack-lvl S > 0› get-maximum-level-exists-lit[of backtrack-lvl S M D] T
      by auto
  }
  moreover {
    assume [simp]: backtrack-lvl S = 0
    have ⋀L. get-level M L = 0
      proof –
        fix L
        have atm-of L ∉ atm-of ‘ (lits-of M) ⟹ get-level M L = 0 by auto
        moreover {
          assume atm-of L ∈ atm-of ‘ (lits-of M)
          have g-r: get-all-levels-of-marked M = rev [Suc 0..<Suc (backtrack-lvl S)]
            using lev tr-S unfolding cdcl_W-M-level-inv-def by auto
          have Max (insert 0 (set (get-all-levels-of-marked M))) = (backtrack-lvl S)
            unfolding g-r by (simp add: Max-n-upt)
          then have get-level M L = 0
            using get-maximum-possible-level-ge-get-level[of M L]
            unfolding get-maximum-possible-level-max-get-all-levels-of-marked by auto
        }
        ultimately show get-level M L = 0 by blast
      qed
    then have ?case using get-maximum-level-exists-lit-of-max-level[of D#∪C M] tr-S T
      by (auto simp: Bex-mset-def)
  }
  ultimately show ?case using resolve.hyps(3) by blast
next
  case (skip L C' M D T) note tr-S = this(1) and D = this(2) and T =this(5)
  then obtain La where La ∈# D and get-level (Propagated L C' # M) La = backtrack-lvl S
    using skip confl-inv by auto
  moreover
    have atm-of La ≠ atm-of L
      proof (rule ccontr)
        assume ¬ ?thesis
        then have La: La = L using ‹La ∈# D› ‹– L ∉# D› by (auto simp add: atm-of-eq-atm-of)
        have Propagated L C' # M ⊨as CNot D
          using conflicting tr-S D unfolding cdcl_W-conflicting-def by auto
        then have –L ∈ lits-of M
          using ‹La ∈# D› in-CNot-implies-uminus(2)[of D L Propagated L C' # M] unfolding La
          by auto
```

then show *False* **using** *lev tr-S* **unfolding** *cdcl$_W$-M-level-inv-def consistent-interp-def* **by** *auto*
qed
then have *get-level* (*Propagated L C′ # M*) *La* = *get-level M La* **by** *auto*
ultimately show *?case* **using** *D tr-S T* **by** *auto*
qed (*auto split*: *split-if-asm simp*: *cdcl$_W$-M-level-inv-decomp*)

5.6.5 Strong completeness

lemma *cdcl$_W$-cp-propagate-confl*:
assumes *cdcl$_W$-cp S T*
shows *propagate** S T* ∨ (∃ *S′*. *propagate** S S′* ∧ *conflict S′ T*)
using *assms* **by** *induction blast+*

lemma *rtranclp-cdcl$_W$-cp-propagate-confl*:
assumes *cdcl$_W$-cp** S T*
shows *propagate** S T* ∨ (∃ *S′*. *propagate** S S′* ∧ *conflict S′ T*)
by (*simp add*: *assms rtranclp-cdcl$_W$-cp-propa-or-propa-confl*)

lemma *cdcl$_W$-cp-propagate-completeness*:
assumes *MN*: *set M* ⊨*s set-mset N* **and**
cons: *consistent-interp* (*set M*) **and**
tot: *total-over-m* (*set M*) (*set-mset N*) **and**
lits-of (*trail S*) ⊆ *set M* **and**
init-clss S = *N* **and**
propagate** S S′ **and**
learned-clss S = {#}
shows *length* (*trail S*) ≤ *length* (*trail S′*) ∧ *lits-of* (*trail S′*) ⊆ *set M*
using *assms*(*6*,*4*,*5*,*7*)
proof (*induction rule*: *rtranclp-induct*)
case *base*
then show *?case* **by** *auto*
next
case (*step Y Z*)
note *st* = *this*(*1*) **and** *propa* = *this*(*2*) **and** *IH* = *this*(*3*) **and** *lits′* = *this*(*4*) **and** *NS* = *this*(*5*) **and** *learned* = *this*(*6*)
then have *len*: *length* (*trail S*) ≤ *length* (*trail Y*) **and** *LM*: *lits-of* (*trail Y*) ⊆ *set M*
by *blast+*

obtain *M′ N′ U k C L* **where**
Y: *state Y* = (*M′*, *N′*, *U*, *k*, *None*) **and**
Z: *state Z* = (*Propagated L* (*C* + {#*L*#}) # *M′*, *N′*, *U*, *k*, *None*) **and**
C: *C* + {#*L*#} ∈# *clauses Y* **and**
M′-C: *M′* ⊨*as CNot C* **and**
undefined-lit (*trail Y*) *L*
using *propa* **by** *auto*
have *init-clss S* = *init-clss Y*
using *st* **by** *induction auto*
then have [*simp*]: *N′* = *N* **using** *NS Y Z* **by** *simp*
have *learned-clss Y* = {#}
using *st learned* **by** *induction auto*
then have [*simp*]: *U* = {#} **using** *Y* **by** *auto*
have *set M* ⊨*s CNot C*
using *M′-C LM Y* **unfolding** *true-annots-def Ball-def true-annot-def true-clss-def true-cls-def*
by *force*
moreover
have *set M* ⊨ *C* + {#*L*#}

using *MN C learned Y* **unfolding** *true-clss-def clauses-def*
by (*metis NS* ‹*init-clss S* = *init-clss Y*› ‹*learned-clss Y* = {#}› *add.right-neutral mem-set-mset-iff*)
ultimately have $L \in$ *set M* **by** (*simp add*: *cons consistent-CNot-not*)
then show *?case* **using** *LM len Y Z* **by** *auto*
qed

lemma *completeness-is-a-full1-propagation*:
fixes *S* :: *'st* **and** *M* :: *'v literal list*
assumes *MN*: *set M* $\models$*s set-mset N*
and *cons*: *consistent-interp* (*set M*)
and *tot*: *total-over-m* (*set M*) (*set-mset N*)
and *alien*: *no-strange-atm S*
and *learned*: *learned-clss S* = {#}
and *clsS*[*simp*]: *init-clss S* = *N*
and *lits*: *lits-of* (*trail S*) $\subseteq$ *set M*
shows $\exists S'$. *propagate*** *S S'* $\wedge$ *full* $cdcl_W$*-cp S S'*
proof –
obtain *S'* **where** *full*: *full* $cdcl_W$*-cp S S'*
using *always-exists-full-*$cdcl_W$*-cp-step alien* **by** *blast*
then consider (*propa*) *propagate*** *S S'*
| (*confl*) $\exists X$. *propagate*** *S X* $\wedge$ *conflict X S'*
using *rtranclp-*$cdcl_W$*-cp-propagate-confl* **unfolding** *full-def* **by** *blast*
then show *?thesis*
proof *cases*
case *propa* **then show** *?thesis* **using** *full* **by** *blast*
next
case *confl*
then obtain *X* **where**
X: *propagate*** *S X* **and**
Xconf: *conflict X S'*
by *blast*
have *clsX*: *init-clss X* = *init-clss S*
using *X* **by** *induction auto*
have *learnedX*: *learned-clss X* = {#} **using** *X learned* **by** *induction auto*
obtain *E* **where**
E: *E* $\in$# *init-clss X* + *learned-clss X* **and**
Not-E: *trail X* $\models$*as CNot E*
using *Xconf* **by** (*auto simp add*: *conflict.simps clauses-def*)
have *lits-of* (*trail X*) $\subseteq$ *set M*
using $cdcl_W$*-cp-propagate-completeness*[*OF assms*(*1–3*) *lits* - *X learned*] *learned* **by** *auto*
then have *MNE*: *set M* $\models$*s CNot E*
using *Not-E*
by (*fastforce simp add*: *true-annots-def true-annot-def true-clss-def true-cls-def*)
have $\neg$ *set M* $\models$*s set-mset N*
using *E consistent-CNot-not*[*OF cons MNE*]
unfolding *learnedX true-clss-def* **unfolding** *clsX clsS* **by** *auto*
then show *?thesis* **using** *MN* **by** *blast*
qed
qed

See also $cdcl_W$*-cp*** *?S ?S'* $\Longrightarrow \exists M$. *trail ?S'* = *M* @ *trail ?S* $\wedge$ ($\forall l \in$*set M*. $\neg$ *is-marked l*)

lemma *rtranclp-propagate-is-trail-append*:
propagate** *S T* $\Longrightarrow \exists c$. *trail T* = *c* @ *trail S*
by (*induction rule*: *rtranclp-induct*) *auto*

lemma *rtranclp-propagate-is-update-trail*:
  propagate** *S T* $\Longrightarrow$ *cdcl*$_W$*-M-level-inv S* $\Longrightarrow$ *T* $\sim$ *delete-trail-and-rebuild* (*trail T*) *S*
proof (*induction rule*: *rtranclp-induct*)
  case *base*
  then show *?case* **unfolding** *state-eq-def* **by** (*auto simp*: *cdcl*$_W$*-M-level-inv-decomp state-access-simp*)
next
  case (*step T U*) **note** *IH=this*(*3*)[*OF this*(*4*)]
  moreover have *cdcl*$_W$*-M-level-inv U*
    using *rtranclp-cdcl*$_W$*-consistent-inv* ‹*propagate*** *S T*› ‹*propagate T U*›
    rtranclp-mono[*of propagate cdcl*$_W$] *cdcl*$_W$*-cp-consistent-inv propagate'*
    rtranclp-propagate-is-rtranclp-cdcl$_W$ *step.prems* **by** *blast*
    then have *no-dup* (*trail U*) **unfolding** *cdcl*$_W$*-M-level-inv-def* **by** *auto*
  ultimately show *?case* **using** ‹*propagate T U*› **unfolding** *state-eq-def*
    by (*fastforce simp*: *state-access-simp*)
qed

lemma *cdcl*$_W$*-stgy-strong-completeness-n*:
  assumes
    MN: *set M* $\models$*s set-mset N* **and**
    cons: *consistent-interp* (*set M*) **and**
    tot: *total-over-m* (*set M*) (*set-mset N*) **and**
    atm-incl: *atm-of* ‘ (*set M*) $\subseteq$ *atms-of-msu N* **and**
    distM: *distinct M* **and**
    length: *n* $\leq$ *length M*
  shows
    $\exists$ *M' k S*. *length M'* $\geq$ *n* $\wedge$
      lits-of M' $\subseteq$ *set M* $\wedge$
      no-dup M' $\wedge$
      S $\sim$ *update-backtrack-lvl k* (*append-trail* (*rev M'*) (*init-state N*)) $\wedge$
      cdcl$_W$*-stgy*** (*init-state N*) *S*
  using *length*
proof (*induction n*)
  case *0*
  have *update-backtrack-lvl 0* (*append-trail* (*rev* []) (*init-state N*)) $\sim$ *init-state N*
    by (*auto simp*: *state-eq-def simp del*: *state-simp*)
  moreover have
    0 $\leq$ *length* [] **and**
    lits-of [] $\subseteq$ *set M* **and**
    cdcl$_W$*-stgy*** (*init-state N*) (*init-state N*)
    and *no-dup* []
    by (*auto simp*: *state-eq-def simp del*: *state-simp*)
  ultimately show *?case* **using** *state-eq-sym* **by** *blast*
next
  case (*Suc n*) **note** *IH* = *this*(*1*) **and** *n* = *this*(*2*)
  then obtain *M' k S* **where**
    l-M': *length M'* $\geq$ *n* **and**
    M': *lits-of M'* $\subseteq$ *set M* **and**
    n-d[*simp*]: *no-dup M'* **and**
    S: *S* $\sim$ *update-backtrack-lvl k* (*append-trail* (*rev M'*) (*init-state N*)) **and**
    st: *cdcl*$_W$*-stgy*** (*init-state N*) *S*
    by *auto*
  have
    M: *cdcl*$_W$*-M-level-inv S* **and**
    alien: *no-strange-atm S*

using *rtranclp-cdcl*$_W$*-consistent-inv*[*OF rtranclp-cdcl*$_W$*-stgy-rtranclp-cdcl*$_W$ [*OF st*]]
rtranclp-cdcl$_W$*-no-strange-atm-inv*[*OF rtranclp-cdcl*$_W$*-stgy-rtranclp-cdcl*$_W$ [*OF st*]]
S **unfolding** *state-eq-def cdcl*$_W$*-M-level-inv-def no-strange-atm-def* **by** *auto*
{ assume *no-step*: ¬*no-step propagate S*

obtain S' **where** S': *propagate*** *S* S' **and** *full*: *full cdcl*$_W$*-cp S* S'
using *completeness-is-a-full1-propagation*[*OF assms*(*1*−*3*), *of S*] *alien* M' *S*
by (*auto simp*: *state-access-simp*)
have *lev*: *cdcl*$_W$*-M-level-inv* S'
using *M* S' *rtranclp-cdcl*$_W$*-consistent-inv rtranclp-propagate-is-rtranclp-cdcl*$_W$ **by** *blast*
then have *n-d'*[*simp*]: *no-dup* (*trail* S')
unfolding *cdcl*$_W$*-M-level-inv-def* **by** *auto*
have *length* (*trail S*) ≤ *length* (*trail* S') ∧ *lits-of* (*trail* S') ⊆ *set M*
using S' *full cdcl*$_W$*-cp-propagate-completeness*[*OF assms*(*1*−*3*), *of S*] M' *S*
by (*auto simp*: *state-access-simp*)
moreover
have *full*: *full1 cdcl*$_W$*-cp S* S'
using *full no-step no-step-cdcl*$_W$*-cp-no-conflict-no-propagate*(*2*) **unfolding** *full1-def full-def*
rtranclp-unfold **by** *blast*
then have *cdcl*$_W$*-stgy S* S' **by** (*simp add*: *cdcl*$_W$*-stgy.conflict'*)
moreover
have *propa*: *propagate*$^{++}$ *S* S' **using** S' *full* **unfolding** *full1-def* **by** (*metis rtranclpD tranclpD*)
have *trail S* = M' **using** *S* **by** (*auto simp*: *state-access-simp*)
with *propa* **have** *length* (*trail* S') > *n*
using *l-M'* *propa* **by** (*induction rule*: *tranclp.induct*) *auto*
moreover
have *stS'*: *cdcl*$_W$*-stgy*** (*init-state N*) S'
using *st cdcl*$_W$*-stgy.conflict'*[*OF full*] **by** *auto*
then have *init-clss* S' = *N* **using** *stS'* *rtranclp-cdcl*$_W$*-stgy-no-more-init-clss* **by** *fastforce*
moreover
have
[*simp*]:*learned-clss* S' = {#} **and**
[*simp*]: *init-clss* S' = *init-clss S* **and**
[*simp*]: *conflicting* S' = *None*
using *tranclp-into-rtranclp*[*OF* ‹*propagate*$^{++}$ *S* S'›] *S*
rtranclp-propagate-is-update-trail[*of S* S'] *S M* **unfolding** *state-eq-def*
by (*auto simp*: *state-access-simp*)
have *S-S'*: S' ∼ *update-backtrack-lvl* (*backtrack-lvl* S')
(*append-trail* (*rev* (*trail* S')) (*init-state N*)) **using** *S*
by (*auto simp*: *state-eq-def state-access-simp simp del*: *state-simp*)
have *cdcl*$_W$*-stgy*** (*init-state* (*init-clss* S')) S'
apply (*rule rtranclp.rtrancl-into-rtrancl*)
using *st* **unfolding** ‹*init-clss* S' = *N*› **apply** *simp*
using ‹*cdcl*$_W$*-stgy S* S'› **by** *simp*
ultimately have *?case*
apply −
apply (*rule exI*[*of - trail* S'], *rule exI*[*of - backtrack-lvl* S'], *rule exI*[*of -* S'])
using *S-S'* **by** (*auto simp*: *state-eq-def simp del*: *state-simp*)
}
moreover {
assume *no-step*: *no-step propagate S*
have *?case*
proof (*cases length* M' ≥ *Suc n*)
case *True*
then show *?thesis* **using** *l-M'* M' *st M alien S* **by** *fastforce*

next
  case *False*
  then have *n'*: *length M' = n* **using** *l-M'* **by** *auto*
  have *no-confl*: *no-step conflict S*
    proof −
      { fix *D*
        assume *D ∈# N* **and** *M' ⊨as CNot D*
        then have *set M ⊨ D* **using** *MN* **unfolding** *true-clss-def* **by** *auto*
        moreover have *set M ⊨s CNot D*
          using ‹*M' ⊨as CNot D*› *M'*
          by (*metis le-iff-sup true-annots-true-cls true-clss-union-increase*)
        ultimately have *False* **using** *cons consistent-CNot-not* **by** *blast*
      }
      then show *?thesis* **using** *S* **by** (*auto simp*: *conflict.simps true-clss-def state-access-simp*)
    qed
  have *lenM*: *length M = card* (*set M*) **using** *distM* **by** (*induction M*) *auto*
  have *no-dup M'* **using** *S M* **unfolding** *cdcl*$_W$*-M-level-inv-def* **by** *auto*
  then have *card* (*lits-of M'*) = *length M'*
    by (*induction M'*) (*auto simp add*: *lits-of-def card-insert-if*)
  then have *lits-of M' ⊂ set M*
    using *n M' n' lenM* **by** *auto*
  then obtain *m* **where** *m*: *m ∈ set M* **and** *undef-m*: *m ∉ lits-of M'* **by** *auto*
  moreover have *undef*: *undefined-lit M' m*
    using *M' Marked-Propagated-in-iff-in-lits-of calculation*(*1*,*2*) *cons*
    consistent-interp-def **by** *blast*
  moreover have *atm-of m ∈ atms-of-msu* (*init-clss S*)
    using *atm-incl calculation S* **by** (*auto simp*: *state-access-simp*)
  ultimately
    have *dec*: *decide S* (*cons-trail* (*Marked m* (*k+1*)) (*incr-lvl S*))
      using *decide.intros*[*of S rev M' N - k m*
        cons-trail (*Marked m* (*k + 1*)) (*incr-lvl S*)] *S*
      by (*auto simp*: *state-access-simp*)
  let *?S' = cons-trail* (*Marked m* (*k+1*)) (*incr-lvl S*)
  have *lits-of* (*trail ?S'*) ⊆ *set M* **using** *m M' S undef* **by** (*auto simp*: *state-access-simp*)
  moreover have *no-strange-atm ?S'*
    using *alien dec M* **by** (*meson cdcl*$_W$*-no-strange-atm-inv decide other*)
  ultimately obtain *S''* **where** *S''*: *propagate*** *?S' S''* **and** *full*: *full cdcl*$_W$*-cp ?S' S''*
    using *completeness-is-a-full1-propagation*[*OF assms*(*1−3*), *of ?S'*] *S undef*
    by (*auto simp*: *state-access-simp*)
  have *cdcl*$_W$*-M-level-inv ?S'*
    using *M dec rtranclp-mono*[*of decide cdcl*$_W$] **by** (*meson cdcl*$_W$*-consistent-inv decide other*)
  then have *lev''*: *cdcl*$_W$*-M-level-inv S''*
    using *S'' rtranclp-cdcl*$_W$*-consistent-inv rtranclp-propagate-is-rtranclp-cdcl*$_W$ **by** *blast*
  then have *n-d''*: *no-dup* (*trail S''*)
    unfolding *cdcl*$_W$*-M-level-inv-def* **by** *auto*
  have *length* (*trail ?S'*) ≤ *length* (*trail S''*) ∧ *lits-of* (*trail S''*) ⊆ *set M*
    using *S'' full cdcl*$_W$*-cp-propagate-completeness*[*OF assms*(*1−3*), *of ?S' S''*] *m M' S undef*
    by (*simp add*: *state-access-simp*)
  then have *Suc n* ≤ *length* (*trail S''*) ∧ *lits-of* (*trail S''*) ⊆ *set M*
    using *l-M' S undef* **by** (*auto simp*: *state-access-simp*)
  moreover
    have *cdcl*$_W$*-M-level-inv* (*cons-trail* (*Marked m* (*Suc* (*backtrack-lvl S*)))
      (*update-backtrack-lvl* (*Suc* (*backtrack-lvl S*)) *S*))
      using *S* ‹*cdcl*$_W$*-M-level-inv* (*cons-trail* (*Marked m* (*k + 1*)) (*incr-lvl S*))› **by** *auto*
    then have *S''*: *S''* ∼ *update-backtrack-lvl* (*backtrack-lvl S''*)

(*append-trail* (*rev* (*trail* S'')) (*init-state* N))
using *rtranclp-propagate-is-update-trail*[*OF* S''] *S undef n-d*$''$ *lev*$''$
by (*auto simp del*: *state-simp simp*: *state-eq-def state-access-simp*)
then have $cdcl_W$*-stgy*** (*init-state* N) S''
using $cdcl_W$*-stgy.intros*(*2*)[*OF decide*[*OF dec*] *- full*] *no-step no-confl st*
by (*auto simp*: $cdcl_W$*-cp.simps*)
ultimately show *?thesis* **using** S'' *n-d*$''$ **by** *blast*
qed
}
ultimately show *?case* **by** *blast*
qed

lemma $cdcl_W$*-stgy-strong-completeness*:
assumes *MN*: *set M* $\models$*s set-mset N*
and *cons*: *consistent-interp* (*set M*)
and *tot*: *total-over-m* (*set M*) (*set-mset N*)
and *atm-incl*: *atm-of* ‘ (*set M*) $\subseteq$ *atms-of-msu N*
and *distM*: *distinct M*
shows
$\exists$ M' *k S*.
lits-of M' = *set M* $\wedge$
S $\sim$ *update-backtrack-lvl k* (*append-trail* (*rev* M') (*init-state N*)) $\wedge$
$cdcl_W$*-stgy*** (*init-state N*) *S* $\wedge$
final-$cdcl_W$*-state S*
proof −
from $cdcl_W$*-stgy-strong-completeness-n*[*OF assms*, *of length M*]
obtain M' *k T* **where**
l: *length M* $\leq$ *length* M' **and**
M'*-M*: *lits-of* M' $\subseteq$ *set M* **and**
no-dup: *no-dup* M' **and**
T: *T* $\sim$ *update-backtrack-lvl k* (*append-trail* (*rev* M') (*init-state N*)) **and**
st: $cdcl_W$*-stgy*** (*init-state N*) *T*
by *auto*
have *card* (*set M*) = *length M* **using** *distM* **by** (*simp add*: *distinct-card*)
moreover
have $cdcl_W$*-M-level-inv T*
using *rtranclp-*$cdcl_W$*-stgy-consistent-inv*[*OF st*] *T* **by** *auto*
then have *card* (*set* ((*map* ($\lambda l.$ *atm-of* (*lit-of l*)) M'))) = *length* M'
using *distinct-card no-dup* **by** *fastforce*
moreover have *card* (*lits-of* M') = *card* (*set* ((*map* ($\lambda l.$ *atm-of* (*lit-of l*)) M')))
using *no-dup* **unfolding** *lits-of-def* **apply** (*induction* M') **by** (*auto simp add*: *card-insert-if*)
ultimately have *card* (*set M*) $\leq$ *card* (*lits-of* M') **using** *l* **unfolding** *lits-of-def* **by** *auto*
then have *set M* = *lits-of* M'
using M'*-M card-seteq* **by** *blast*
moreover
then have M' $\models$*asm N*
using *MN* **unfolding** *true-annots-def Ball-def true-annot-def true-clss-def* **by** *auto*
then have *final-*$cdcl_W$*-state T*
using *T no-dup* **unfolding** *final-*$cdcl_W$*-state-def* **by** (*auto simp*: *state-access-simp*)
ultimately show *?thesis* **using** *st T* **by** *blast*
qed

5.6.6 No conflict with only variables of level less than backtrack level

This invariant is stronger than the previous argument in the sense that it is a property about all possible conflicts.

```
definition no-smaller-confl (S::'st) ≡
  (∀ M K i M' D. M' @ Marked K i # M = trail S ⟶ D ∈# clauses S
    ⟶ ¬M ⊨as CNot D)

lemma no-smaller-confl-init-sate[simp]:
  no-smaller-confl (init-state N) unfolding no-smaller-confl-def by auto

lemma cdcl_W-o-no-smaller-confl-inv:
  fixes S S' :: 'st
  assumes
    cdcl_W-o S S' and
    lev: cdcl_W-M-level-inv S and
    max-lev: conflict-is-false-with-level S and
    smaller: no-smaller-confl S and
    no-f: no-clause-is-false S
  shows no-smaller-confl S'
  using assms(1,2) unfolding no-smaller-confl-def
proof (induct rule: cdcl_W-o-induct-lev2)
  case (decide L T) note confl = this(1) and undef = this(2) and T = this(4)
  have [simp]: clauses T = clauses S
    using T undef by auto
  show ?case
    proof (intro allI impI)
      fix M'' K i M' Da
      assume M'' @ Marked K i # M' = trail T
      and D: Da ∈# local.clauses T
      then have tl M'' @ Marked K i # M' = trail S
        ∨ (M'' = [] ∧ Marked K i # M' = Marked L (backtrack-lvl S + 1) # trail S)
        using T undef by (cases M'') auto
      moreover {
        assume tl M'' @ Marked K i # M' = trail S
        then have ¬M' ⊨as CNot Da
          using D T undef no-f confl smaller unfolding no-smaller-confl-def smaller by fastforce
      }
      moreover {
        assume Marked K i # M' = Marked L (backtrack-lvl S + 1) # trail S
        then have ¬M' ⊨as CNot Da using no-f D confl T by auto
      }
      ultimately show ¬M' ⊨as CNot Da by fast
   qed
next
  case resolve
  then show ?case using smaller no-f max-lev unfolding no-smaller-confl-def by auto
next
  case skip
  then show ?case using smaller no-f max-lev unfolding no-smaller-confl-def by auto
next
  case (backtrack K i M1 M2 L D T) note decomp = this(1) and confl = this(3) and undef = this(6)
    and T =this(7)
  obtain c where M: trail S = c @ M2 @ Marked K (i+1) # M1
    using decomp by auto
```

show *?case*
proof (*intro allI impI*)
fix *M ia K′ M′ Da*
assume *M′ @ Marked K′ ia # M = trail T*
then have *tl M′ @ Marked K′ ia # M = M1*
using *T decomp undef lev* **by** (*cases M′*) (*auto simp*: $cdcl_W$*-M-level-inv-decomp*)
assume *D*: *Da ∈# clauses T*
moreover{
assume *Da ∈# clauses S*
then have ¬*M* ⊨*as CNot Da* **using** ‹*tl M′ @ Marked K′ ia # M = M1*› *M confl undef smaller*
unfolding *no-smaller-confl-def* **by** *auto*
}
moreover {
assume *Da*: *Da = D + {#L#}*
have ¬*M* ⊨*as CNot Da*
proof (*rule ccontr*)
assume ¬ *?thesis*
then have −*L* ∈ *lits-of M* **unfolding** *Da* **by** *auto*
then have −*L* ∈ *lits-of* (*Propagated L* ((*D + {#L#}*)) # *M1*)
using *UnI2* ‹*tl M′ @ Marked K′ ia # M = M1*›
by *auto*
moreover
have *backtrack S*
(*cons-trail* (*Propagated L* (*D + {#L#}*))
(*reduce-trail-to M1* (*add-learned-cls* (*D + {#L#}*)
(*update-backtrack-lvl i* (*update-conflicting None S*)))))
using *backtrack.intros*[*of S*] *backtrack.hyps*
by (*force simp*: *state-eq-def simp del*: *state-simp*)
then have $cdcl_W$*-M-level-inv*
(*cons-trail* (*Propagated L* (*D + {#L#}*))
(*reduce-trail-to M1* (*add-learned-cls* (*D + {#L#}*)
(*update-backtrack-lvl i* (*update-conflicting None S*)))))
using $cdcl_W$*-consistent-inv*[*OF - lev*] *other*[*OF bj*] **by** *auto*
then have *no-dup* (*Propagated L* (*D + {#L#}*) # *M1*)
using *decomp undef lev* **unfolding** $cdcl_W$*-M-level-inv-def* **by** *auto*
ultimately show *False* **by** (*metis consistent-interp-def distinctconsistent-interp*
insertCI lits-of-cons ann-literal.sel(*2*))
qed
}
ultimately show ¬*M* ⊨*as CNot Da*
using *T undef* ‹*Da = D + {#L#}* ⟹ ¬ *M* ⊨*as CNot Da*› *decomp lev*
unfolding $cdcl_W$*-M-level-inv-def* **by** *fastforce*
qed
qed

lemma *conflict-no-smaller-confl-inv*:
assumes *conflict S S′*
and *no-smaller-confl S*
shows *no-smaller-confl S′*
using *assms* **unfolding** *no-smaller-confl-def* **by** *fastforce*

lemma *propagate-no-smaller-confl-inv*:
assumes *propagate*: *propagate S S′*
and *n-l*: *no-smaller-confl S*

shows *no-smaller-confl* S'
unfolding *no-smaller-confl-def*
proof (*intro allI impI*)
fix M' K i M'' D
assume M': M'' @ *Marked* K i # M' = *trail* S'
and D ∈# *clauses* S'
obtain M N U k C L **where**
S: *state* S = (M, N, U, k, *None*) **and**
S': *state* S' = (*Propagated* L ((C + {#L#})) # M, N, U, k, *None*) **and**
C + {#L#} ∈# *clauses* S **and**
M ⊨*as CNot* C **and**
undefined-lit M L
using *propagate* **by** *auto*
have *tl* M'' @ *Marked* K i # M' = *trail* S **using** M' S S'
by (*metis Pair-inject list.inject list.sel*(*3*) *ann-literal.distinct*(*1*) *self-append-conv2*
tl-append2)
then have ¬M' ⊨*as CNot* D
using ‹D ∈# *clauses* S'› *n-l* S S' *clauses-def* **unfolding** *no-smaller-confl-def* **by** *auto*
then show ¬M' ⊨*as CNot* D **by** *auto*
qed

lemma *cdcl*$_W$*-cp-no-smaller-confl-inv*:
assumes *propagate*: *cdcl*$_W$*-cp* S S'
and *n-l*: *no-smaller-confl* S
shows *no-smaller-confl* S'
using *assms*
proof (*induct rule*: *cdcl*$_W$*-cp.induct*)
case (*conflict'* S S')
then show *?case* **using** *conflict-no-smaller-confl-inv*[*of* S S'] **by** *blast*
next
case (*propagate'* S S')
then show *?case* **using** *propagate-no-smaller-confl-inv*[*of* S S'] **by** *fastforce*
qed

lemma *rtrancp-cdcl*$_W$*-cp-no-smaller-confl-inv*:
assumes *propagate*: *cdcl*$_W$*-cp*** S S'
and *n-l*: *no-smaller-confl* S
shows *no-smaller-confl* S'
using *assms*
proof (*induct rule*: *rtranclp-induct*)
case *base*
then show *?case* **by** *simp*
next
case (*step* S' S'')
then show *?case* **using** *cdcl*$_W$*-cp-no-smaller-confl-inv*[*of* S' S''] **by** *fast*
qed

lemma *trancp-cdcl*$_W$*-cp-no-smaller-confl-inv*:
assumes *propagate*: *cdcl*$_W$*-cp*$^{++}$ S S'
and *n-l*: *no-smaller-confl* S
shows *no-smaller-confl* S'
using *assms*
proof (*induct rule*: *tranclp.induct*)
case (*r-into-trancl* S S')
then show *?case* **using** *cdcl*$_W$*-cp-no-smaller-confl-inv*[*of* S S'] **by** *blast*

next
  case (*trancl-into-trancl S S′ S′′*)
  then show *?case* **using** *cdcl*$_W$*-cp-no-smaller-confl-inv*[*of S′ S′′*] **by** *fast*
qed

lemma *full-cdcl*$_W$*-cp-no-smaller-confl-inv*:
  assumes *full cdcl*$_W$*-cp S S′*
  and *n-l*: *no-smaller-confl S*
  shows *no-smaller-confl S′*
  using *assms* **unfolding** *full-def*
  using *rtrancp-cdcl*$_W$*-cp-no-smaller-confl-inv*[*of S S′*] **by** *blast*

lemma *full1-cdcl*$_W$*-cp-no-smaller-confl-inv*:
  assumes *full1 cdcl*$_W$*-cp S S′*
  and *n-l*: *no-smaller-confl S*
  shows *no-smaller-confl S′*
  using *assms* **unfolding** *full1-def*
  using *trancp-cdcl*$_W$*-cp-no-smaller-confl-inv*[*of S S′*] **by** *blast*

lemma *cdcl*$_W$*-stgy-no-smaller-confl-inv*:
  assumes *cdcl*$_W$*-stgy S S′*
  and *n-l*: *no-smaller-confl S*
  and *conflict-is-false-with-level S*
  and *cdcl*$_W$*-M-level-inv S*
  shows *no-smaller-confl S′*
  using *assms*
proof (*induct rule*: *cdcl*$_W$*-stgy.induct*)
  case (*conflict′ S′*)
  then show *?case* **using** *full1-cdcl*$_W$*-cp-no-smaller-confl-inv*[*of S S′*] **by** *blast*
next
  case (*other′ S′ S′′*)
  have *no-smaller-confl S′*
    using *cdcl*$_W$*-o-no-smaller-confl-inv*[*OF other′.hyps*(*1*) *other′.prems*(*3*,*2*,*1*)]
    not-conflict-not-any-negated-init-clss other′.hyps(*2*) **by** *blast*
  then show *?case* **using** *full-cdcl*$_W$*-cp-no-smaller-confl-inv*[*of S′ S′′*] *other′.hyps* **by** *blast*
qed

lemma *conflict-conflict-is-no-clause-is-false-test*:
  assumes *conflict S S′*
  and ($\forall$ *D* $\in$# *init-clss S* + *learned-clss S*. *trail S* $\models$*as CNot D*
    $\longrightarrow$ ($\exists$ *L*. *L* $\in$# *D* $\wedge$ *get-level* (*trail S*) *L* = *backtrack-lvl S*))
  shows $\forall$ *D* $\in$# *init-clss S′* + *learned-clss S′*. *trail S′* $\models$*as CNot D*
    $\longrightarrow$ ($\exists$ *L*. *L* $\in$# *D* $\wedge$ *get-level* (*trail S′*) *L* = *backtrack-lvl S′*)
  using *assms* **by** *auto*

lemma *is-conflicting-exists-conflict*:
  assumes $\neg$($\forall$ *D*$\in$#*init-clss S′* + *learned-clss S′*. $\neg$ *trail S′* $\models$*as CNot D*)
  and *conflicting S′* = *None*
  shows $\exists$ *S′′*. *conflict S′ S′′*
  using *assms clauses-def not-conflict-not-any-negated-init-clss* **by** *fastforce*

lemma *cdcl*$_W$*-o-conflict-is-no-clause-is-false*:
  fixes *S S′* :: *′st*
  assumes

    cdcl$_W$*-o S S'* **and**
    lev: *cdcl*$_W$*-M-level-inv S* **and**
    max-lev: *conflict-is-false-with-level S* **and**
    no-f: *no-clause-is-false S* **and**
    no-l: *no-smaller-confl S*
  shows *no-clause-is-false S'*
    ∨ (*conflicting S'* = *None*
      ⟶ (∀ *D* ∈# *clauses S'*. *trail S'* ⊨*as CNot D*
        ⟶ (∃ *L*. *L* ∈# *D* ∧ *get-level* (*trail S'*) *L* = *backtrack-lvl S'*)))
  using *assms*(*1*,*2*)
proof (*induct rule*: *cdcl*$_W$*-o-induct-lev2*)
  case (*decide L T*) **note** *S* = *this*(*1*) **and** *undef* = *this*(*2*) **and** *T* =*this*(*4*)
  show *?case*
    proof (*rule HOL.disjI2*, *clarify*)
      fix *D*
      assume *D*: *D* ∈# *clauses T* **and** *M-D*: *trail T* ⊨*as CNot D*
      let *?M* = *trail S*
      let *?M'* = *trail T*
      let *?k* = *backtrack-lvl S*
      have ¬*?M* ⊨*as CNot D*
        using *no-f D S T undef* **by** *auto*
      have −*L* ∈# *D*
        proof (*rule ccontr*)
          assume ¬ *?thesis*
          have *?M* ⊨*as CNot D*
            unfolding *true-annots-def Ball-def true-annot-def CNot-def true-cls-def*
            proof (*intro allI impI*)
              fix *x*
              assume *x*: *x* ∈ {{#− *L*#} |*L*. *L* ∈# *D*}

              then obtain *L'* **where** *L'*: *x* = {#−*L'*#} *L'* ∈# *D* **by** *auto*
              obtain *L''* **where** *L''* ∈# *x* **and** *lits-of* (*Marked L* (*?k* + *1*) # *?M*) ⊨*l L''*
                using *M-D x T undef* **unfolding** *true-annots-def Ball-def true-annot-def CNot-def*
                true-cls-def Bex-mset-def **by** *auto*
              show ∃ *L* ∈# *x*. *lits-of ?M* ⊨*l L* **unfolding** *Bex-mset-def*
                by (*metis* ‹− *L* ∉# *D*› ‹*L''* ∈# *x*› *L'* ‹*lits-of* (*Marked L* (*?k* + *1*) # *?M*) ⊨*l L''*›
                  count-single insertE less-numeral-extra(*3*) *lits-of-cons ann-literal.sel*(*1*)
                  true-lit-def uminus-of-uminus-id)
            qed
          then show *False* **using** ‹¬ *?M* ⊨*as CNot D*› **by** *auto*
        qed
      have *atm-of L* ∉ *atm-of* ‘ (*lits-of ?M*)
        using *undef defined-lit-map* **unfolding** *lits-of-def* **by** *fastforce*
      then have *get-level* (*Marked L* (*?k* + *1*) # *?M*) (−*L*) = *?k* + *1* **by** *simp*
      then show ∃ *La*. *La* ∈# *D* ∧ *get-level ?M' La* = *backtrack-lvl T*
        using ‹−*L* ∈# *D*› *T undef* **by** *auto*
    qed
next
  case *resolve*
  then show *?case* **by** *auto*
next
  case *skip*
  then show *?case* **by** *auto*
next
  case (*backtrack K i M1 M2 L D T*) **note** *decomp* = *this*(*1*) **and** *undef* = *this*(*6*) **and** *T* =*this*(*7*)

```
  show ?case
    proof (rule HOL.disjI2, clarify)
      fix Da
      assume Da: Da ∈# clauses T
      and M-D: trail T ⊨as CNot Da
      obtain c where M: trail S = c @ M2 @ Marked K (i + 1) # M1
        using decomp by auto
      have tr-T: trail T = Propagated L (D + {#L#}) # M1
        using T decomp undef lev by (auto simp: cdcl_W-M-level-inv-decomp)
      have backtrack S T
       using backtrack.intros backtrack.hyps T by (force simp del: state-simp simp: state-eq-def)
      then have lev': cdcl_W-M-level-inv T
        using cdcl_W-consistent-inv lev other by blast
      then have − L ∉ lits-of M1
        unfolding cdcl_W-M-level-inv-def lits-of-def
        proof −
          have consistent-interp (lits-of (trail S)) ∧ no-dup (trail S)
            ∧ backtrack-lvl S = length (get-all-levels-of-marked (trail S))
            ∧ get-all-levels-of-marked (trail S)
              = rev [1..<1 + length (get-all-levels-of-marked (trail S))]
            using lev cdcl_W-M-level-inv-def by blast
          then show − L ∉ lit-of ‘ set M1
            by (metis (no-types) One-nat-def add.right-neutral add-Suc-right
              atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set backtrack.hyps(2)
              cdcl_W.backtrack-lit-skiped cdcl_W-axioms decomp lits-of-def)
        qed
      { assume Da ∈# clauses S
        then have ¬M1 ⊨as CNot Da using no-l M unfolding no-smaller-confl-def by auto
      }
      moreover {
        assume Da: Da = D + {#L#}
        have ¬M1 ⊨as CNot Da using ‹− L ∉ lits-of M1› unfolding Da by simp
      }
      ultimately have ¬M1 ⊨as CNot Da
        using Da T undef decomp lev by (fastforce simp: cdcl_W-M-level-inv-decomp)
      then have −L ∈# Da
        using M-D ‹− L ∉ lits-of M1› in-CNot-implies-uminus(2)
           true-annots-CNot-lit-of-notin-skip T unfolding tr-T
        by (smt insert-iff lits-of-cons ann-literal.sel(2))
      have g-M1: get-all-levels-of-marked M1 = rev [1..<i+1]
        using lev lev' T decomp undef unfolding cdcl_W-M-level-inv-def by auto
      have no-dup (Propagated L (D + {#L#}) # M1)
        using lev lev' T decomp undef unfolding cdcl_W-M-level-inv-def by auto
      then have L: atm-of L ∉ atm-of ‘ lits-of M1 unfolding lits-of-def by auto
      have get-level (Propagated L ((D + {#L#})) # M1) (−L) = i
        using get-level-get-rev-level-get-all-levels-of-marked[OF L,
          of [Propagated L ((D + {#L#}))]]
        by (simp add: g-M1 split: if-splits)
      then show ∃ La. La ∈# Da ∧ get-level (trail T) La = backtrack-lvl T
        using ‹−L ∈# Da› T decomp undef lev by (auto simp: cdcl_W-M-level-inv-def)
    qed
qed

lemma full1-cdcl_W-cp-exists-conflict-decompose:
  assumes confl: ∃ D∈#clauses S. trail S ⊨as CNot D
```

and *full*: *full* $cdcl_W$*-cp* *S U*
and *no-confl*: *conflicting S* = *None*
shows ∃ *T*. *propagate*** *S T* ∧ *conflict T U*
proof −
consider (*propa*) *propagate*** *S U*
| (*confl*) *T* **where** *propagate*** *S T* **and** *conflict T U*
using *full* **unfolding** *full-def* **by** (*blast dest*:*rtranclp-*$cdcl_W$*-cp-propa-or-propa-confl*)
then show *?thesis*
proof *cases*
case *confl*
then show *?thesis* **by** *blast*
next
case *propa*
then have *conflicting U* = *None*
using *no-confl* **by** *induction auto*
moreover have [*simp*]: *learned-clss U* = *learned-clss S* **and**
[*simp*]: *init-clss U* = *init-clss S*
using *propa* **by** *induction auto*
moreover
obtain *D* **where** *D*: *D*∈#*clauses U* **and**
trS: *trail S* ⊨*as CNot D*
using *confl clauses-def* **by** *auto*
obtain *M* **where** *M*: *trail U* = *M* @ *trail S*
using *full rtranclp-*$cdcl_W$*-cp-dropWhile-trail* **unfolding** *full-def* **by** *meson*
have *tr-U*: *trail U* ⊨*as CNot D*
apply (*rule true-annots-mono*)
using *trS* **unfolding** *M* **by** *simp-all*
have ∃ *V*. *conflict U V*
using ‹*conflicting U* = *None*› *D clauses-def not-conflict-not-any-negated-init-clss tr-U*
by *blast*
then have *False* **using** *full* $cdcl_W$*-cp.conflict′* **unfolding** *full-def* **by** *blast*
then show *?thesis* **by** *fast*
qed
qed

lemma *full1-*$cdcl_W$*-cp-exists-conflict-full1-decompose*:
assumes *confl*: ∃ *D*∈#*clauses S*. *trail S* ⊨*as CNot D*
and *full*: *full* $cdcl_W$*-cp* *S U*
and *no-confl*: *conflicting S* = *None*
shows ∃ *T D*. *propagate*** *S T* ∧ *conflict T U*
∧ *trail T* ⊨*as CNot D* ∧ *conflicting U* = *Some D* ∧ *D* ∈# *clauses S*
proof −
obtain *T* **where** *propa*: *propagate*** *S T* **and** *conf*: *conflict T U*
using *full1-*$cdcl_W$*-cp-exists-conflict-decompose*[*OF assms*] **by** *blast*
have *p*: *learned-clss T* = *learned-clss S init-clss T* = *init-clss S*
using *propa* **by** *induction auto*
have *c*: *learned-clss U* = *learned-clss T init-clss U* = *init-clss T*
using *conf* **by** *induction auto*
obtain *D* **where** *trail T* ⊨*as CNot D* ∧ *conflicting U* = *Some D* ∧ *D* ∈# *clauses S*
using *conf p c* **by** (*fastforce simp*: *clauses-def*)
then show *?thesis*
using *propa conf* **by** *blast*
qed

lemma $cdcl_W$*-stgy-no-smaller-confl*:

assumes *$cdcl_W$-stgy S S'*
and *n-l*: *no-smaller-confl S*
and *conflict-is-false-with-level S*
and *$cdcl_W$-M-level-inv S*
and *no-clause-is-false S*
and *distinct-$cdcl_W$-state S*
and *$cdcl_W$-conflicting S*
shows *no-smaller-confl S'*
using *assms*
proof (*induct rule*: *$cdcl_W$-stgy.induct*)
case (*conflict' S'*)
show *no-smaller-confl S'*
using *conflict'.hyps conflict'.prems(1) full1-$cdcl_W$-cp-no-smaller-confl-inv* **by** *blast*
next
case (*other' S' S''*)
have *lev'*: *$cdcl_W$-M-level-inv S'*
using *$cdcl_W$-consistent-inv other other'.hyps(1) other'.prems(3)* **by** *blast*
show *no-smaller-confl S''*
using *$cdcl_W$-stgy-no-smaller-confl-inv[OF $cdcl_W$-stgy.other'[OF other'.hyps(1−3)]]*
other'.prems(1−3) **by** *blast*
qed

lemma *$cdcl_W$-stgy-ex-lit-of-max-level*:
assumes *$cdcl_W$-stgy S S'*
and *n-l*: *no-smaller-confl S*
and *conflict-is-false-with-level S*
and *$cdcl_W$-M-level-inv S*
and *no-clause-is-false S*
and *distinct-$cdcl_W$-state S*
and *$cdcl_W$-conflicting S*
shows *conflict-is-false-with-level S'*
using *assms*
proof (*induct rule*: *$cdcl_W$-stgy.induct*)
case (*conflict' S'*)
have *no-smaller-confl S'*
using *conflict'.hyps conflict'.prems(1) full1-$cdcl_W$-cp-no-smaller-confl-inv* **by** *blast*
moreover have *conflict-is-false-with-level S'*
using *conflict'.hyps conflict'.prems(2−4)*
rtranclp-$cdcl_W$-co-conflict-ex-lit-of-max-level[of S S']
unfolding *full-def full1-def rtranclp-unfold* **by** *presburger*
then show *?case* **by** *blast*
next
case (*other' S' S''*)
have *lev'*: *$cdcl_W$-M-level-inv S'*
using *$cdcl_W$-consistent-inv other other'.hyps(1) other'.prems(3)* **by** *blast*
moreover
have *no-clause-is-false S'*
$\lor$ (*conflicting S' = None* $\longrightarrow$ ($\forall$ *D*$\in$#*clauses S'*. *trail S'* $\models$*as CNot D*
$\longrightarrow$ ($\exists$ *L*. *L* $\in$# *D* $\wedge$ *get-level* (*trail S'*) *L* = *backtrack-lvl S'*)))
using *$cdcl_W$-o-conflict-is-no-clause-is-false[of S S'] other'.hyps(1) other'.prems(1−4)* **by** *fast*
moreover {
assume *no-clause-is-false S'*
{
assume *conflicting S' = None*
then have *conflict-is-false-with-level S'* **by** *auto*

moreover have *full* $cdcl_W$*-cp* S' S''
by (*metis* (*no-types*) *other'.hyps*(*3*))
ultimately have *conflict-is-false-with-level* S''
using *rtranclp-*$cdcl_W$*-co-conflict-ex-lit-of-max-level*[*of* S' S''] *lev'* ‹*no-clause-is-false* S'›
by *blast*
}
moreover
{
assume *c*: *conflicting* $S' \neq None$
have *conflicting* $S \neq None$ **using** *other'.hyps*(*1*) *c*
by (*induct rule*: $cdcl_W$*-o-induct*) *auto*
then have *conflict-is-false-with-level* S'
using $cdcl_W$*-o-conflict-is-false-with-level-inv*[*OF other'.hyps*(*1*)]
other'.prems(*3*,*5*,*6*,*2*) **by** *blast*
moreover have $cdcl_W$*-cp*** S' S'' **using** *other'.hyps*(*3*) **unfolding** *full-def* **by** *auto*
then have $S' = S''$ **using** *c*
by (*induct rule*: *rtranclp-induct*)
(*fastforce intro*: *option.exhaust*)+
ultimately have *conflict-is-false-with-level* S'' **by** *auto*
}
ultimately have *conflict-is-false-with-level* S'' **by** *blast*
}
moreover {
assume
confl: *conflicting* $S' = None$ **and**
D-L: $\forall$ *D* $\in\#$ *clauses* S'. *trail* S' $\models$*as CNot D*
$\longrightarrow$ ($\exists$ *L*. *L* $\in\#$ *D* $\wedge$ *get-level* (*trail* S') *L* = *backtrack-lvl* S')
{ **assume** $\forall$ *D*$\in\#$*clauses* S'. $\neg$ *trail* S' $\models$*as CNot D*
then have *no-clause-is-false* S' **using** *confl* **by** *simp*
then have *conflict-is-false-with-level* S'' **using** *calculation*(*3*) **by** *presburger*
}
moreover {
assume $\neg$($\forall$ *D*$\in\#$*clauses* S'. $\neg$ *trail* S' $\models$*as CNot D*)
then obtain *T D* **where**
propagate** S' *T* **and**
conflict *T* S'' **and**
D: *D* $\in\#$ *clauses* S' **and**
trail S'' $\models$*as CNot D* **and**
conflicting S'' = *Some D*
using *full1-*$cdcl_W$*-cp-exists-conflict-full1-decompose*[*OF - - confl*]
other'(*3*) **by** (*metis* (*mono-tags*, *lifting*) *ball-msetI bex-msetI conflictE state-eq-trail*
trail-update-conflicting)
obtain *M* **where** *M*: *trail* S'' = *M* @ *trail* S' **and** *nm*: $\forall$ *m*$\in$*set M*. $\neg$*is-marked m*
using *rtranclp-*$cdcl_W$*-cp-dropWhile-trail other'*(*3*) **unfolding** *full-def* **by** *meson*
have *btS*: *backtrack-lvl* S'' = *backtrack-lvl* S'
using *other'.hyps*(*3*) **unfolding** *full-def* **by** (*metis rtranclp-*$cdcl_W$*-cp-backtrack-lvl*)
have *inv*: $cdcl_W$*-M-level-inv* S''
by (*metis* (*no-types*) $cdcl_W$*-stgy.conflict'* $cdcl_W$*-stgy-consistent-inv full-unfold lev'*
other'.hyps(*3*))
then have *nd*: *no-dup* (*trail* S'')
by (*metis* (*no-types*) $cdcl_W$*-M-level-inv-decomp*(*2*))
have *conflict-is-false-with-level* S''
proof *cases*
assume *trail* S' $\models$*as CNot D*
moreover then obtain *L* **where**

$L \in\# D$ **and**
lev-L: *get-level* (*trail* S') L = *backtrack-lvl* S'
using *D-L D* **by** *blast*
moreover
have LS': $-L \in$ *lits-of* (*trail* S')
using ‹*trail* $S' \models$*as CNot D*› ‹$L \in\# D$› *in-CNot-implies-uminus*(*2*) **by** *blast*
{ fix x :: ($'v$, *nat*, $'v$ *literal multiset*) *ann-literal* **and**
xb :: ($'v$, *nat*, $'v$ *literal multiset*) *ann-literal*
assume *a1*: $x \in$ *set* (*trail* S') **and**
a2: *xb* $\in$ *set M* **and**
a3: (λl. *atm-of* (*lit-of l*)) ‘ *set M* $\cap$ (λl. *atm-of* (*lit-of l*)) ‘ *set* (*trail* S')
= {} **and**
a4: $-$ L = *lit-of x* **and**
a5: *atm-of L* = *atm-of* (*lit-of xb*)
moreover have *atm-of* (*lit-of x*) = *atm-of L*
using *a4* **by** (*metis* (*no-types*) *atm-of-uminus*)
ultimately have *False*
using *a5 a3 a2 a1* **by** *auto*
}
then have *atm-of L* $\notin$ *atm-of* ‘ *lits-of M*
using *nd* LS' **unfolding** *M* **by** (*auto simp add*: *lits-of-def*)
then have *get-level* (*trail* S'') L = *get-level* (*trail* S') L
unfolding *M* **by** (*simp add*: *lits-of-def*)
ultimately show *?thesis* **using** *btS* ‹*conflicting* S'' = *Some D*› **by** *auto*
next
assume $\neg$*trail* $S' \models$*as CNot D*
then obtain L **where** $L \in\# D$ **and** *LM*: $-L \in$ *lits-of M*
using ‹*trail* $S'' \models$*as CNot D*›
by (*auto simp add*: *CNot-def true-cls-def M true-annots-def true-annot-def*
split: *split-if-asm*)
{ fix x :: ($'v$, *nat*, $'v$ *literal multiset*) *ann-literal* **and**
xb :: ($'v$, *nat*, $'v$ *literal multiset*) *ann-literal*
assume *a1*: *xb* $\in$ *set* (*trail* S') **and**
a2: $x \in$ *set M* **and**
a3: *atm-of L* = *atm-of* (*lit-of xb*) **and**
a4: $-$ L = *lit-of x* **and**
a5: (λl. *atm-of* (*lit-of l*)) ‘ *set M* $\cap$ (λl. *atm-of* (*lit-of l*)) ‘ *set* (*trail* S')
= {}
moreover have *atm-of* (*lit-of xb*) = *atm-of* ($-$ L)
using *a3* **by** *simp*
ultimately have *False*
by *auto* **}**
then have LS': *atm-of L* $\notin$ *atm-of* ‘ *lits-of* (*trail* S')
using *nd* ‹$L \in\# D$› *LM* **unfolding** *M* **by** (*auto simp add*: *lits-of-def*)
show *?thesis*
proof *cases*
assume *ne*: *get-all-levels-of-marked* (*trail* S') = []
have *backtrack-lvl* S'' = *0*
using *inv ne nm* **unfolding** $cdcl_W$*-M-level-inv-def M*
by (*simp add*: *get-all-levels-of-marked-nil-iff-not-is-marked*)
moreover
have *a1*: *get-level M L* = *0*
using *nm* **by** *auto*
then have *get-level* (*M* @ *trail* S') L = *0*
by (*metis* LS' *get-all-levels-of-marked-nil-iff-not-is-marked*

get-level-skip-beginning-not-marked lits-of-def ne)
ultimately show *?thesis* **using** ‹*conflicting* S'' = *Some D*› ‹$L \in\#$ *D*› **unfolding** *M*
by *auto*
next
assume *ne*: *get-all-levels-of-marked* (*trail* S') $\neq$ []
have *hd* (*get-all-levels-of-marked* (*trail* S')) = *backtrack-lvl* S'
using *ne lev' M nm* **unfolding** $cdcl_W$*-M-level-inv-def*
by (*cases get-all-levels-of-marked* (*trail* S'))
(*simp-all add*: *get-all-levels-of-marked-nil-iff-not-is-marked*[*symmetric*])
moreover have *atm-of L* $\in$ *atm-of* ‘ *lits-of M*
using ‹$-L \in$ *lits-of M*›
by (*simp add*: *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set lits-of-def*)
ultimately show *?thesis*
using *nm ne* ‹$L \in\# D$› ‹*conflicting* S'' = *Some D*›
get-level-skip-beginning-hd-get-all-levels-of-marked[*OF LS'*, *of M*]
get-level-skip-in-all-not-marked[*of rev M L backtrack-lvl* S']
unfolding *lits-of-def btS M*
by *auto*
qed
qed
}
ultimately have *conflict-is-false-with-level* S'' **by** *blast*
}
moreover
{
assume *conflicting* $S' \neq$ *None*
have *no-clause-is-false* S' **using** ‹*conflicting* $S' \neq$ *None*› **by** *auto*
then have *conflict-is-false-with-level* S'' **using** *calculation*(*3*) **by** *presburger*
}
ultimately show *?case* **by** *fast*
qed

lemma *rtranclp*-$cdcl_W$*-stgy-no-smaller-confl-inv*:
assumes
$cdcl_W$*-stgy*** *S* S' **and**
n-l: *no-smaller-confl S* **and**
cls-false: *conflict-is-false-with-level S* **and**
lev: $cdcl_W$*-M-level-inv S* **and**
no-f: *no-clause-is-false S* **and**
dist: *distinct*-$cdcl_W$*-state S* **and**
conflicting: $cdcl_W$*-conflicting S* **and**
decomp: *all-decomposition-implies-m* (*init-clss S*) (*get-all-marked-decomposition* (*trail S*)) **and**
learned: $cdcl_W$*-learned-clause S* **and**
alien: *no-strange-atm S*
shows *no-smaller-confl* S' $\wedge$ *conflict-is-false-with-level* S'
using *assms*(*1*)
proof (*induct rule*: *rtranclp-induct*)
case *base*
then show *?case* **using** *n-l cls-false* **by** *auto*
next
case (*step* S' S'') **note** *st* = *this*(*1*) **and** *cdcl* = *this*(*2*) **and** *IH* = *this*(*3*)
have *no-smaller-confl* S' **and** *conflict-is-false-with-level* S'
using *IH* **by** *blast+*
moreover have $cdcl_W$*-M-level-inv* S'
using *st lev rtranclp*-$cdcl_W$*-stgy-rtranclp*-$cdcl_W$

by (*blast intro*: *rtranclp-cdcl$_W$-consistent-inv*)+
moreover have *no-clause-is-false* S'
using *st no-f rtranclp-cdcl$_W$-stgy-not-non-negated-init-clss* **by** *presburger*
moreover have *distinct-cdcl$_W$-state* S'
using *rtanclp-distinct-cdcl$_W$-state-inv*[*of* S S'] *lev rtranclp-cdcl$_W$-stgy-rtranclp-cdcl$_W$*[*OF st*] *dist* **by** *auto*
moreover have *cdcl$_W$-conflicting* S'
using *rtranclp-cdcl$_W$-all-inv*(*6*)[*of* S S'] *st alien conflicting decomp dist learned lev rtranclp-cdcl$_W$-stgy-rtranclp-cdcl$_W$* **by** *blast*
ultimately show *?case*
using *cdcl$_W$-stgy-no-smaller-confl*[*OF cdcl*] *cdcl$_W$-stgy-ex-lit-of-max-level*[*OF cdcl*] **by** *fast*
qed

5.6.7 Final States are Conclusive

lemma *full-cdcl$_W$-stgy-final-state-conclusive-non-false*:
fixes S' :: *'st*
assumes *full*: *full cdcl$_W$-stgy* (*init-state* N) S'
and *no-d*: *distinct-mset-mset* N
and *no-empty*: $\forall D \in \# N.\ D \neq \{\#\}$
shows (*conflicting* S' = *Some* $\{\#\}$ $\wedge$ *unsatisfiable* (*set-mset* (*init-clss* S')))
$\vee$ (*conflicting* S' = *None* $\wedge$ *trail* S' $\models$*asm init-clss* S')
proof −
let *?S* = *init-state* N
have
termi: $\forall S''$. $\neg$*cdcl$_W$-stgy* S' S'' **and**
step: *cdcl$_W$-stgy*** (*init-state* N) S' **using** *full* **unfolding** *full-def* **by** *auto*
moreover have
learned: *cdcl$_W$-learned-clause* S' **and**
level-inv: *cdcl$_W$-M-level-inv* S' **and**
alien: *no-strange-atm* S' **and**
no-dup: *distinct-cdcl$_W$-state* S' **and**
confl: *cdcl$_W$-conflicting* S' **and**
decomp: *all-decomposition-implies-m* (*init-clss* S') (*get-all-marked-decomposition* (*trail* S'))
using *no-d tranclp-cdcl$_W$-stgy-tranclp-cdcl$_W$*[*of ?S* S'] *step rtranclp-cdcl$_W$-all-inv*(*1−6*)[*of ?S* S']
unfolding *rtranclp-unfold* **by** *auto*
moreover
have $\forall D \in \# N.\ \neg$ [] $\models$*as CNot D* **using** *no-empty* **by** *auto*
then have *confl-k*: *conflict-is-false-with-level* S'
using *rtranclp-cdcl$_W$-stgy-no-smaller-confl-inv*[*OF step*] *no-d* **by** *auto*
show *?thesis*
using *cdcl$_W$-stgy-final-state-conclusive*[*OF termi decomp learned level-inv alien no-dup confl confl-k*] **.**
qed

lemma *conflict-is-full1-cdcl$_W$-cp*:
assumes *cp*: *conflict* S S'
shows *full1 cdcl$_W$-cp* S S'
proof −
have *cdcl$_W$-cp* S S' **and** *conflicting* $S' \neq$ *None* **using** *cp cdcl$_W$-cp.intros* **by** *auto*
then have *cdcl$_W$-cp^{++}* S S' **by** *blast*
moreover have *no-step cdcl$_W$-cp* S'
using ‹*conflicting* $S' \neq$ *None*› **by** (*metis cdcl$_W$-cp-conflicting-not-empty option.exhaust*)
ultimately show *full1 cdcl$_W$-cp* S S' **unfolding** *full1-def* **by** *blast*+

qed

lemma *$cdcl_W$-cp-fst-empty-conflicting-false*:
assumes *$cdcl_W$-cp S S'*
and *trail S = []*
and *conflicting S ≠ None*
shows *False*
using *assms* **by** (*induct rule*: *$cdcl_W$-cp.induct*) *auto*

lemma *$cdcl_W$-o-fst-empty-conflicting-false*:
assumes *$cdcl_W$-o S S'*
and *trail S = []*
and *conflicting S ≠ None*
shows *False*
using *assms* **by** (*induct rule*: *$cdcl_W$-o-induct*) *auto*

lemma *$cdcl_W$-stgy-fst-empty-conflicting-false*:
assumes *$cdcl_W$-stgy S S'*
and *trail S = []*
and *conflicting S ≠ None*
shows *False*
using *assms* **apply** (*induct rule*: *$cdcl_W$-stgy.induct*)
using *tranclpD $cdcl_W$-cp-fst-empty-conflicting-false* **unfolding** *full1-def* **apply** *metis*
using *$cdcl_W$-o-fst-empty-conflicting-false* **by** *blast*
thm *$cdcl_W$-cp.induct*[*split-format*(*complete*)]

lemma *$cdcl_W$-cp-conflicting-is-false*:
$cdcl_W$-cp S S' ⟹ conflicting S = Some {#} ⟹ False
by (*induction rule*: *$cdcl_W$-cp.induct*) *auto*

lemma *rtranclp-$cdcl_W$-cp-conflicting-is-false*:
$cdcl_W$-cp^{++} S S' ⟹ conflicting S = Some {#} ⟹ False
apply (*induction rule*: *tranclp.induct*)
by (*auto dest*: *$cdcl_W$-cp-conflicting-is-false*)

lemma *$cdcl_W$-o-conflicting-is-false*:
$cdcl_W$-o S S' ⟹ conflicting S = Some {#} ⟹ False
by (*induction rule*: *$cdcl_W$-o-induct*) *auto*

lemma *$cdcl_W$-stgy-conflicting-is-false*:
$cdcl_W$-stgy S S' ⟹ conflicting S = Some {#} ⟹ False
apply (*induction rule*: *$cdcl_W$-stgy.induct*)
unfolding *full1-def* **apply** (*metis* (*no-types*) *$cdcl_W$-cp-conflicting-not-empty tranclpD*)
unfolding *full-def* **by** (*metis conflict-with-false-implies-terminated other*)

lemma *rtranclp-$cdcl_W$-stgy-conflicting-is-false*:
$cdcl_W$-stgy** S S' ⟹ conflicting S = Some {#} ⟹ S' = S
apply (*induction rule*: *rtranclp-induct*)
apply *simp*
using *$cdcl_W$-stgy-conflicting-is-false* **by** *blast*

lemma *full-$cdcl_W$-init-clss-with-false-normal-form*:
assumes
∀ m∈ set M. ¬is-marked m **and**

E = *Some D* **and**
state S = (*M*, *N*, *U*, *0*, *E*)
full $cdcl_W$*-stgy S S'* **and**
all-decomposition-implies-m (*init-clss S*) (*get-all-marked-decomposition* (*trail S*))
$cdcl_W$*-learned-clause S*
$cdcl_W$*-M-level-inv S*
no-strange-atm S
distinct-$cdcl_W$*-state S*
$cdcl_W$*-conflicting S*
shows ∃ *M''*. *state S'* = (*M''*, *N*, *U*, *0*, *Some* {#})
using *assms*(*10*,*9*,*8*,*7*,*6*,*5*,*4*,*3*,*2*,*1*)
proof (*induction M arbitrary*: *E D S*)
case *Nil*
then show *?case*
using *rtranclp-*$cdcl_W$*-stgy-conflicting-is-false* **unfolding** *full-def* $cdcl_W$*-conflicting-def* **by** *auto*
next
case (*Cons L M*) **note** *IH* = *this*(*1*) **and** *full* = *this*(*8*) **and** *E* = *this*(*10*) **and** *inv* = *this*(*2*−*7*) **and** *S* = *this*(*9*) **and** *nm* = *this*(*11*)
obtain *K p* **where** *K*: *L* = *Propagated K p*
using *nm* **by** (*cases L*) *auto*
have *every-mark-is-a-conflict S* **using** *inv* **unfolding** $cdcl_W$*-conflicting-def* **by** *auto*
then have *MpK*: *M* ⊨*as CNot* (*p* − {#*K*#}) **and** *Kp*: *K* ∈# *p*
using *S* **unfolding** *K* **by** *fastforce+*
then have *p*: *p* = (*p* − {#*K*#}) + {#*K*#}
by (*auto simp add*: *multiset-eq-iff*)
then have *K'*: *L* = *Propagated K* (((*p* − {#*K*#}) + {#*K*#}))
using *K* **by** *auto*

consider (*D*) *D* = {#} | (*D'*) *D* ≠ {#} **by** *blast*
then show *?case*
proof *cases*
case *D*
then show *?thesis*
using *full rtranclp-*$cdcl_W$*-stgy-conflicting-is-false S* **unfolding** *full-def E D* **by** *auto*
next
case *D'*
then have *no-p*: *no-step propagate S* **and** *no-c*: *no-step conflict S*
using *S E* **by** *auto*
then have *no-step* $cdcl_W$*-cp S* **by** (*auto simp*: $cdcl_W$*-cp.simps*)
have *res-skip*: ∃ *T*. (*resolve S T* ∧ *no-step skip S* ∧ *full* $cdcl_W$*-cp T T*)
∨ (*skip S T* ∧ *no-step resolve S* ∧ *full* $cdcl_W$*-cp T T*)
proof *cases*
assume −*lit-of L* ∉# *D*
then obtain *T* **where** *sk*: *skip S T* **and** *res*: *no-step resolve S*
using *S that D' K* **unfolding** *skip.simps E* **by** *fastforce*
have *full* $cdcl_W$*-cp T T*
using *sk* **by** (*auto simp add*: *option-full-*$cdcl_W$*-cp*)
then show *?thesis*
using *sk res* **by** *blast*
next
assume *LD*: ¬−*lit-of L* ∉# *D*
then have *D*: *Some D* = *Some* ((*D* − {#−*lit-of L*#}) + {#−*lit-of L*#})
by (*auto simp add*: *multiset-eq-iff*)

have ⋀*L*. *get-level M L* = *0*

```
          by (simp add: nm)
          then have get-maximum-level (Propagated K (p − {#K#} + {#K#}) # M) (D − {#−
K#}) = 0
          using  LD get-maximum-level-exists-lit-of-max-level
          proof −
            obtain L' where get-level (L#M) L' = get-maximum-level (L#M) D
              using  LD get-maximum-level-exists-lit-of-max-level[of D L#M] by fastforce
            then show ?thesis by (metis (mono-tags) K' bex-msetE get-level-skip-all-not-marked
              get-maximum-level-exists-lit nm not-gr0)
          qed
        then obtain T where sk: resolve S T and res: no-step skip S
          using resolve-rule[of S K  p − {#K#} M N U 0 (D − {#−K#})
          update-conflicting (Some (remdups-mset (D − {#− K#} + (p − {#K#})))) (tl-trail S)]
          S unfolding K' D E  by fastforce
        have full cdcl_W-cp T T
          using sk by (auto simp add: option-full-cdcl_W-cp)
        then show ?thesis
         using sk res by blast
      qed
    then have step-s: ∃ T. cdcl_W-stgy S T
      using ‹no-step cdcl_W-cp S› other' by (meson bj resolve skip)
    have get-all-marked-decomposition (L # M) = [([], L#M)]
      using nm unfolding K apply (induction M rule: ann-literal-list-induct, simp)
        by (rename-tac L l xs, case-tac hd (get-all-marked-decomposition xs), auto)+
    then have no-b: no-step backtrack S
      using nm S by auto
    have no-d: no-step decide S
      using S E by auto

    have full-S-S: full cdcl_W-cp S S
      using S E by (auto simp add: option-full-cdcl_W-cp)
    then have no-f: no-step (full1 cdcl_W-cp) S
      unfolding full-def full1-def rtranclp-unfold by (meson tranclpD)
    obtain T where
      s: cdcl_W-stgy S T and st: cdcl_W-stgy** T S'
      using full step-s full unfolding full-def by (metis rtranclp-unfold tranclpD)
    have resolve S T ∨ skip S T
      using s no-b no-d res-skip full-S-S unfolding cdcl_W-stgy.simps cdcl_W-o.simps full-unfold
      full1-def
      by (auto dest!: tranclpD simp: cdcl_W-bj.simps)
    then obtain D' where T: state T = (M, N, U, 0, Some D')
      using S E by auto

    have st-c: cdcl_W** S T
      using E T rtranclp-cdcl_W-stgy-rtranclp-cdcl_W  s by blast
    have cdcl_W-conflicting T
      using rtranclp-cdcl_W-all-inv(6)[OF st-c  inv(6,5,4,3,2,1)]  .
    show ?thesis
      apply (rule IH[of T])
                using rtranclp-cdcl_W-all-inv(6)[OF st-c inv(6,5,4,3,2,1)] apply blast
              using rtranclp-cdcl_W-all-inv(5)[OF st-c inv(6,5,4,3,2,1)] apply blast
             using rtranclp-cdcl_W-all-inv(4)[OF st-c inv(6,5,4,3,2,1)] apply blast
            using rtranclp-cdcl_W-all-inv(3)[OF st-c inv(6,5,4,3,2,1)] apply blast
           using rtranclp-cdcl_W-all-inv(2)[OF st-c inv(6,5,4,3,2,1)] apply blast
          using rtranclp-cdcl_W-all-inv(1)[OF st-c inv(6,5,4,3,2,1)] apply blast
```

apply (*metis full-def st full*)
using *T E* **apply** *blast*
apply *auto*[]
using *nm* **by** *simp*
qed
qed

lemma *full-cdcl$_W$-stgy-final-state-conclusive-is-one-false*:
fixes S' :: *'st*
assumes *full*: *full cdcl$_W$-stgy* (*init-state N*) S'
and *no-d*: *distinct-mset-mset N*
and *empty*: {#} ∈# *N*
shows *conflicting* S' = *Some* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss* S'))
proof −
let *?S* = *init-state N*
have *cdcl$_W$-stgy** ?S* S' **and** *no-step cdcl$_W$-stgy* S' **using** *full* **unfolding** *full-def* **by** *auto*
then have *plus-or-eq*: *cdcl$_W$-stgy^{++} ?S* S' ∨ S' = *?S* **unfolding** *rtranclp-unfold* **by** *auto*
have ∃ S''. *conflict ?S* S'' **using** *empty not-conflict-not-any-negated-init-clss* **by** *force*

then have *cdcl$_W$-stgy*: ∃ S'. *cdcl$_W$-stgy ?S* S'
using *cdcl$_W$-cp.conflict'*[*of ?S*] *conflict-is-full1-cdcl$_W$-cp cdcl$_W$-stgy.intros*(*1*) **by** *metis*
have S' ≠ *?S* **using** ‹*no-step cdcl$_W$-stgy* S'› *cdcl$_W$-stgy* **by** *blast*

then obtain *St*:: *'st* **where** *St*: *cdcl$_W$-stgy ?S St* **and** *cdcl$_W$-stgy** St* S'
using *plus-or-eq* **by** (*metis* (*no-types*) ‹*cdcl$_W$-stgy** ?S* S'› *converse-rtranclpE*)
have *st*: *cdcl$_W$** ?S St*
by (*simp add*: *rtranclp-unfold* ‹*cdcl$_W$-stgy ?S St*› *cdcl$_W$-stgy-tranclp-cdcl$_W$*)

have ∃ *T*. *conflict ?S T*
using *empty not-conflict-not-any-negated-init-clss* **by** *force*
then have *fullSt*: *full1 cdcl$_W$-cp ?S St*
using *St* **unfolding** *cdcl$_W$-stgy.simps* **by** *blast*
then have *bt*: *backtrack-lvl St* = (*0*::*nat*)
using *rtranclp-cdcl$_W$-cp-backtrack-lvl* **unfolding** *full1-def*
by (*fastforce dest*!: *tranclp-into-rtranclp*)
have *cls-St*: *init-clss St* = *N*
using *fullSt cdcl$_W$-stgy-no-more-init-clss*[*OF St*] **by** *auto*
have *conflicting St* ≠ *None*
proof (*rule ccontr*)
assume ¬ *?thesis*
then have ∃ *T*. *conflict St T*
using *empty cls-St*[] *conflict-rule*[*of St trail St N learned-clss St backtrack-lvl St* {#}]
by (*auto simp*: *clauses-def*)
then show *False* **using** *fullSt* **unfolding** *full1-def* **by** *blast*
qed

have *1*: ∀ *m*∈*set* (*trail St*). ¬ *is-marked m*
using *fullSt* **unfolding** *full1-def* **by** (*auto dest*!: *tranclp-into-rtranclp rtranclp-cdcl$_W$-cp-dropWhile-trail*)
have *2*: *full cdcl$_W$-stgy St* S'
using ‹*cdcl$_W$-stgy** St* S'› ‹*no-step cdcl$_W$-stgy* S'› *bt* **unfolding** *full-def* **by** *auto*
have *3*: *all-decomposition-implies-m*
(*init-clss St*)
(*get-all-marked-decomposition*

(*trail St*))
using *rtranclp-cdcl*$_W$*-all-inv*(*1*)[*OF st*] *no-d bt* **by** *simp*
have *4*: *cdcl*$_W$*-learned-clause St*
using *rtranclp-cdcl*$_W$*-all-inv*(*2*)[*OF st*] *no-d bt bt* **by** *simp*
have *5*: *cdcl*$_W$*-M-level-inv St*
using *rtranclp-cdcl*$_W$*-all-inv*(*3*)[*OF st*] *no-d bt* **by** *simp*
have *6*: *no-strange-atm St*
using *rtranclp-cdcl*$_W$*-all-inv*(*4*)[*OF st*] *no-d bt* **by** *simp*
have *7*: *distinct-cdcl*$_W$*-state St*
using *rtranclp-cdcl*$_W$*-all-inv*(*5*)[*OF st*] *no-d bt* **by** *simp*
have *8*: *cdcl*$_W$*-conflicting St*
using *rtranclp-cdcl*$_W$*-all-inv*(*6*)[*OF st*] *no-d bt* **by** *simp*
have *init-clss* S' = *init-clss St* **and** *conflicting* S' = *Some* {#}
using ‹*conflicting St* ≠ *None*› *full-cdcl*$_W$*-init-clss-with-false-normal-form*[*OF 1*, *of - - St*]
2 3 4 5 6 7 8 St **apply** (*metis* ‹*cdcl*$_W$*-stgy*** *St S'*› *rtranclp-cdcl*$_W$*-stgy-no-more-init-clss*)
using ‹*conflicting St* ≠ *None*› *full-cdcl*$_W$*-init-clss-with-false-normal-form*[*OF 1*, *of - - St - - S'*] *2 3 4 5 6 7 8* **by** (*metis bt option.exhaust prod.inject*)

moreover have *init-clss* S' = *N*
using ‹*cdcl*$_W$*-stgy*** (*init-state N*) *S'*› *rtranclp-cdcl*$_W$*-stgy-no-more-init-clss* **by** *fastforce*
moreover have *unsatisfiable* (*set-mset N*)
by (*meson empty mem-set-mset-iff satisfiable-def true-cls-empty true-clss-def*)
ultimately show *?thesis* **by** *auto*
qed

lemma *full-cdcl*$_W$*-stgy-final-state-conclusive*:
fixes S' :: *'st*
assumes *full*: *full cdcl*$_W$*-stgy* (*init-state N*) S' **and** *no-d*: *distinct-mset-mset N*
shows (*conflicting* S' = *Some* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss* S')))
∨ (*conflicting* S' = *None* ∧ *trail* S' ⊨*asm init-clss* S')
using *assms full-cdcl*$_W$*-stgy-final-state-conclusive-is-one-false*
full-cdcl$_W$*-stgy-final-state-conclusive-non-false* **by** *blast*

lemma *full-cdcl*$_W$*-stgy-final-state-conclusive-from-init-state*:
fixes S' :: *'st*
assumes *full*: *full cdcl*$_W$*-stgy* (*init-state N*) S'
and *no-d*: *distinct-mset-mset N*
shows (*conflicting* S' = *Some* {#} ∧ *unsatisfiable* (*set-mset N*))
∨ (*conflicting* S' = *None* ∧ *trail* S' ⊨*asm N* ∧ *satisfiable* (*set-mset N*))
proof −
have *N*: *init-clss* S' = *N*
using *full* **unfolding** *full-def* **by** (*auto dest*: *rtranclp-cdcl*$_W$*-stgy-no-more-init-clss*)
consider
(*confl*) *conflicting* S' = *Some* {#} **and** *unsatisfiable* (*set-mset* (*init-clss* S'))
| (*sat*) *conflicting* S' = *None* **and** *trail* S' ⊨*asm init-clss* S'
using *full-cdcl*$_W$*-stgy-final-state-conclusive*[*OF assms*] **by** *auto*
then show *?thesis*
proof *cases*
case *confl*
then show *?thesis* **by** (*auto simp*: *N*)
next
case *sat*
have *cdcl*$_W$*-M-level-inv* (*init-state N*) **by** *auto*
then have *cdcl*$_W$*-M-level-inv* S'

using *full rtranclp-cdcl$_W$-stgy-consistent-inv* **unfolding** *full-def* **by** *blast*
then have *consistent-interp* (*lits-of* (*trail S'*)) **unfolding** *cdcl$_W$-M-level-inv-def* **by** *blast*
moreover have *lits-of* (*trail S'*) $\models$*s set-mset* (*init-clss S'*)
using *sat*(*2*) **by** (*auto simp add*: *true-annots-def true-annot-def true-clss-def*)
ultimately have *satisfiable* (*set-mset* (*init-clss S'*)) **by** *simp*
then show *?thesis* **using** *sat* **unfolding** *N* **by** *blast*
qed
qed
end
end
theory *CDCL-W-Termination*
imports *CDCL-W*
begin

context *cdcl$_W$*
begin

5.7 Termination

The condition that no learned clause is a tautology is overkill (in the sense that the no-duplicate condition is enough), but we can reuse *simple-clss*.

The invariant contains all the structural invariants that holds,

definition *cdcl$_W$-all-struct-inv* **where**
cdcl$_W$-all-struct-inv S =
(*no-strange-atm S* ∧ *cdcl$_W$-M-level-inv S*
∧ (∀ *s* ∈# *learned-clss S*. ¬*tautology s*)
∧ *distinct-cdcl$_W$-state S* ∧ *cdcl$_W$-conflicting S*
∧ *all-decomposition-implies-m* (*init-clss S*) (*get-all-marked-decomposition* (*trail S*))
∧ *cdcl$_W$-learned-clause S*)

lemma *cdcl$_W$-all-struct-inv-inv*:
assumes *cdcl$_W$ S S'* **and** *cdcl$_W$-all-struct-inv S*
shows *cdcl$_W$-all-struct-inv S'*
unfolding *cdcl$_W$-all-struct-inv-def*
proof (*intro HOL.conjI*)
show *no-strange-atm S'*
using *cdcl$_W$-all-inv*[*OF assms*(*1*)] *assms*(*2*) **unfolding** *cdcl$_W$-all-struct-inv-def* **by** *auto*
show *cdcl$_W$-M-level-inv S'*
using *cdcl$_W$-all-inv*[*OF assms*(*1*)] *assms*(*2*) **unfolding** *cdcl$_W$-all-struct-inv-def* **by** *fast*
show *distinct-cdcl$_W$-state S'*
using *cdcl$_W$-all-inv*[*OF assms*(*1*)] *assms*(*2*) **unfolding** *cdcl$_W$-all-struct-inv-def* **by** *fast*
show *cdcl$_W$-conflicting S'*
using *cdcl$_W$-all-inv*[*OF assms*(*1*)] *assms*(*2*) **unfolding** *cdcl$_W$-all-struct-inv-def* **by** *fast*
show *all-decomposition-implies-m* (*init-clss S'*) (*get-all-marked-decomposition* (*trail S'*))
using *cdcl$_W$-all-inv*[*OF assms*(*1*)] *assms*(*2*) **unfolding** *cdcl$_W$-all-struct-inv-def* **by** *fast*
show *cdcl$_W$-learned-clause S'*
using *cdcl$_W$-all-inv*[*OF assms*(*1*)] *assms*(*2*) **unfolding** *cdcl$_W$-all-struct-inv-def* **by** *fast*

show ∀ *s*∈#*learned-clss S'*. ¬ *tautology s*
using *assms*(*1*)[*THEN learned-clss-are-not-tautologies*] *assms*(*2*)
unfolding *cdcl$_W$-all-struct-inv-def* **by** *fast*
qed

lemma *rtranclp-cdcl$_W$-all-struct-inv-inv*:
assumes *cdcl$_W$** S S'* **and** *cdcl$_W$-all-struct-inv S*

shows *cdcl$_W$-all-struct-inv S'*
using *assms* **by** *induction* (*auto intro*: *cdcl$_W$-all-struct-inv-inv*)

lemma *cdcl$_W$-stgy-cdcl$_W$-all-struct-inv*:
cdcl$_W$-stgy S T $\Longrightarrow$ *cdcl$_W$-all-struct-inv S* $\Longrightarrow$ *cdcl$_W$-all-struct-inv T*
by (*meson cdcl$_W$-stgy-tranclp-cdcl$_W$ rtranclp-cdcl$_W$-all-struct-inv-inv rtranclp-unfold*)

lemma *rtranclp-cdcl$_W$-stgy-cdcl$_W$-all-struct-inv*:
cdcl$_W$-stgy** S T $\Longrightarrow$ *cdcl$_W$-all-struct-inv S* $\Longrightarrow$ *cdcl$_W$-all-struct-inv T*
by (*induction rule*: *rtranclp-induct*) (*auto intro*: *cdcl$_W$-stgy-cdcl$_W$-all-struct-inv*)

5.8 No Relearning of a clause

lemma *cdcl$_W$-o-new-clause-learned-is-backtrack-step*:
assumes *learned*: *D* $\in\#$ *learned-clss T* **and**
new: *D* $\notin\#$ *learned-clss S* **and**
cdcl$_W$: *cdcl$_W$-o S T* **and**
lev: *cdcl$_W$-M-level-inv S*
shows *backtrack S T* $\wedge$ *conflicting S* = *Some D*
using *cdcl$_W$ lev learned new*
proof (*induction rule*: *cdcl$_W$-o-induct-lev2*)
case (*backtrack K i M1 M2 L C T*) **note** *decomp* =*this*(*1*) **and** *undef* = *this*(*6*) **and** *T* = *this*(*7*)
and
D-T = *this*(*9*) **and** *D-S* = *this*(*10*)
then have *D* = *C* + {#*L*#}
using *not-gr0 lev* **by** (*auto simp*: *cdcl$_W$-M-level-inv-decomp*)
then show *?case*
using *T backtrack.hyps*(*1*−*5*) *backtrack.intros* **by** *auto*
qed *auto*

lemma *cdcl$_W$-cp-new-clause-learned-has-backtrack-step*:
assumes *learned*: *D* $\in\#$ *learned-clss T* **and**
new: *D* $\notin\#$ *learned-clss S* **and**
cdcl$_W$: *cdcl$_W$-stgy S T* **and**
lev: *cdcl$_W$-M-level-inv S*
shows $\exists$ *S'*. *backtrack S S'* $\wedge$ *cdcl$_W$-stgy** S' T* $\wedge$ *conflicting S* = *Some D*
using *cdcl$_W$ learned new*
proof (*induction rule*: *cdcl$_W$-stgy.induct*)
case (*conflict' S'*)
then show *?case*
unfolding *full1-def* **by** (*metis* (*mono-tags*, *lifting*) *rtranclp-cdcl$_W$-cp-learned-clause-inv tranclp-into-rtranclp*)
next
case (*other' S' S''*)
then have *D* $\in\#$ *learned-clss S'*
unfolding *full-def* **by** (*auto dest*: *rtranclp-cdcl$_W$-cp-learned-clause-inv*)
then show *?case*
using *cdcl$_W$-o-new-clause-learned-is-backtrack-step*[*OF* - ‹*D* $\notin\#$ *learned-clss S*› ‹*cdcl$_W$-o S S'*›]
‹*full cdcl$_W$-cp S' S''*› *lev* **by** (*metis cdcl$_W$-stgy.conflict' full-unfold r-into-rtranclp rtranclp.rtrancl-refl*)
qed

lemma *rtranclp-cdcl$_W$-cp-new-clause-learned-has-backtrack-step*:
assumes *learned*: *D* $\in\#$ *learned-clss T* **and**
new: *D* $\notin\#$ *learned-clss S* **and**
cdcl$_W$: *cdcl$_W$-stgy** S T* **and**

lev: *cdcl*$_W$*-M-level-inv S*
shows $\exists S'\ S''$. *cdcl*$_W$*-stgy*** *S S'* $\wedge$ *backtrack S' S''* $\wedge$ *conflicting S'* = *Some D* $\wedge$
cdcl$_W$*-stgy*** *S'' T*
using *cdcl*$_W$ *learned new*
proof (*induction rule*: *rtranclp-induct*)
case *base*
then show *?case* **by** *blast*
next
case (*step T U*) **note** *st* =*this*(*1*) **and** *o* = *this*(*2*) **and** *IH* = *this*(*3*) **and**
D-U = *this*(*4*) **and** *D-S* = *this*(*5*)
show *?case*
proof (*cases D* ∈# *learned-clss T*)
case *True*
then obtain *S' S''* **where**
st': *cdcl*$_W$*-stgy*** *S S'* **and**
bt: *backtrack S' S''* **and**
confl: *conflicting S'* = *Some D* **and**
st'': *cdcl*$_W$*-stgy*** *S'' T*
using *IH D-S* **by** *metis*
then show *?thesis* **using** *o* **by** (*meson rtranclp.simps*)
next
case *False*
have *cdcl*$_W$*-M-level-inv T*
using *lev rtranclp-cdcl*$_W$*-stgy-consistent-inv st* **by** *blast*
then obtain *S'* **where**
bt: *backtrack T S'* **and**
st': *cdcl*$_W$*-stgy*** *S' U* **and**
confl: *conflicting T* = *Some D*
using *cdcl*$_W$*-cp-new-clause-learned-has-backtrack-step*[*OF D-U False o*]
by *metis*
then have *cdcl*$_W$*-stgy*** *S T* **and**
backtrack T S' **and**
conflicting T = *Some D* **and**
cdcl$_W$*-stgy*** *S' U*
using *o st* **by** *auto*
then show *?thesis* **by** *blast*
qed
qed

lemma *propagate-no-more-Marked-lit*:
assumes *propagate S S'*
shows *Marked K i* ∈ *set* (*trail S*) ⟷ *Marked K i* ∈ *set* (*trail S'*)
using *assms* **by** *auto*

lemma *conflict-no-more-Marked-lit*:
assumes *conflict S S'*
shows *Marked K i* ∈ *set* (*trail S*) ⟷ *Marked K i* ∈ *set* (*trail S'*)
using *assms* **by** *auto*

lemma *cdcl*$_W$*-cp-no-more-Marked-lit*:
assumes *cdcl*$_W$*-cp S S'*
shows *Marked K i* ∈ *set* (*trail S*) ⟷ *Marked K i* ∈ *set* (*trail S'*)
using *assms* **apply** (*induct rule*: *cdcl*$_W$*-cp.induct*)
using *conflict-no-more-Marked-lit propagate-no-more-Marked-lit* **by** *auto*

lemma *rtranclp-cdcl$_W$-cp-no-more-Marked-lit*:
  assumes *cdcl$_W$-cp** S S'*
  shows *Marked K i* $\in$ *set* (*trail S*) $\longleftrightarrow$ *Marked K i* $\in$ *set* (*trail S'*)
  using *assms* **apply** (*induct rule*: *rtranclp-induct*)
  using *cdcl$_W$-cp-no-more-Marked-lit* **by** *blast+*

lemma *cdcl$_W$-o-no-more-Marked-lit*:
  assumes *cdcl$_W$-o S S'* **and** *cdcl$_W$-M-level-inv S* **and** $\neg$*decide S S'*
  shows *Marked K i* $\in$ *set* (*trail S'*) $\longrightarrow$ *Marked K i* $\in$ *set* (*trail S*)
  using *assms*
proof (*induct rule*: *cdcl$_W$-o-induct-lev2*)
  case *backtrack* **note** *decomp* = *this*(*1*) **and** *undef* = *this*(*6*) **and** *T* =*this*(*7*) **and** *lev* = *this*(*8*)
  then show *?case*
    by (*auto simp*: *cdcl$_W$-M-level-inv-decomp*)
next
  case (*decide L T*)
  then show *?case* **by** *blast*
qed *auto*

lemma *cdcl$_W$-new-marked-at-beginning-is-decide*:
  assumes *cdcl$_W$-stgy S S'* **and**
  lev: *cdcl$_W$-M-level-inv S* **and**
  trail S' = *M'* @ *Marked L i* # *M* **and**
  trail S = *M*
  shows $\exists$ *T*. *decide S T* $\wedge$ *no-step cdcl$_W$-cp S*
  using *assms*
proof (*induct rule*: *cdcl$_W$-stgy.induct*)
  case (*conflict' S'*) **note** *st* =*this*(*1*) **and** *no-dup* = *this*(*2*) **and** *S'* = *this*(*3*) **and** *S* = *this*(*4*)
  have *cdcl$_W$-M-level-inv S'*
    using *full1-cdcl$_W$-cp-consistent-inv no-dup st* **by** *blast*
  then have *Marked L i* $\in$ *set* (*trail S'*) **and** *Marked L i* $\notin$ *set* (*trail S*)
    using *no-dup* **unfolding** *S S' cdcl$_W$-M-level-inv-def* **by** (*auto simp add*: *rev-image-eqI*)
  then have *False*
    using *st rtranclp-cdcl$_W$-cp-no-more-Marked-lit*[*of S S'*]
    unfolding *full1-def rtranclp-unfold* **by** *blast*
  then show *?case* **by** *fast*
next
  case (*other' T U*) **note** *o* = *this*(*1*) **and** *ns* = *this*(*2*) **and** *st* = *this*(*3*) **and** *no-dup* = *this*(*4*) **and**
    S' = *this*(*5*) **and** *S* = *this*(*6*)
  have *cdcl$_W$-M-level-inv U*
    by (*metis* (*full-types*) *lev cdcl$_W$.simps cdcl$_W$-consistent-inv full-def o*
      other'.hyps(*3*) *rtranclp-cdcl$_W$-cp-consistent-inv*)
  then have *Marked L i* $\in$ *set* (*trail U*) **and** *Marked L i* $\notin$ *set* (*trail S*)
    using *no-dup* **unfolding** *S S' cdcl$_W$-M-level-inv-def* **by** (*auto simp add*: *rev-image-eqI*)
  then have *Marked L i* $\in$ *set* (*trail T*)
    using *st rtranclp-cdcl$_W$-cp-no-more-Marked-lit* **unfolding** *full-def* **by** *blast*
  then show *?case*
    using *cdcl$_W$-o-no-more-Marked-lit*[*OF o*] ‹*Marked L i* $\notin$ *set* (*trail S*)› *ns lev* **by** *meson*
qed

lemma *cdcl$_W$-o-is-decide*:
  assumes *cdcl$_W$-o S' T* **and** *cdcl$_W$-M-level-inv S'*
  trail T = *drop* (*length M_0*) *M'* @ *Marked L i* # *H* @ *M***and**
  $\neg$ ($\exists$ *M'*. *trail S'* = *M'* @ *Marked L i* # *H* @ *M*)
  shows *decide S' T*

using *assms*
proof (*induction rule*:*cdcl*$_W$*-o-induct-lev2*)
case (*backtrack K i M1 M2 L D*)
then obtain *c* **where** *trail S′ = c @ M2 @ Marked K (Suc i) # M1*
by *auto*
then show *?case*
using *backtrack* **by** (*cases drop (length* M_0*) M′*) (*auto simp*: *cdcl*$_W$*-M-level-inv-def*)
next
case *decide*
show *?case* **using** *decide-rule*[*of S′*] *decide*(*1−4*) **by** *auto*
qed *auto*

lemma *rtranclp-cdcl*$_W$*-new-marked-at-beginning-is-decide*:
assumes *cdcl*$_W$*-stgy*** *R U* **and**
trail U = M′ @ Marked L i # H @ M **and**
trail R = M **and**
cdcl$_W$*-M-level-inv R*
shows
∃*S T T′*. *cdcl*$_W$*-stgy*** *R S* ∧ *decide S T* ∧ *cdcl*$_W$*-stgy*** *T U* ∧ *cdcl*$_W$*-stgy*** *S U* ∧
no-step cdcl$_W$*-cp S* ∧ *trail T = Marked L i # H @ M* ∧ *trail S = H @ M* ∧ *cdcl*$_W$*-stgy S T′* ∧
cdcl$_W$*-stgy*** *T′ U*
using *assms*
proof (*induct arbitrary*: *M H M′ i rule*: *rtranclp-induct*)
case *base*
then show *?case* **by** *auto*
next
case (*step T U*) **note** *st = this*(*1*) **and** *IH = this*(*3*) **and** *s = this*(*2*) **and**
U = this(*4*) **and** *S = this*(*5*) **and** *lev = this*(*6*)
show *?case*
proof (*cases* ∃*M′*. *trail T = M′ @ Marked L i # H @ M*)
case *False*
with *s* **show** *?thesis* **using** *U s st S*
proof *induction*
case (*conflict′ W*) **note** *cp = this*(*1*) **and** *nd = this*(*2*) **and** *W = this*(*3*)
then obtain M_0 **where** *trail W =* M_0 *@ trail T* **and** *nmarked*: ∀ *l*∈*set* M_0. ¬ *is-marked l*
using *rtranclp-cdcl*$_W$*-cp-dropWhile-trail* **unfolding** *full1-def rtranclp-unfold* **by** *meson*
then have *MV*: *M′ @ Marked L i # H @ M =* M_0 *@ trail T* **unfolding** *W* **by** *simp*
then have *V*: *trail T = drop* (*length* M_0) (*M′ @ Marked L i # H @ M*)
by *auto*
have *takeWhile* (*Not o is-marked*) *M′ =* M_0 *@ takeWhile* (*Not* ∘ *is-marked*) (*trail T*)
using *arg-cong*[*OF MV*, *of takeWhile* (*Not o is-marked*)] *nmarked*
by (*simp add*: *takeWhile-tail*)
from *arg-cong*[*OF this*, *of length*] **have** *length* M_0 ≤ *length M′*
unfolding *length-append* **by** (*metis* (*no-types*, *lifting*) *Nat.le-trans le-add1*
length-takeWhile-le)
then have *False* **using** *nd V* **by** *auto*
then show *?case* **by** *fast*
next
case (*other′ T′ U*) **note** *o = this*(*1*) **and** *ns =this*(*2*) **and** *cp = this*(*3*) **and** *nd = this*(*4*)
and *U = this*(*5*) **and** *st = this*(*6*)
obtain M_0 **where** *trail U =* M_0 *@ trail T′* **and** *nmarked*: ∀ *l*∈*set* M_0. ¬ *is-marked l*
using *rtranclp-cdcl*$_W$*-cp-dropWhile-trail cp* **unfolding** *full-def* **by** *meson*
then have *MV*: *M′ @ Marked L i # H @ M =* M_0 *@ trail T′* **unfolding** *U* **by** *simp*
then have *V*: *trail T′ = drop* (*length* M_0) (*M′ @ Marked L i # H @ M*)
by *auto*

have *takeWhile* (*Not o is-marked*) $M' = M_0$ @ *takeWhile* (*Not* ∘ *is-marked*) (*trail* T')
using *arg-cong*[*OF MV*, *of takeWhile* (*Not o is-marked*)] *nmarked*
by (*simp add*: *takeWhile-tail*)
from *arg-cong*[*OF this*, *of length*] **have** *length* $M_0 \leq$ *length* M'
unfolding *length-append* **by** (*metis* (*no-types*, *lifting*) *Nat.le-trans le-add1 length-takeWhile-le*)
then have *tr-*T': *trail* $T' =$ *drop* (*length* M_0) M' @ *Marked L i* # *H* @ *M* **using** *V* **by** *auto*
then have *LT*$'$: *Marked L i* ∈ *set* (*trail* T') **by** *auto*
moreover
have *cdcl*$_W$*-M-level-inv T*
using *lev rtranclp-cdcl*$_W$*-stgy-consistent-inv step.hyps*(*1*) **by** *blast*
then have *decide T* T' **using** *o nd tr-*T' *cdcl*$_W$*-o-is-decide* **by** *metis*
ultimately have *decide T* T' **using** *cdcl*$_W$*-o-no-more-Marked-lit*[*OF o*] **by** *blast*
then have *1*: *cdcl*$_W$*-stgy*** *R T* **and** *2*: *decide T* T' **and** *3*: *cdcl*$_W$*-stgy*** T' *U*
using *st other*$'$*.prems*(*4*)
by (*metis cdcl*$_W$*-stgy.conflict*$'$ *cp full-unfold r-into-rtranclp rtranclp.rtrancl-refl*)+
have [*simp*]: *drop* (*length* M_0) M' = []
using ‹*decide T* T'› ‹*Marked L i* ∈ *set* (*trail* T')› *nd tr-*T'
by (*auto simp add*: *Cons-eq-append-conv*)
have T': *drop* (*length* M_0) M' @ *Marked L i* # *H* @ *M* = *Marked L i* # *trail T*
using ‹*decide T* T'› ‹*Marked L i* ∈ *set* (*trail* T')› *nd tr-*T'
by *auto*
have *trail* T' = *Marked L i* # *trail T*
using ‹*decide T* T'› ‹*Marked L i* ∈ *set* (*trail* T')› *tr-*T'
by *auto*
then have *5*: *trail* T' = *Marked L i* # *H* @ *M*
using *append.simps*(*1*) *list.sel*(*3*) *local.other*$'$(*5*) *tl-append2* **by** (*simp add*: *tr-*T')
have *6*: *trail T* = *H* @ *M*
by (*metis* (*no-types*) ‹*trail* T' = *Marked L i* # *trail T*›
‹*trail* T' = *drop* (*length* M_0) M' @ *Marked L i* # *H* @ *M*› *append-Nil list.sel*(*3*) *nd tl-append2*)
have *7*: *cdcl*$_W$*-stgy*** *T U* **using** *other*$'$*.prems*(*4*) *st* **by** *auto*
have *8*: *cdcl*$_W$*-stgy T U cdcl*$_W$*-stgy*** *U U*
using *cdcl*$_W$*-stgy.other*$'$[*OF other*$'$(*1*−*3*)] **by** *simp-all*
show *?case* **apply** (*rule exI*[*of - T*], *rule exI*[*of -* T'], *rule exI*[*of - U*])
using *ns 1 2 3 5 6 7 8* **by** *fast*
qed
next
case *True*
then obtain M' **where** *T*: *trail T* = M' @ *Marked L i* # *H* @ *M* **by** *metis*
from *IH*[*OF this S lev*] **obtain** S' S'' S''' **where**
1: *cdcl*$_W$*-stgy*** *R* S' **and**
2: *decide* S' S'' **and**
3: *cdcl*$_W$*-stgy*** S'' *T* **and**
4: *no-step cdcl*$_W$*-cp* S' **and**
6: *trail* S'' = *Marked L i* # *H* @ *M* **and**
7: *trail* S' = *H* @ *M* **and**
8: *cdcl*$_W$*-stgy*** S' *T* **and**
9: *cdcl*$_W$*-stgy* S' S''' **and**
10: *cdcl*$_W$*-stgy*** S''' *T*
by *blast*
have *cdcl*$_W$*-stgy*** S'' *U* **using** *s* ‹*cdcl*$_W$*-stgy*** S'' *T* › **by** *auto*
moreover have *cdcl*$_W$*-stgy*** S' *U* **using** *8 s* **by** *auto*
moreover have *cdcl*$_W$*-stgy*** S''' *U* **using** *10 s* **by** *auto*
ultimately show *?thesis* **apply** − **apply** (*rule exI*[*of -* S'], *rule exI*[*of -* S''])

using *1 2 4 6 7 8 9* **by** *blast*
qed
qed

lemma *rtranclp-cdcl$_W$-new-marked-at-beginning-is-decide$'$*:
assumes *cdcl$_W$-stgy** R U* **and**
trail U = M$'$ @ Marked L i # H @ M **and**
trail R = M **and**
cdcl$_W$-M-level-inv R
shows $\exists$ *y y$'$. cdcl$_W$-stgy** R y* $\wedge$ *cdcl$_W$-stgy y y$'$* $\wedge$ $\neg$ ($\exists$ *c. trail y = c @ Marked L i # H @ M*)
$\wedge$ (λ*a b. cdcl$_W$-stgy a b* $\wedge$ ($\exists$ *c. trail a = c @ Marked L i # H @ M*))** *y$'$ U*
proof –
fix *T$'$*
obtain *S$'$ T T$'$* **where**
st: *cdcl$_W$-stgy** R S$'$* **and**
decide S$'$ T **and**
TU: *cdcl$_W$-stgy** T U* **and**
no-step cdcl$_W$-cp S$'$ **and**
trT: *trail T = Marked L i # H @ M* **and**
trS$'$: *trail S$'$ = H @ M* **and**
S$'$U: *cdcl$_W$-stgy** S$'$ U* **and**
S$'$T$'$: *cdcl$_W$-stgy S$'$ T$'$* **and**
T$'$U: *cdcl$_W$-stgy** T$'$ U*
using *rtranclp-cdcl$_W$-new-marked-at-beginning-is-decide*[*OF assms*] **by** *blast*
have *n*: $\neg$ ($\exists$ *c. trail S$'$ = c @ Marked L i # H @ M*) **using** *trS$'$* **by** *auto*
show *?thesis*
using *rtranclp-trans*[*OF st*] *rtranclp-exists-last-with-prop*[*of cdcl$_W$-stgy S$'$ T$'$ -*
λ*a -.* $\neg$($\exists$ *c. trail a = c @ Marked L i # H @ M*), *OF S$'$T$'$ T$'$U n*]
by *meson*
qed

lemma *beginning-not-marked-invert*:
assumes *A*: *M @ A = M$'$ @ Marked K i # H* **and**
nm: $\forall$ *m*$\in$*set M*. $\neg$*is-marked m*
shows $\exists$ *M. A = M @ Marked K i # H*
proof –
have *A = drop* (*length M*) (*M$'$ @ Marked K i # H*)
using *arg-cong*[*OF A*, *of drop* (*length M*)] **by** *auto*
moreover have *drop* (*length M*) (*M$'$ @ Marked K i # H*) = *drop* (*length M*) *M$'$ @ Marked K i # H*
using *nm* **by** (*metis* (*no-types*, *lifting*) *A drop-Cons$'$ drop-append ann-literal.disc*(*1*) *not-gr0*
nth-append nth-append-length nth-mem zero-less-diff)
finally show *?thesis* **by** *fast*
qed

lemma *cdcl$_W$-stgy-trail-has-new-marked-is-decide-step*:
assumes *cdcl$_W$-stgy S T*
$\neg$ ($\exists$ *c. trail S = c @ Marked L i # H @ M*) **and**
(λ*a b. cdcl$_W$-stgy a b* $\wedge$ ($\exists$ *c. trail a = c @ Marked L i # H @ M*))** *T U* **and**
$\exists$ *M$'$. trail U = M$'$ @ Marked L i # H @ M* **and**
lev: *cdcl$_W$-M-level-inv S*
shows $\exists$ *S$'$. decide S S$'$* $\wedge$ *full cdcl$_W$-cp S$'$ T* $\wedge$ *no-step cdcl$_W$-cp S*
using *assms*(*3*,*1*,*2*,*4*,*5*)
proof *induction*
case (*step T U*)
then show *?case* **by** *fastforce*

next
  case *base*
  then show *?case*
    proof (*induction rule*: $cdcl_W$*-stgy.induct*)
      case (*conflict′ T*) **note** *cp* = *this*(*1*) **and** *nd* = *this*(*2*) **and** *M′* =*this*(*3*) **and** *no-dup* = *this*(*3*)
      then obtain *M′* **where** *M′*: *trail T* = *M′* @ *Marked L i* # *H* @ *M* **by** *metis*
      obtain *M′′* **where** *M′′*: *trail T* = *M′′* @ *trail S* **and** *nm*: ∀ *m*∈ *set M′′*. ¬*is-marked m*
        using *cp* **unfolding** *full1-def*
        by (*metis rtranclp-*$cdcl_W$*-cp-dropWhile-trail′ tranclp-into-rtranclp*)
      have *False*
        using *beginning-not-marked-invert*[*of M′′ trail S M′ L i H* @ *M*] *M′ nm nd* **unfolding** *M′′*
        by *fast*
      then show *?case* **by** *fast*
    next
      case (*other′ T U′*) **note** *o* = *this*(*1*) **and** *ns* = *this*(*2*) **and** *cp* = *this*(*3*) **and** *nd* = *this*(*4*)
        and *trU′* = *this*(*5*)
      have $cdcl_W$*-cp*** *T U′* **using** *cp* **unfolding** *full-def* **by** *blast*
      from *rtranclp-*$cdcl_W$*-cp-dropWhile-trail*[*OF this*]
      have ∃ *M′*. *trail T* = *M′* @ *Marked L i* # *H* @ *M*
        using *trU′ beginning-not-marked-invert*[*of - trail T - L i H* @ *M*] **by** *metis*
      then obtain *M′* **where** *M′*: *trail T* = *M′* @ *Marked L i* # *H* @ *M*
        by *auto*
      with *o lev nd cp ns*
      show *?case*
        proof (*induction rule*: $cdcl_W$*-o-induct-lev2*)
          case (*decide L*) **note** *dec* = *this*(*1*) **and** *cp* = *this*(*5*) **and** *ns* = *this*(*4*)
          then have *decide S* (*cons-trail* (*Marked L* (*backtrack-lvl S* +*1*)) (*incr-lvl S*))
            using *decide.hyps decide.intros*[*of S*] **by** *force*
          then show *?case* **using** *cp decide.prems* **by** (*meson decide-state-eq-compatible ns state-eq-ref*
            state-eq-sym)
        next
          case (*backtrack K j M1 M2 L′ D T*) **note** *decomp* = *this*(*1*) **and** *cp* = *this*(*3*)
            and *undef* = *this*(*6*) **and** *T* = *this*(*7*) **and** *trT* = *this*(*12*) **and** *ns* = *this*(*4*)
          obtain *MS3* **where** *MS3*: *trail S* = *MS3* @ *M2* @ *Marked K* (*Suc j*) # *M1*
            using *get-all-marked-decomposition-exists-prepend*[*OF decomp*] **by** *metis*
          have *tl* (*M′* @ *Marked L i* # *H* @ *M*) = *tl M′* @ *Marked L i* # *H* @ *M*
            using *lev trT T lev undef decomp* **by** (*cases M′*) (*auto simp*: $cdcl_W$*-M-level-inv-decomp*)
          then have *M′′*: *M1* = *tl M′* @ *Marked L i* # *H* @ *M*
            using *arg-cong*[*OF trT*[*simplified*], *of tl*] *T decomp undef lev*
            by (*simp add*: $cdcl_W$*-M-level-inv-decomp*)
          have *False* **using** *nd MS3 T undef decomp* **unfolding** *M′′* **by** *auto*
          then show *?case* **by** *fast*
        qed *auto*
      qed
qed

lemma *rtranclp-*$cdcl_W$*-stgy-with-trail-end-has-trail-end*:
  assumes (λ*a b*. $cdcl_W$*-stgy a b* ∧ (∃ *c*. *trail a* = *c* @ *Marked L i* # *H* @ *M*))** *T U* **and**
  ∃ *M′*. *trail U* = *M′* @ *Marked L i* # *H* @ *M*
  shows ∃ *M′*. *trail T* = *M′* @ *Marked L i* # *H* @ *M*
  using *assms* **by** (*induction rule*: *rtranclp-induct*) *auto*

lemma $cdcl_W$*-o-cannot-learn*:
  assumes
    $cdcl_W$*-o y z* **and**

lev: *cdcl*$_W$*-M-level-inv y* **and**
trM: *trail y* = *c* @ *Marked Kh i* # *H* **and**
DL: *D* + {#*L*#} ∉# *learned-clss y* **and**
DH: *atms-of D* ⊆ *atm-of* ‘*lits-of H* **and**
LH: *atm-of L* ∉ *atm-of* ‘*lits-of H* **and**
learned: ∀ *T*. *conflicting y* = *Some T* ⟶ *trail y* ⊨*as CNot T* **and**
z: *trail z* = *c'* @ *Marked Kh i* # *H*
shows *D* + {#*L*#} ∉# *learned-clss z*
using *assms*(*1*−*2*) *trM DL DH LH learned z*
proof (*induction rule*: *cdcl*$_W$*-o-induct-lev2*)
case (*backtrack K j M1 M2 L' D' T*) **note** *decomp* = *this*(*1*) **and** *confl* = *this*(*3*) **and** *levD* = *this*(*5*)
and *undef* = *this*(*6*) **and** *T* = *this*(*7*)
obtain *M3* **where** *M3*: *trail y* = *M3* @ *M2* @ *Marked K* (*Suc j*) # *M1*
using *decomp get-all-marked-decomposition-exists-prepend* **by** *metis*
have *M*: *trail y* = *c* @ *Marked Kh i* # *H* **using** *trM* **by** *simp*
have *H*: *get-all-levels-of-marked* (*trail y*) = *rev* [*1*..<*1* + *backtrack-lvl y*]
using *lev* **unfolding** *cdcl*$_W$*-M-level-inv-def* **by** *auto*
have *c'* @ *Marked Kh i* # *H* = *Propagated L'* (*D'* + {#*L'*#}) # *trail* (*reduce-trail-to M1 y*)
using *backtrack.prems*(*6*) *decomp undef T lev* **by** (*force simp*: *cdcl*$_W$*-M-level-inv-def*)
then obtain *d* **where** *d*: *M1* = *d* @ *Marked Kh i* # *H*
by (*metis* (*no-types*) *decomp in-get-all-marked-decomposition-trail-update-trail list.inject*
list.sel(*3*) *ann-literal.distinct*(*1*) *self-append-conv2 tl-append2*)
have *i* ∈ *set* (*get-all-levels-of-marked* (*M3* @ *M2* @ *Marked K* (*Suc j*) # *d* @ *Marked Kh i* # *H*))
by *auto*
then have *i* > *0* **unfolding** *H*[*unfolded M3 d*] **by** *auto*
show *?case*
proof
assume *D* + {#*L*#} ∈# *learned-clss T*
then have *DLD'*: *D* + {#*L*#} = *D'* + {#*L'*#}
using *DL T neq0-conv undef decomp lev* **by** (*fastforce simp*: *cdcl*$_W$*-M-level-inv-def*)
have *L-cKh*: *atm-of L* ∈ *atm-of* ‘*lits-of* (*c* @ [*Marked Kh i*])
using *LH learned M DLD'*[*symmetric*] *confl* **by** (*fastforce simp add*: *image-iff*)
have *get-all-levels-of-marked* (*M3* @ *M2* @ *Marked K* (*j* + *1*) # *M1*)
= *rev* [*1*..<*1* + *backtrack-lvl y*]
using *lev* **unfolding** *cdcl*$_W$*-M-level-inv-def M3* **by** *auto*
from *arg-cong*[*OF this*, *of* λ*a*. (*Suc j*) ∈ *set a*] **have** *backtrack-lvl y* ≥ *j* **by** *auto*

have *DD'*[*simp*]: *D* = *D'*
proof (*rule ccontr*)
assume *D* ≠ *D'*
then have *L'* ∈# *D* **using** *DLD'* **by** (*metis add.left-neutral count-single count-union*
diff-union-cancelR neq0-conv union-single-eq-member)
then have *get-level* (*trail y*) *L'* ≤ *get-maximum-level* (*trail y*) *D*
using *get-maximum-level-ge-get-level* **by** *blast*
moreover {
have *get-maximum-level* (*trail y*) *D* = *get-maximum-level H D*
using *DH* **unfolding** *M* **by** (*simp add*: *get-maximum-level-skip-beginning*)
moreover
have *get-all-levels-of-marked* (*trail y*) = *rev* [*1*..<*1* + *backtrack-lvl y*]
using *lev* **unfolding** *cdcl*$_W$*-M-level-inv-def* **by** *auto*
then have *get-all-levels-of-marked H* = *rev* [*1*..< *i*]
unfolding *M* **by** (*auto dest*: *append-cons-eq-upt-length-i*
simp add: *rev-swap*[*symmetric*])
then have *get-maximum-possible-level H* < *i*
using *get-maximum-possible-level-max-get-all-levels-of-marked*[*of H*] ‹*i* > *0*› **by** *auto*

ultimately have *get-maximum-level* (*trail y*) *D* < *i*
by (*metis* (*full-types*) *dual-order.strict-trans nat-neq-iff not-le*
get-maximum-possible-level-ge-get-maximum-level) **}**
moreover
have *L* ∈# *D'*
by (*metis DLD'* ‹*D* ≠ *D'*› *add.left-neutral count-single count-union diff-union-cancelR*
neq0-conv union-single-eq-member)
then have *get-maximum-level* (*trail y*) *D'* ≥ *get-level* (*trail y*) *L*
using *get-maximum-level-ge-get-level* **by** *blast*
moreover {
have *get-all-levels-of-marked* (*c* @ [*Marked Kh i*]) = *rev* [*i*..< *backtrack-lvl y*+*1*]
using *append-cons-eq-upt-length-i-end*[*of rev* (*get-all-levels-of-marked H*) *i*
rev (*get-all-levels-of-marked c*) *Suc 0 Suc* (*backtrack-lvl y*)] *H*
unfolding *M* **apply** (*auto simp add*: *rev-swap*[*symmetric*])
by (*metis* (*no-types*, *hide-lams*) *Nil-is-append-conv Suc-le-eq less-Suc-eq list.sel*(*1*)
rev.simps(*2*) *rev-rev-ident upt-Suc upt-rec*)
have *get-level* (*trail y*) *L* = *get-level* (*c* @ [*Marked Kh i*]) *L*
using *L-cKh LH* **unfolding** *M* **by** *simp*
have *get-level* (*c* @ [*Marked Kh i*]) *L* ≥ *i*
using *L-cKh*
‹*get-all-levels-of-marked* (*c* @ [*Marked Kh i*]) = *rev* [*i*..<*backtrack-lvl y* + *1*]›
backtrack.hyps(*2*) *calculation*(*1*,*2*) **by** *auto*
then have *get-level* (*trail y*) *L* ≥ *i*
using *M* ‹*get-level* (*trail y*) *L* = *get-level* (*c* @ [*Marked Kh i*]) *L*› **by** *auto* **}**
moreover have *get-maximum-level* (*trail y*) *D'* < *get-level* (*trail y*) *L*
using ‹*j* ≤ *backtrack-lvl y*› *backtrack.hyps*(*2*,*5*) *calculation*(*1*−*4*) **by** *linarith*
ultimately show *False* **using** *backtrack.hyps*(*4*) **by** *linarith*
qed
then have *LL'*: *L* = *L'* **using** *DLD'* **by** *auto*
have *nd*: *no-dup* (*trail y*) **using** *lev* **unfolding** $cdcl_W$-*M-level-inv-def* **by** *auto*

{ assume *D*: *D'* = {#}
then have *j*: *j* = *0* **using** *levD* **by** *auto*
have ∀ *m* ∈ *set M1*. ¬*is-marked m*
using *H* **unfolding** *M3 j*
by (*auto simp add*: *rev-swap*[*symmetric*] *get-all-levels-of-marked-no-marked*
dest!: *append-cons-eq-upt-length-i*)
then have *False* **using** *d* **by** *auto*
}
moreover {
assume *D*[*simp*]: *D'* ≠ {#}
have *i* ≤ *j*
using *H* **unfolding** *M3 d* **by** (*auto simp add*: *rev-swap*[*symmetric*]
dest: *upt-decomp-lt*)
have *j* > *0* **apply** (*rule ccontr*)
using *H* ‹ *i* > *0*› **unfolding** *M3 d*
by (*auto simp add*: *rev-swap*[*symmetric*] *dest*!: *upt-decomp-lt*)
obtain *L''* **where**
L''∈#*D'* **and**
L''D': *get-level* (*trail y*) *L''* = *get-maximum-level* (*trail y*) *D'*
using *get-maximum-level-exists-lit-of-max-level*[*OF D*, *of trail y*] **by** *auto*
have *L''M*: *atm-of L''* ∈ *atm-of* ‘ *lits-of* (*trail y*)
using *get-rev-level-ge-0-atm-of-in*[*of 0 rev* (*trail y*) *L''*] ‹*j*>*0*› *levD L''D'* **by** *auto*
then have *L''* ∈ *lits-of* (*Marked Kh i* # *d*)
proof −

{
assume $L''H$: *atm-of* $L'' \in$ *atm-of* ‘ *lits-of* H
have *get-all-levels-of-marked* H = *rev* [*1*..<*i*]
using H **unfolding** M
by (*auto simp add*: *rev-swap*[*symmetric*] *dest*!: *append-cons-eq-upt-length-i*)
moreover have *get-level* (*trail* y) L'' = *get-level* H L''
using $L''H$ **unfolding** M **by** *simp*
ultimately have *False*
using *levD* ‹$j>0$› *get-rev-level-in-levels-of-marked*[*of rev* H *0* L''] ‹$i \le j$›
unfolding $L''D'$[*symmetric*] *nd* **by** *auto*
}
then show *?thesis*
using DD' DH ‹$L'' \in\#$ D'› *atm-of-lit-in-atms-of contra-subsetD* **by** *metis*
qed
then have *False*
using DH ‹$L''\in\#D'$› *nd* **unfolding** *M3 d*
by (*auto simp add*: *atms-of-def image-iff image-subset-iff lits-of-def*)
}
ultimately show *False* **by** *blast*
qed
qed *auto*

lemma *cdcl*$_W$*-stgy-with-trail-end-has-not-been-learned*:
assumes *cdcl*$_W$*-stgy* y z **and**
cdcl$_W$*-M-level-inv* y **and**
trail y = c @ *Marked Kh* i # H **and**
D + {#L#} $\notin\#$ *learned-clss* y **and**
DH: *atms-of* $D \subseteq$ *atm-of* ‘*lits-of* H **and**
LH: *atm-of* $L \notin$ *atm-of* ‘*lits-of* H **and**
$\forall T$. *conflicting* y = *Some* $T \longrightarrow$ *trail* y $\models$*as CNot* T **and**
trail z = c' @ *Marked Kh* i # H
shows D + {#L#} $\notin\#$ *learned-clss* z
using *assms*
proof *induction*
case *conflict'*
then show *?case*
unfolding *full1-def* **using** *tranclp-cdcl*$_W$*-cp-learned-clause-inv* **by** *auto*
next
case (*other'* T U) **note** o = *this*(*1*) **and** *cp* = *this*(*3*) **and** *lev* = *this*(*4*) **and** *trY* = *this*(*5*) **and**
notin = *this*(*6*) **and** DH = *this*(*7*) **and** LH = *this*(*8*) **and** *confl* = *this*(*9*) **and** *trU* = *this*(*10*)
obtain c' **where** c': *trail* T = c' @ *Marked Kh* i # H
using *cp beginning-not-marked-invert*[*of - trail* T c' *Kh* i H]
rtranclp-cdcl$_W$*-cp-dropWhile-trail*[*of* T U] **unfolding** *trU full-def* **by** *fastforce*
show *?case*
using *cdcl*$_W$*-o-cannot-learn*[*OF o lev trY notin DH LH confl* c']
rtranclp-cdcl$_W$*-cp-learned-clause-inv cp* **unfolding** *full-def* **by** *auto*
qed

lemma *rtranclp-cdcl*$_W$*-stgy-with-trail-end-has-not-been-learned*:
assumes (λa b. *cdcl*$_W$*-stgy* a b $\wedge$ ($\exists c$. *trail* a = c @ *Marked* K i # H @ []))** S z **and**
cdcl$_W$*-all-struct-inv* S **and**
trail S = c @ *Marked* K i # H **and**
D + {#L#} $\notin\#$ *learned-clss* S **and**
DH: *atms-of* $D \subseteq$ *atm-of* ‘*lits-of* H **and**
LH: *atm-of* $L \notin$ *atm-of* ‘*lits-of* H **and**

$\exists c'.$ *trail z* $=$ c' @ *Marked K i* # *H*
shows $D + \{\#L\#\} \notin\#$ *learned-clss z*
using *assms*(*1*−*4*,*7*)
proof (*induction rule*: *rtranclp-induct*)
case *base*
then show *?case* **by** *auto*[*1*]
next
case (*step T U*) **note** *st* = *this*(*1*) **and** *s* = *this*(*2*) **and** *IH* = *this*(*3*)[*OF this*(*4*−*6*)]
and *lev* = *this*(*4*) **and** *trS* = *this*(*5*) **and** *DL-S* = *this*(*6*) **and** *trU* = *this*(*7*)
obtain *c* **where** *c*: *trail T* = *c* @ *Marked K i* # *H* **using** *s* **by** *auto*
obtain c' **where** c': *trail U* = c' @ *Marked K i* # *H* **using** *trU* **by** *blast*
have $cdcl_W{}^{**}$ *S T*
proof −
have $\forall p\ pa.\ \exists s\ sa.\ \forall sb\ sc\ sd\ se.\ (\neg\ p^{**}\ (sb{::}'st)\ sc \lor p\ s\ sa \lor pa^{**}\ sb\ sc)$
$\land (\neg\ pa\ s\ sa \lor \neg\ p^{**}\ sd\ se \lor pa^{**}\ sd\ se)$
by (*metis* (*no-types*) *mono-rtranclp*)
then have $cdcl_W$-$stgy^{**}$ *S T*
using *st* **by** *blast*
then show *?thesis*
using *rtranclp-*$cdcl_W$*-stgy-rtranclp-*$cdcl_W$ **by** *blast*
qed
then have lev': $cdcl_W$*-all-struct-inv T*
using *rtranclp-*$cdcl_W$*-all-struct-inv-inv*[*of S T*] *lev* **by** *auto*
then have $confl'$: $\forall$ *Ta. conflicting T* = *Some Ta* $\longrightarrow$ *trail T* $\models$*as CNot Ta*
unfolding $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-conflicting-def* **by** *blast*
show *?case*
apply (*rule* $cdcl_W$*-stgy-with-trail-end-has-not-been-learned*[*OF - - c - DH LH* $confl'$ c'])
using *s* lev' *IH c* **unfolding** $cdcl_W$*-all-struct-inv-def* **by** *blast+*
qed

lemma $cdcl_W$*-stgy-new-learned-clause*:
assumes $cdcl_W$*-stgy S T* **and**
lev: $cdcl_W$*-M-level-inv S* **and**
$E \notin\#$ *learned-clss S* **and**
$E \in\#$ *learned-clss T*
shows $\exists S'$. *backtrack S* S' $\land$ *conflicting S* = *Some E* $\land$ *full* $cdcl_W$*-cp* S' *T*
using *assms*
proof *induction*
case $conflict'$
then show *?case* **unfolding** *full1-def* **by** (*auto dest*: *tranclp-*$cdcl_W$*-cp-learned-clause-inv*)
next
case ($other'$ *T U*) **note** *o* = *this*(*1*) **and** *cp* = *this*(*3*) **and** *not-yet* = *this*(*5*) **and** *learned* = *this*(*6*)
have $E \in\#$ *learned-clss T*
using *learned cp rtranclp-*$cdcl_W$*-cp-learned-clause-inv* **unfolding** *full-def* **by** *auto*
then have *backtrack S T* **and** *conflicting S* = *Some E*
using $cdcl_W$*-o-new-clause-learned-is-backtrack-step*[*OF - not-yet o*] *lev* **by** *blast+*
then show *?case* **using** *cp* **by** *blast*
qed

lemma $cdcl_W$*-stgy-no-relearned-clause*:
assumes
invR: $cdcl_W$*-all-struct-inv R* **and**
st': $cdcl_W$-$stgy^{**}$ *R S* **and**
bt: *backtrack S T* **and**
confl: *conflicting S* = *Some E* **and**

already-learned: $E \in\#$ *clauses* S **and**
R: *trail* $R = []$
shows *False*
proof $-$
have *M-lev*: $cdcl_W$*-M-level-inv* R
using *invR* **unfolding** $cdcl_W$*-all-struct-inv-def* **by** *auto*
have $cdcl_W$*-M-level-inv* S
using *M-lev assms(2) rtranclp-*$cdcl_W$*-stgy-consistent-inv* **by** *blast*
with *bt* **obtain** *D L M1 M2-loc K i* **where**
T: $T \sim$ *cons-trail* (*Propagated L* ((*D* + {#*L*#})))
(*reduce-trail-to M1* (*add-learned-cls* (*D* + {#*L*#})
(*update-backtrack-lvl* (*get-maximum-level* (*trail S*) *D*) (*update-conflicting None S*))))
and
decomp: (*Marked K* (*Suc* (*get-maximum-level* (*trail S*) *D*)) # *M1*, *M2-loc*) $\in$
set (*get-all-marked-decomposition* (*trail S*)) **and**
k: *get-level* (*trail S*) *L* = *backtrack-lvl S* **and**
level: *get-level* (*trail S*) *L* = *get-maximum-level* (*trail S*) (*D*+{#*L*#}) **and**
confl-S: *conflicting S* = *Some* (*D* + {#*L*#}) **and**
i: *i* = *get-maximum-level* (*trail S*) *D* **and**
undef: *undefined-lit M1 L*
by (*induction rule*: *backtrack-induction-lev2*) *metis*
obtain *M2* **where**
M: *trail S* = *M2* @ *Marked K* (*Suc i*) # *M1*
using *get-all-marked-decomposition-exists-prepend*[*OF decomp*] **unfolding** *i* **by** (*metis append-assoc*)

have *invS*: $cdcl_W$*-all-struct-inv* S
using *invR rtranclp-*$cdcl_W$*-all-struct-inv-inv rtranclp-*$cdcl_W$*-stgy-rtranclp-*$cdcl_W$ *st'* **by** *blast*
then have *conf*: $cdcl_W$*-conflicting* S **unfolding** $cdcl_W$*-all-struct-inv-def* **by** *blast*
then have *trail S* $\models$*as CNot* (*D* + {#*L*#}) **unfolding** $cdcl_W$*-conflicting-def confl-S* **by** *auto*
then have *MD*: *trail S* $\models$*as CNot D* **by** *auto*

have *lev'*: $cdcl_W$*-M-level-inv* S **using** *invS* **unfolding** $cdcl_W$*-all-struct-inv-def* **by** *blast*

have *get-lvls-M*: *get-all-levels-of-marked* (*trail S*) = *rev* [*1..<Suc* (*backtrack-lvl S*)]
using *lev'* **unfolding** $cdcl_W$*-M-level-inv-def* **by** *auto*

have *lev*: $cdcl_W$*-M-level-inv* R **using** *invR* **unfolding** $cdcl_W$*-all-struct-inv-def* **by** *blast*
then have *vars-of-D*: *atms-of D* $\subseteq$ *atm-of* ‘ *lits-of M1*
using *backtrack-atms-of-D-in-M1*[*OF lev' undef - decomp - - - T*] *confl-S conf T decomp k level*
lev' i undef **unfolding** $cdcl_W$*-conflicting-def* **by** (*auto simp*: $cdcl_W$*-M-level-inv-def*)
have *no-dup* (*trail S*) **using** *lev'* **by** (*auto simp*: $cdcl_W$*-M-level-inv-decomp*)
have *vars-in-M1*:
$\forall x \in$ *atms-of D*. $x \notin$ *atm-of* ‘ *lits-of* (*M2* @ [*Marked K* (*get-maximum-level* (*trail S*) *D* + *1*)])
apply (*rule vars-of-D distinct-atms-of-incl-not-in-other*[*of*
M2 @ *Marked K* (*get-maximum-level* (*trail S*) *D* + *1*) # [] *M1 D*])
using ‹*no-dup* (*trail S*)› *M vars-of-D* **by** *simp-all*
have *M1-D*: *M1* $\models$*as CNot D*
using *vars-in-M1 true-annots-remove-if-notin-vars*[*of M2* @ *Marked K* (*i* + *1*) # [] *M1 CNot D*]
‹*trail S* $\models$*as CNot D*› *M* **by** *simp*

have *get-lvls-M*: *get-all-levels-of-marked* (*trail S*) = *rev* [*1..<Suc* (*backtrack-lvl S*)]
using *lev'* **unfolding** $cdcl_W$*-M-level-inv-def* **by** *auto*
then have *backtrack-lvl S* > *0* **unfolding** *M* **by** (*auto split*: *split-if-asm simp add*: *upt.simps*(*2*))

obtain *M1' K' Ls* **where**

M': *trail S = Ls @ Marked K' (backtrack-lvl S) # M1'* **and**
Ls: $\forall l \in$ *set Ls.* $\neg$ *is-marked l* **and**
set M1 $\subseteq$ *set M1'*
proof –
let *?Ls = takeWhile (Not o is-marked) (trail S)*
have *MLs*: *trail S = ?Ls @ dropWhile (Not o is-marked) (trail S)*
by *auto*
have *dropWhile (Not o is-marked) (trail S)* $\neq$ [] **unfolding** *M* **by** *auto*
moreover
from *hd-dropWhile[OF this]* **have** *is-marked(hd (dropWhile (Not o is-marked) (trail S)))*
by *simp*
ultimately
obtain *K' K'k* **where**
K'k: *dropWhile (Not o is-marked) (trail S)*
= Marked K' K'k # tl (dropWhile (Not o is-marked) (trail S))
by *(cases dropWhile (Not* $\circ$ *is-marked) (trail S);*
cases hd (dropWhile (Not $\circ$ *is-marked) (trail S)))*
simp-all
moreover have $\forall l \in$ *set ?Ls.* $\neg$*is-marked l* **using** *set-takeWhileD* **by** *force*
moreover
have *get-all-levels-of-marked (trail S)*
= K'k # get-all-levels-of-marked(tl (dropWhile (Not $\circ$ *is-marked) (trail S)))*
apply *(subst MLs, subst K'k)*
using *calculation(2)* **by** *(auto simp add: get-all-levels-of-marked-no-marked)*
then have *K'k = backtrack-lvl S*
using *calculation(2)* **by** *(auto split: split-if-asm simp add: get-lvls-M upt.simps(2))*
moreover have *set M1* $\subseteq$ *set (tl (dropWhile (Not o is-marked) (trail S)))*
unfolding *M* **by** *(induction M2) auto*
ultimately show *?thesis* **using** *that MLs* **by** *metis*
qed

have *get-lvls-M*: *get-all-levels-of-marked (trail S) = rev [1..<Suc (backtrack-lvl S)]*
using *lev'* **unfolding** *cdcl*$_W$*-M-level-inv-def* **by** *auto*
then have *backtrack-lvl S > 0* **unfolding** *M* **by** *(auto split: split-if-asm simp add: upt.simps(2) i)*

have *M1'-D*: *M1'* $\models$*as CNot D* **using** *M1-D* ‹*set M1* $\subseteq$ *set M1'*› **by** *(auto intro: true-annots-mono)*
have $-L \in$ *lits-of (trail S)* **using** *conf confl-S* **unfolding** *cdcl*$_W$*-conflicting-def* **by** *auto*
have *lvls-M1'*: *get-all-levels-of-marked M1' = rev [1..<backtrack-lvl S]*
using *get-lvls-M Ls* **by** *(auto simp add: get-all-levels-of-marked-no-marked M'*
split: split-if-asm simp add: upt.simps(2))
have *L-notin*: *atm-of L* $\in$ *atm-of ‘ lits-of Ls* $\vee$ *atm-of L = atm-of K'*
proof *(rule ccontr)*
assume $\neg$ *?thesis*
then have *atm-of L* $\notin$ *atm-of ‘ lits-of (Marked K' (backtrack-lvl S) # rev Ls)* **by** *simp*
then have *get-level (trail S) L = get-level M1' L*
unfolding *M'* **by** *auto*
then show *False* **using** *get-level-in-levels-of-marked[of M1' L]* ‹*backtrack-lvl S > 0*›
unfolding *k lvls-M1'* **by** *auto*
qed
obtain *Y Z* **where**
RY: *cdcl*$_W$*-stgy*** *R Y* **and**
YZ: *cdcl*$_W$*-stgy Y Z* **and**
nt: $\neg$ ($\exists$ *c. trail Y = c @ Marked K' (backtrack-lvl S) # M1' @ []*) **and**
Z: (λ*a b. cdcl*$_W$*-stgy a b* $\wedge$ ($\exists$ *c. trail a = c @ Marked K' (backtrack-lvl S) # M1' @ []*))**
Z S

using *rtranclp-cdcl*$_W$*-new-marked-at-beginning-is-decide*′[*OF st*′ - - *lev*, *of Ls K*′
backtrack-lvl S M1′ []]
unfolding *R M*′ **by** *auto*
have [*simp*]: *cdcl*$_W$*-M-level-inv Y*
using *RY lev rtranclp-cdcl*$_W$*-stgy-consistent-inv* **by** *blast*
obtain *M*′ **where** *trZ*: *trail Z* = *M*′ @ *Marked K*′ (*backtrack-lvl S*) # *M1*′
using *rtranclp-cdcl*$_W$*-stgy-with-trail-end-has-trail-end*[*OF Z*] *M*′ **by** *auto*
have *no-dup* (*trail Y*)
using *RY lev rtranclp-cdcl*$_W$*-stgy-consistent-inv* **unfolding** *cdcl*$_W$*-M-level-inv-def* **by** *blast*
then obtain *Y*′ **where**
dec: *decide Y Y*′ **and**
Y′*Z*: *full cdcl*$_W$*-cp Y*′ *Z* **and**
no-step cdcl$_W$*-cp Y*
using *cdcl*$_W$*-stgy-trail-has-new-marked-is-decide-step*[*OF YZ nt Z*] *M*′ **by** *auto*
have *trY*: *trail Y* = *M1*′
proof −
obtain *M*′ **where** *M*: *trail Z* = *M*′ @ *Marked K*′ (*backtrack-lvl S*) # *M1*′
using *rtranclp-cdcl*$_W$*-stgy-with-trail-end-has-trail-end*[*OF Z*] *M*′ **by** *auto*
obtain *M*′′ **where** *M*′′: *trail Z* = *M*′′ @ *trail Y*′ **and** ∀ *m*∈*set M*′′. ¬*is-marked m*
using *Y*′*Z rtranclp-cdcl*$_W$*-cp-dropWhile-trail*′ **unfolding** *full-def* **by** *blast*
obtain *M*′′′ **where** *trail Y*′ = *M*′′′ @ *Marked K*′ (*backtrack-lvl S*) # *M1*′
using *M*′′ **unfolding** *M*
by (*metis* (*no-types*, *lifting*) ‹∀ *m*∈*set M*′′. ¬ *is-marked m*› *beginning-not-marked-invert*)
then show *?thesis* **using** *dec nt* **by** (*induction M*′′′) *auto*
qed
have *Y-CT*: *conflicting Y* = *None* **using** ‹*decide Y Y*′› **by** *auto*
have *cdcl*$_W$** *R Y* **by** (*simp add*: *RY rtranclp-cdcl*$_W$*-stgy-rtranclp-cdcl*$_W$)
then have *init-clss Y* = *init-clss R* **using** *rtranclp-cdcl*$_W$*-init-clss*[*of R Y*] *M-lev* **by** *auto*
{ assume *DL*: *D* + {#*L*#} ∈# *clauses Y*
have *atm-of L* ∉ *atm-of* ‘ *lits-of M1*
apply (*rule backtrack-lit-skiped*[*of S*])
using *decomp i k lev*′ **unfolding** *cdcl*$_W$*-M-level-inv-def* **by** *auto*
then have *LM1*: *undefined-lit M1 L*
by (*metis Marked-Propagated-in-iff-in-lits-of atm-of-uminus image-eqI*)
have *L-trY*: *undefined-lit* (*trail Y*) *L*
using *L-notin* ‹*no-dup* (*trail S*)› **unfolding** *defined-lit-map trY M*′
by (*auto simp add*: *image-iff lits-of-def*)
have ∃ *Y*′. *propagate Y Y*′
using *propagate-rule*[*of Y*] *DL M1*′*-D L-trY Y-CT trY DL* **by** (*metis state-eq-ref*)
then have *False* **using** ‹*no-step cdcl*$_W$*-cp Y*› *propagate*′ **by** *blast*
}
moreover {
assume *DL*: *D* + {#*L*#} ∉# *clauses Y*
have *lY-lZ*: *learned-clss Y* = *learned-clss Z*
using *dec Y*′*Z rtranclp-cdcl*$_W$*-cp-learned-clause-inv*[*of Y*′ *Z*] **unfolding** *full-def*
by *auto*
have *invZ*: *cdcl*$_W$*-all-struct-inv Z*
by (*meson RY YZ invR r-into-rtranclp rtranclp-cdcl*$_W$*-all-struct-inv-inv*
rtranclp-cdcl$_W$*-stgy-rtranclp-cdcl*$_W$)
have *D* + {#*L*#} ∉#*learned-clss S*
apply (*rule rtranclp-cdcl*$_W$*-stgy-with-trail-end-has-not-been-learned*[*OF Z invZ trZ*])
using *DL lY-lZ* **unfolding** *clauses-def* **apply** *simp*
apply (*metis* (*no-types*, *lifting*) ‹*set M1* ⊆ *set M1*′› *image-mono order-trans*
vars-of-D lits-of-def)
using *L-notin* ‹*no-dup* (*trail S*)› **unfolding** *M*′ **by** (*auto simp add*: *image-iff lits-of-def*)

```
    then have False
      using already-learned DL confl  st' M-lev unfolding M'
      by (simp add: ‹init-clss Y = init-clss R› clauses-def confl-S
        rtranclp-cdcl_W-stgy-no-more-init-clss)
  }
  ultimately show False by blast
qed

lemma rtranclp-cdcl_W-stgy-distinct-mset-clauses:
  assumes
    invR: cdcl_W-all-struct-inv R and
    st: cdcl_W-stgy** R S and
    dist: distinct-mset (clauses R) and
    R: trail R = []
  shows distinct-mset (clauses S)
  using st
proof (induction)
  case base
  then show ?case using dist by simp
next
  case (step S T) note st = this(1) and s = this(2) and IH = this(3)
  from s show ?case
    proof (cases rule: cdcl_W-stgy.cases)
      case conflict'
      then show ?thesis
        using IH unfolding full1-def by (auto dest: tranclp-cdcl_W-cp-no-more-clauses)
    next
      case (other' S') note o = this(1) and full = this(3)
      have [simp]: clauses T = clauses S'
        using full unfolding full-def by (auto dest: rtranclp-cdcl_W-cp-no-more-clauses)
      show ?thesis
        using o IH
        proof (cases rule: cdcl_W-o-rule-cases)
          case backtrack
          moreover
            have cdcl_W-all-struct-inv S
              using invR rtranclp-cdcl_W-stgy-cdcl_W-all-struct-inv st by blast
            then have cdcl_W-M-level-inv S
              unfolding cdcl_W-all-struct-inv-def by auto
          ultimately obtain E where
            conflicting S = Some E and
            cls-S': clauses S' = {#E#} + clauses S
            using ‹cdcl_W-M-level-inv S›
            by (induction rule: backtrack-induction-lev2) (auto simp: cdcl_W-M-level-inv-decomp)
          then have E ∉# clauses S
            using cdcl_W-stgy-no-relearned-clause R invR local.backtrack st by blast
          then show ?thesis using IH by (simp add: distinct-mset-add-single cls-S')
        qed auto
    qed
qed

lemma cdcl_W-stgy-distinct-mset-clauses:
  assumes
    st: cdcl_W-stgy** (init-state N) S and
    no-duplicate-clause: distinct-mset N and
```

no-duplicate-in-clause: *distinct-mset-mset N*
shows *distinct-mset* (*clauses S*)
using *rtranclp-cdcl*$_W$*-stgy-distinct-mset-clauses*[*OF - st*] *assms*
by (*auto simp*: *cdcl*$_W$*-all-struct-inv-def distinct-cdcl*$_W$*-state-def*)

5.9 Decrease of a measure

fun *cdcl*$_W$*-measure* **where**
cdcl$_W$*-measure S* =
[(*3*::*nat*) ^ (*card* (*atms-of-msu* (*init-clss S*))) − *card* (*set-mset* (*learned-clss S*)),
if conflicting S = None then 1 else 0,
if conflicting S = None then card (*atms-of-msu* (*init-clss S*)) − *length* (*trail S*)
else length (*trail S*)
]

lemma *length-model-le-vars-all-inv*:
assumes *cdcl*$_W$*-all-struct-inv S*
shows *length* (*trail S*) ≤ *card* (*atms-of-msu* (*init-clss S*))
using *assms length-model-le-vars*[*of S*] **unfolding** *cdcl*$_W$*-all-struct-inv-def*
by (*auto simp*: *cdcl*$_W$*-M-level-inv-decomp*)
end

context *cdcl*$_W$
begin

lemma *learned-clss-less-upper-bound*:
fixes *S* :: *'st*
assumes
distinct-cdcl$_W$*-state S* **and**
∀ *s* ∈# *learned-clss S*. ¬*tautology s*
shows *card*(*set-mset* (*learned-clss S*)) ≤ *3* ^ *card* (*atms-of-msu* (*learned-clss S*))
proof −
have *set-mset* (*learned-clss S*) ⊆ *simple-clss* (*atms-of-msu* (*learned-clss S*))
apply (*rule simplified-in-simple-clss*)
using *assms* **unfolding** *distinct-cdcl*$_W$*-state-def* **by** *auto*
then have *card*(*set-mset* (*learned-clss S*))
≤ *card* (*simple-clss* (*atms-of-msu* (*learned-clss S*)))
by (*simp add*: *simple-clss-finite card-mono*)
then show *?thesis*
by (*meson atms-of-ms-finite simple-clss-card finite-set-mset order-trans*)
qed

lemma *lexn3*[*intro*!, *simp*]:
$a < a' \vee (a = a' \wedge b < b') \vee (a = a' \wedge b = b' \wedge c < c')$
$\Longrightarrow ([a{::}nat,\ b,\ c],\ [a',\ b',\ c']) \in lexn\ \{(x,\ y).\ x < y\}\ 3$
apply *auto*
unfolding *lexn-conv* **apply** *fastforce*
unfolding *lexn-conv* **apply** *auto*
apply (*metis append.simps*(*1*) *append.simps*(*2*))+
done

lemma *cdcl*$_W$*-measure-decreasing*:
fixes *S* :: *'st*
assumes
cdcl$_W$ *S S'* **and**
no-restart:

$\neg$(*learned-clss S* $\subseteq\#$ *learned-clss S'* $\wedge$ [] = *trail S'* $\wedge$ *conflicting S'* = *None*)
and
learned-clss S $\subseteq\#$ *learned-clss S'* **and**
no-relearn: $\bigwedge S'$. *backtrack S S'* $\Longrightarrow \forall T$. *conflicting S* = *Some T* $\longrightarrow T \notin\#$ *learned-clss S*
and
alien: *no-strange-atm S* **and**
M-level: *cdcl*$_W$*-M-level-inv S* **and**
no-taut: $\forall s \in\#$ *learned-clss S*. $\neg$*tautology s* **and**
no-dup: *distinct-cdcl*$_W$*-state S* **and**
confl: *cdcl*$_W$*-conflicting S*
shows (*cdcl*$_W$*-measure S'*, *cdcl*$_W$*-measure S*) $\in$ *lexn* $\{(a, b).\ a < b\}$ *3*
using *assms*(*1*) *M-level assms*(*2*,*3*)
proof (*induct rule*: *cdcl*$_W$*-all-induct-lev2*)
case (*propagate C L*) **note** *undef* = *this*(*3*) **and** *T* = *this*(*4*) **and** *conf* = *this*(*5*)
have *propa*: *propagate S* (*cons-trail* (*Propagated L* (*C* + {#*L*#})) *S*)
using *propagate-rule*[*OF* - *propagate.hyps*(*1*,*2*)] *propagate.hyps* **by** *auto*
then have *no-dup'*: *no-dup* (*Propagated L* ((*C* + {#*L*#})) # *trail S*)
by (*metis M-level cdcl*$_W$*-M-level-inv-decomp*(*2*) *ann-literal.sel*(*2*) *propagate'*
r-into-rtranclp rtranclp-cdcl$_W$*-cp-consistent-inv trail-cons-trail undef*)

let *?N* = *init-clss S*
have *no-strange-atm* (*cons-trail* (*Propagated L* (*C* + {#*L*#})) *S*)
using *alien cdcl*$_W$*.propagate cdcl*$_W$*-no-strange-atm-inv propa M-level* **by** *blast*
then have *atm-of* ' *lits-of* (*Propagated L* ((*C* + {#*L*#})) # *trail S*)
$\subseteq$ *atms-of-msu* (*init-clss S*)
using *undef* **unfolding** *no-strange-atm-def* **by** *auto*
then have *card* (*atm-of* ' *lits-of* (*Propagated L* ((*C* + {#*L*#})) # *trail S*))
$\leq$ *card* (*atms-of-msu* (*init-clss S*))
by (*meson atms-of-ms-finite card-mono finite-set-mset*)
then have *length* (*Propagated L* ((*C* + {#*L*#})) # *trail S*) $\leq$ *card* (*atms-of-msu ?N*)
using *no-dup-length-eq-card-atm-of-lits-of no-dup'* **by** *fastforce*
then have *H*: *card* (*atms-of-msu* (*init-clss S*)) $-$ *length* (*trail S*)
= *Suc* (*card* (*atms-of-msu* (*init-clss S*)) $-$ *Suc* (*length* (*trail S*)))
by *simp*
show *?case* **using** *conf T undef* **by** (*auto simp*: *H*)
next
case (*decide L*) **note** *conf* = *this*(*1*) **and** *undef* = *this*(*2*) **and** *T* = *this*(*4*)
moreover
have *dec*: *decide S* (*cons-trail* (*Marked L* (*backtrack-lvl S* + *1*)) (*incr-lvl S*))
using *decide.intros decide.hyps* **by** *force*
then have *cdcl*$_W$:*cdcl*$_W$ *S* (*cons-trail* (*Marked L* (*backtrack-lvl S* + *1*)) (*incr-lvl S*))
using *cdcl*$_W$*.simps* **by** *blast*
moreover
have *lev*: *cdcl*$_W$*-M-level-inv* (*cons-trail* (*Marked L* (*backtrack-lvl S* + *1*)) (*incr-lvl S*))
using *cdcl*$_W$ *M-level cdcl*$_W$*-consistent-inv*[*OF cdcl*$_W$] **by** *auto*
then have *no-dup*: *no-dup* (*Marked L* (*backtrack-lvl S* + *1*) # *trail S*)
using *undef* **unfolding** *cdcl*$_W$*-M-level-inv-def* **by** *auto*
have *no-strange-atm* (*cons-trail* (*Marked L* (*backtrack-lvl S* + *1*)) (*incr-lvl S*))
using *M-level alien calculation*(*4*) *cdcl*$_W$*-no-strange-atm-inv* **by** *blast*
then have *length* (*Marked L* ((*backtrack-lvl S*) + *1*) # (*trail S*))
$\leq$ *card* (*atms-of-msu* (*init-clss S*))
using *no-dup clauses-def undef*
length-model-le-vars[*of cons-trail* (*Marked L* (*backtrack-lvl S* + *1*)) (*incr-lvl S*)]
by *fastforce*
ultimately show *?case* **using** *conf* **by** *auto*

next
  case (*skip L C' M D*) **note** *tr* = *this*(*1*) **and** *conf* = *this*(*2*) **and** *T* = *this*(*5*)
  show *?case* **using** *conf T* **unfolding** *clauses-def* **by** (*simp add*: *tr*)
next
  case *conflict*
  then show *?case* **by** *simp*
next
  case *resolve*
  then show *?case* **using** *finite* **unfolding** *clauses-def* **by** *simp*
next
  case (*backtrack K i M1 M2 L D T*) **note** *decomp* = *this*(*1*) **and** *conf* = *this*(*3*) **and** *undef* = *this*(*6*)
and
    T =*this*(*7*) **and** *lev* = *this*(*8*)
  let *?S'* = *T*
  have *bt*: *backtrack S ?S'*
    using *backtrack.hyps backtrack.intros*[*of S - - - - D L K i*] **by** *auto*
  have *D* + {#*L*#} ∉# *learned-clss S*
    using *no-relearn conf bt* **by** *auto*
  then have *card-T*:
    card (*set-mset* ({#*D* + {#*L*#}#} + *learned-clss S*)) = *Suc* (*card* (*set-mset* (*learned-clss S*)))
    by (*simp add*:)
  have *distinct-cdcl$_W$-state ?S'*
    using *bt M-level distinct-cdcl$_W$-state-inv no-dup other* **by** *blast*
  moreover have ∀ *s*∈#*learned-clss ?S'*. ¬ *tautology s*
    using *learned-clss-are-not-tautologies*[*OF cdcl$_W$.other*[*OF cdcl$_W$-o.bj*[*OF*
      cdcl$_W$-bj.backtrack[*OF bt*]]]] *M-level no-taut confl* **by** *auto*
  ultimately have *card* (*set-mset* (*learned-clss T*)) ≤ *3* ^ *card* (*atms-of-msu* (*learned-clss T*))
      by (*auto simp*: *clauses-def learned-clss-less-upper-bound*)
    then have *H*: *card* (*set-mset* ({#*D* + {#*L*#}#} + *learned-clss S*))
      ≤ *3* ^ *card* (*atms-of-msu* ({#*D* + {#*L*#}#} + *learned-clss S*))
      using *T undef decomp lev* **by** (*auto simp*: *cdcl$_W$-M-level-inv-decomp*)
  moreover
    have *atms-of-msu* ({#*D* + {#*L*#}#} + *learned-clss S*) ⊆ *atms-of-msu* (*init-clss S*)
      using *alien conf* **unfolding** *no-strange-atm-def* **by** *auto*
    then have *card-f*: *card* (*atms-of-msu* ({#*D* + {#*L*#}#} + *learned-clss S*))
      ≤ *card* (*atms-of-msu* (*init-clss S*))
      by (*meson atms-of-ms-finite card-mono finite-set-mset*)
    then have (*3*::*nat*) ^ *card* (*atms-of-msu* ({#*D* + {#*L*#}#} + *learned-clss S*))
      ≤ *3* ^ *card* (*atms-of-msu* (*init-clss S*)) **by** *simp*
  ultimately have (*3*::*nat*) ^ *card* (*atms-of-msu* (*init-clss S*))
    ≥ *card* (*set-mset* ({#*D* + {#*L*#}#} + *learned-clss S*))
    using *le-trans* **by** *blast*
  then show *?case* **using** *decomp undef diff-less-mono2 card-T T lev*
    by (*auto simp*: *cdcl$_W$-M-level-inv-decomp*)
next
  case *restart*
  then show *?case* **using** *alien* **by** (*auto simp*: *state-eq-def simp del*: *state-simp*)
next
  case (*forget C T*)
  then have *C* ∈# *learned-clss S* **and** *C* ∉# *learned-clss T*
    by *auto*
  then show *?case* **using** *forget*(*9*) **by** (*simp add*: *mset-leD*)
qed

lemma *propagate-measure-decreasing*:

fixes S :: $'st$
assumes *propagate* S S' **and** $cdcl_W$*-all-struct-inv* S
shows ($cdcl_W$*-measure* S', $cdcl_W$*-measure* S) $\in$ *lexn* $\{(a, b).\ a < b\}$ *3*
apply (*rule* $cdcl_W$*-measure-decreasing*)
using *assms*(*1*) *propagate* **apply** *blast*
using *assms*(*1*) **apply** (*auto simp add*: *propagate.simps*)[*3*]
using *assms*(*2*) **apply** (*auto simp add*: $cdcl_W$*-all-struct-inv-def*)
done

lemma *conflict-measure-decreasing*:
fixes S :: $'st$
assumes *conflict* S S' **and** $cdcl_W$*-all-struct-inv* S
shows ($cdcl_W$*-measure* S', $cdcl_W$*-measure* S) $\in$ *lexn* $\{(a, b).\ a < b\}$ *3*
apply (*rule* $cdcl_W$*-measure-decreasing*)
using *assms*(*1*) *conflict* **apply** *blast*
using *assms*(*1*) **apply** (*auto simp add*: *propagate.simps*)[*3*]
using *assms*(*2*) **apply** (*auto simp add*: $cdcl_W$*-all-struct-inv-def*)
done

lemma *decide-measure-decreasing*:
fixes S :: $'st$
assumes *decide* S S' **and** $cdcl_W$*-all-struct-inv* S
shows ($cdcl_W$*-measure* S', $cdcl_W$*-measure* S) $\in$ *lexn* $\{(a, b).\ a < b\}$ *3*
apply (*rule* $cdcl_W$*-measure-decreasing*)
using *assms*(*1*) *decide other* **apply** *blast*
using *assms*(*1*) **apply** (*auto simp add*: *propagate.simps*)[*3*]
using *assms*(*2*) **apply** (*auto simp add*: $cdcl_W$*-all-struct-inv-def*)
done

lemma *trans-le*:
trans $\{(a, (b::nat)).\ a < b\}$
unfolding *trans-def* **by** *auto*

lemma $cdcl_W$*-cp-measure-decreasing*:
fixes S :: $'st$
assumes $cdcl_W$*-cp* S S' **and** $cdcl_W$*-all-struct-inv* S
shows ($cdcl_W$*-measure* S', $cdcl_W$*-measure* S) $\in$ *lexn* $\{(a, b).\ a < b\}$ *3*
using *assms*
proof *induction*
case *conflict'*
then show *?case* **using** *conflict-measure-decreasing* **by** *blast*
next
case *propagate'*
then show *?case* **using** *propagate-measure-decreasing* **by** *blast*
qed

lemma *tranclp-*$cdcl_W$*-cp-measure-decreasing*:
fixes S :: $'st$
assumes $cdcl_W$*-cp*$^{++}$ S S' **and** $cdcl_W$*-all-struct-inv* S
shows ($cdcl_W$*-measure* S', $cdcl_W$*-measure* S) $\in$ *lexn* $\{(a, b).\ a < b\}$ *3*
using *assms*
proof *induction*
case *base*
then show *?case* **using** $cdcl_W$*-cp-measure-decreasing* **by** *blast*
next

case (*step T U*) **note** *st* = *this*(*1*) **and** *step* = *this*(*2*) **and** *IH* =*this*(*3*) **and** *inv* = *this*(*4*)
then have ($cdcl_W$*-measure T*, $cdcl_W$*-measure S*) ∈ *lexn* {*a. case a of* (*a*, *b*) ⇒ *a* < *b*} *3* **by** *blast*

moreover have ($cdcl_W$*-measure U*, $cdcl_W$*-measure T*) ∈ *lexn* {*a. case a of* (*a*, *b*) ⇒ *a* < *b*} *3*
using $cdcl_W$*-cp-measure-decreasing*[*OF step*] *rtranclp-*$cdcl_W$*-all-struct-inv-inv inv*
tranclp-$cdcl_W$*-cp-tranclp-*$cdcl_W$[*OF st*]
unfolding *trans-def rtranclp-unfold*
by *blast*
ultimately show *?case* **using** *lexn-transI*[*OF trans-le*] **unfolding** *trans-def* **by** *blast*
qed

lemma $cdcl_W$*-stgy-step-decreasing*:
fixes *R S T* :: *'st*
assumes $cdcl_W$*-stgy S T* **and**
$cdcl_W$*-stgy*** *R S*
trail R = [] **and**
$cdcl_W$*-all-struct-inv R*
shows ($cdcl_W$*-measure T*, $cdcl_W$*-measure S*) ∈ *lexn* {(*a*, *b*). *a* < *b*} *3*
proof −
have $cdcl_W$*-all-struct-inv S*
using *assms*
by (*metis rtranclp-unfold rtranclp-*$cdcl_W$*-all-struct-inv-inv tranclp-*$cdcl_W$*-stgy-tranclp-*$cdcl_W$)
with *assms* **show** *?thesis*
proof *induction*
case (*conflict' V*) **note** *cp* = *this*(*1*) **and** *inv* = *this*(*5*)
show *?case*
using *tranclp-*$cdcl_W$*-cp-measure-decreasing*[*OF HOL.conjunct1*[*OF cp*[*unfolded full1-def*]] *inv*]
.
next
case (*other' T U*) **note** *st* = *this*(*1*) **and** *H*= *this*(*4*,*5*,*6*,*7*) **and** *cp* = *this*(*3*)
have $cdcl_W$*-all-struct-inv T*
using $cdcl_W$*-all-struct-inv-inv other other'.hyps*(*1*) *other'.prems*(*4*) **by** *blast*
from *tranclp-*$cdcl_W$*-cp-measure-decreasing*[*OF - this*]
have *le-or-eq*: ($cdcl_W$*-measure U*, $cdcl_W$*-measure T*) ∈ *lexn* {*a. case a of* (*a*, *b*) ⇒ *a* < *b*} *3* ∨
$cdcl_W$*-measure U* = $cdcl_W$*-measure T*
using *cp* **unfolding** *full-def rtranclp-unfold* **by** *blast*
moreover
have $cdcl_W$*-M-level-inv S*
using $cdcl_W$*-all-struct-inv-def other'.prems*(*4*) **by** *blast*
with *st* **have** ($cdcl_W$*-measure T*, $cdcl_W$*-measure S*) ∈ *lexn* {*a. case a of* (*a*, *b*) ⇒ *a* < *b*} *3*
proof (*induction rule*:$cdcl_W$*-o-induct-lev2*)
case (*decide T*)
then show *?case* **using** *decide-measure-decreasing H* **by** *blast*
next
case (*backtrack K i M1 M2 L D T*) **note** *decomp* = *this*(*1*) **and** *undef* = *this*(*6*) **and** *T* = *this*(*7*)
have *bt*: *backtrack S T*
apply (*rule backtrack-rule*)
using *backtrack.hyps* **by** *auto*
then have *no-relearn*: ∀ *T*. *conflicting S* = *Some T* ⟶ *T* ∉# *learned-clss S*
using $cdcl_W$*-stgy-no-relearned-clause*[*of R S T*] *H*
unfolding $cdcl_W$*-all-struct-inv-def clauses-def* **by** *auto*
have *inv*: $cdcl_W$*-all-struct-inv S*
using ⟨$cdcl_W$*-all-struct-inv S*⟩ **by** *blast*
show *?case*

apply (*rule* $cdcl_W$*-measure-decreasing*)
using *bt* $cdcl_W$*-bj.backtrack* $cdcl_W$*-o.bj other* **apply** *simp*
using *bt T undef decomp inv* **unfolding** $cdcl_W$*-all-struct-inv-def*
$cdcl_W$*-M-level-inv-def* **apply** *auto*[]
using *bt T undef decomp inv* **unfolding** $cdcl_W$*-all-struct-inv-def*
$cdcl_W$*-M-level-inv-def* **apply** *auto*[]
using *bt no-relearn* **apply** *auto*[]
using *inv* **unfolding** $cdcl_W$*-all-struct-inv-def* **apply** *simp*
using *inv* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-def* **apply** *simp*
using *inv* **unfolding** $cdcl_W$*-all-struct-inv-def* **apply** *simp*
using *inv* **unfolding** $cdcl_W$*-all-struct-inv-def* **apply** *simp*
using *inv* **unfolding** $cdcl_W$*-all-struct-inv-def* **by** *simp*
next
case *skip*
then show *?case* **by** *force*
next
case *resolve*
then show *?case* **by** *force*
qed
ultimately show *?case*
by (*metis lexn-transI transD trans-le*)
qed
qed

lemma *tranclp-*$cdcl_W$*-stgy-decreasing*:
fixes $R\ S\ T :: 'st$
assumes $cdcl_W$*-stgy*$^{++}$ $R\ S$
trail $R = []$ **and**
$cdcl_W$*-all-struct-inv* R
shows ($cdcl_W$*-measure* S, $cdcl_W$*-measure* R) $\in$ *lexn* $\{(a, b).\ a < b\}$ *3*
using *assms*
apply *induction*
using $cdcl_W$*-stgy-step-decreasing*[*of R - R*] **apply** *blast*
using $cdcl_W$*-stgy-step-decreasing*[*of - - R*] *tranclp-into-rtranclp*[*of* $cdcl_W$*-stgy R*]
lexn-transI[*OF trans-le*, *of 3*] **unfolding** *trans-def* **by** *blast*

lemma *tranclp-*$cdcl_W$*-stgy-S0-decreasing*:
fixes $R\ S\ T :: 'st$
assumes *pl*: $cdcl_W$*-stgy*$^{++}$ (*init-state* N) S **and**
no-dup: *distinct-mset-mset* N
shows ($cdcl_W$*-measure* S, $cdcl_W$*-measure* (*init-state* N)) $\in$ *lexn* $\{(a, b).\ a < b\}$ *3*
proof −
have $cdcl_W$*-all-struct-inv* (*init-state* N)
using *no-dup* **unfolding** $cdcl_W$*-all-struct-inv-def* **by** *auto*
then show *?thesis* **using** *pl tranclp-*$cdcl_W$*-stgy-decreasing init-state-trail* **by** *blast*
qed

lemma *wf-tranclp-*$cdcl_W$*-stgy*:
wf $\{(S::'st,$ *init-state* $N)|\ S\ N.$ *distinct-mset-mset* $N \wedge$ $cdcl_W$*-stgy*$^{++}$ (*init-state* N) $S\}$
apply (*rule wf-wf-if-measure′-notation2*[*of lexn* $\{(a, b).\ a < b\}$ *3 - -* $cdcl_W$*-measure*])
apply (*simp add*: *wf wf-lexn*)
using *tranclp-*$cdcl_W$*-stgy-S0-decreasing* **by** *blast*
end

end

theory *DPLL-CDCL-W-Implementation*
imports *Partial-Annotated-Clausal-Logic*
begin

6 Simple Implementation of the DPLL and CDCL

6.1 Common Rules

6.1.1 Propagation

The following theorem holds:

lemma *lits-of-unfold*[*iff*]:
  ($\forall$ *c* $\in$ *set C*. $-$*c* $\in$ *lits-of Ms*) $\longleftrightarrow$ *Ms* $\models$*as CNot* (*mset C*)
  unfolding *true-annots-def Ball-def true-annot-def CNot-def mem-set-multiset-eq* **by** *auto*

The right-hand version is written at a high-level, but only the left-hand side is executable.

definition *is-unit-clause* :: *'a literal list* $\Rightarrow$ (*'a*, *'b*, *'c*) *ann-literal list* $\Rightarrow$ *'a literal option*
 where
 is-unit-clause l M =
   (*case List.filter* (λ*a*. *atm-of a* $\notin$ *atm-of* ' *lits-of M*) *l of*
     a # [] $\Rightarrow$ *if M* $\models$*as CNot* (*mset l* $-$ {#*a*#}) *then Some a else None*
   | - $\Rightarrow$ *None*)

definition *is-unit-clause-code* :: *'a literal list* $\Rightarrow$ (*'a*, *'b*, *'c*) *ann-literal list*
  $\Rightarrow$ *'a literal option* **where**
 is-unit-clause-code l M =
   (*case List.filter* (λ*a*. *atm-of a* $\notin$ *atm-of* ' *lits-of M*) *l of*
     a # [] $\Rightarrow$ *if* ($\forall$ *c* $\in$*set* (*remove1 a l*). $-$*c* $\in$ *lits-of M*) *then Some a else None*
   | - $\Rightarrow$ *None*)

lemma *is-unit-clause-is-unit-clause-code*[*code*]:
  is-unit-clause l M = *is-unit-clause-code l M*
proof $-$
  have *1*: $\bigwedge$*a*. ($\forall$ *c*$\in$*set* (*remove1 a l*). $-$ *c* $\in$ *lits-of M*) $\longleftrightarrow$ *M* $\models$*as CNot* (*mset l* $-$ {#*a*#})
    using *lits-of-unfold*[*of remove1 - l*, *of - M*] **by** *simp*
  thus *?thesis*
    unfolding *is-unit-clause-code-def is-unit-clause-def 1* **by** *blast*
qed

lemma *is-unit-clause-some-undef*:
  assumes *is-unit-clause l M* = *Some a*
  shows *undefined-lit M a*
proof $-$
  have (*case* [*a*$\leftarrow$*l* . *atm-of a* $\notin$ *atm-of* ' *lits-of M*] *of* [] $\Rightarrow$ *None*
          | [*a*] $\Rightarrow$ *if M* $\models$*as CNot* (*mset l* $-$ {#*a*#}) *then Some a else None*
          | *a* # *ab* # *xa* $\Rightarrow$ *Map.empty xa*) = *Some a*
    using *assms* **unfolding** *is-unit-clause-def* **.**
  hence *a* $\in$ *set* [*a*$\leftarrow$*l* . *atm-of a* $\notin$ *atm-of* ' *lits-of M*]
    apply (*cases* [*a*$\leftarrow$*l* . *atm-of a* $\notin$ *atm-of* ' *lits-of M*])
      apply *simp*
    apply (*rename-tac aa list*; *case-tac list*) **by** (*auto split*: *split-if-asm*)
  hence *atm-of a* $\notin$ *atm-of* ' *lits-of M* **by** *auto*
  thus *?thesis*
    by (*simp add*: *Marked-Propagated-in-iff-in-lits-of*
      atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set)

qed

lemma *is-unit-clause-some-CNot*: *is-unit-clause l M = Some a* $\Longrightarrow$ *M* $\models$*as CNot* (*mset l* $-$ {#*a*#})
 unfolding *is-unit-clause-def*
proof $-$
 assume (*case* [*a*$\leftarrow$*l* . *atm-of a* $\notin$ *atm-of* ‘ *lits-of M*] *of* [] $\Rightarrow$ *None*
  | [*a*] $\Rightarrow$ *if M* $\models$*as CNot* (*mset l* $-$ {#*a*#}) *then Some a else None*
  | *a* # *ab* # *xa* $\Rightarrow$ *Map.empty xa*) = *Some a*
 thus *?thesis*
  apply (*cases* [*a*$\leftarrow$*l* . *atm-of a* $\notin$ *atm-of* ‘ *lits-of M*], *simp*)
   apply *simp*
  apply (*rename-tac aa list*, *case-tac list*) **by** (*auto split*: *split-if-asm*)
qed

lemma *is-unit-clause-some-in*: *is-unit-clause l M = Some a* $\Longrightarrow$ *a* $\in$ *set l*
 unfolding *is-unit-clause-def*
proof $-$
 assume (*case* [*a*$\leftarrow$*l* . *atm-of a* $\notin$ *atm-of* ‘ *lits-of M*] *of* [] $\Rightarrow$ *None*
  | [*a*] $\Rightarrow$ *if M* $\models$*as CNot* (*mset l* $-$ {#*a*#}) *then Some a else None*
  | *a* # *ab* # *xa* $\Rightarrow$ *Map.empty xa*) = *Some a*
 thus *a* $\in$ *set l*
  by (*cases* [*a*$\leftarrow$*l* . *atm-of a* $\notin$ *atm-of* ‘ *lits-of M*])
   (*fastforce dest*: *filter-eq-ConsD split*: *split-if-asm* *split*: *list.splits*)+
qed

lemma *is-unit-clause-nil*[*simp*]: *is-unit-clause* [] *M = None*
 unfolding *is-unit-clause-def* **by** *auto*

6.1.2 Unit propagation for all clauses

Finding the first clause to propagate

fun *find-first-unit-clause* :: *'a literal list list* $\Rightarrow$ (*'a*, *'b*, *'c*) *ann-literal list*
 $\Rightarrow$ (*'a literal* $\times$ *'a literal list*) *option* **where**
find-first-unit-clause (*a* # *l*) *M* =
 (*case is-unit-clause a M of*
  None $\Rightarrow$ *find-first-unit-clause l M*
 | *Some L* $\Rightarrow$ *Some* (*L*, *a*)) |
find-first-unit-clause [] - = *None*

lemma *find-first-unit-clause-some*:
 find-first-unit-clause l M = Some (*a*, *c*)
 $\Longrightarrow$ *c* $\in$ *set l* $\wedge$ *M* $\models$*as CNot* (*mset c* $-$ {#*a*#}) $\wedge$ *undefined-lit M a* $\wedge$ *a* $\in$ *set c*
 apply (*induction l*)
  apply *simp*
 by (*auto split*: *option.splits dest*: *is-unit-clause-some-in is-unit-clause-some-CNot*
   is-unit-clause-some-undef)

lemma *propagate-is-unit-clause-not-None*:
 assumes *dist*: *distinct c* **and**
 M: *M* $\models$*as CNot* (*mset c* $-$ {#*a*#}) **and**
 undef: *undefined-lit M a* **and**
 ac: *a* $\in$ *set c*
 shows *is-unit-clause c M* $\neq$ *None*
proof $-$
 have [*a*$\leftarrow$*c* . *atm-of a* $\notin$ *atm-of* ‘ *lits-of M*] = [*a*]

using *assms*
proof (*induction c*)
case *Nil* **thus** *?case* **by** *simp*
next
case (*Cons ac c*)
show *?case*
proof (*cases a = ac*)
case *True*
thus *?thesis* **using** *Cons*
by (*auto simp del*: *lits-of-unfold*
simp add: *lits-of-unfold*[*symmetric*] *Marked-Propagated-in-iff-in-lits-of*
atm-of-eq-atm-of atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set)
next
case *False*
hence *T*: $mset\ c + \{\#ac\#\} - \{\#a\#\} = mset\ c - \{\#a\#\} + \{\#ac\#\}$
by (*auto simp add*: *multiset-eq-iff*)
show *?thesis* **using** *False Cons*
by (*auto simp add*: *T atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set*)
qed
qed
thus *?thesis*
using *M* **unfolding** *is-unit-clause-def* **by** *auto*
qed

lemma *find-first-unit-clause-none*:
distinct c $\Longrightarrow$ *c* $\in$ *set l* $\Longrightarrow$ *M* $\models$*as CNot* (*mset c* $-$ $\{\#a\#\}$) $\Longrightarrow$ *undefined-lit M a* $\Longrightarrow$ *a* $\in$ *set c*
$\Longrightarrow$ *find-first-unit-clause l M* $\neq$ *None*
by (*induction l*)
(*auto split*: *option.split simp add*: *propagate-is-unit-clause-not-None*)

6.1.3 Decide

fun *find-first-unused-var* :: *'a literal list list* $\Rightarrow$ *'a literal set* $\Rightarrow$ *'a literal option* **where**
find-first-unused-var (*a* # *l*) *M* =
(*case List.find* (λ*lit. lit* $\notin$ *M* $\wedge$ $-$*lit* $\notin$ *M*) *a of*
None $\Rightarrow$ *find-first-unused-var l M*
| *Some a* $\Rightarrow$ *Some a*) |
find-first-unused-var [] - = *None*

lemma *find-none*[*iff*]:
List.find (λ*lit. lit* $\notin$ *M* $\wedge$ $-$*lit* $\notin$ *M*) *a* = *None* $\longleftrightarrow$ *atm-of* ` *set a* $\subseteq$ *atm-of* ` *M*
apply (*induct a*)
using *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set*
by (*force simp add*: *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set*)+

lemma *find-some*: *List.find* (λ*lit. lit* $\notin$ *M* $\wedge$ $-$*lit* $\notin$ *M*) *a* = *Some b* $\Longrightarrow$ *b* $\in$ *set a* $\wedge$ *b* $\notin$ *M* $\wedge$ $-$*b* $\notin$ *M*
unfolding *find-Some-iff* **by** (*metis nth-mem*)

lemma *find-first-unused-var-None*[*iff*]:
find-first-unused-var l M = *None* $\longleftrightarrow$ ($\forall$ *a* $\in$ *set l. atm-of* ` *set a* $\subseteq$ *atm-of* ` *M*)
by (*induct l*)
(*auto split*: *option.splits dest*!: *find-some*
simp add: *image-subset-iff atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set*)

lemma *find-first-unused-var-Some-not-all-incl*:
assumes *find-first-unused-var l M* = *Some c*

shows $\neg(\forall a \in set\ l.\ atm\text{-}of\ `\ set\ a \subseteq atm\text{-}of\ `\ M)$
proof −
  have *find-first-unused-var l M* $\neq$ *None*
    using *assms* **by** (*cases find-first-unused-var l M*) *auto*
  thus $\neg(\forall a \in set\ l.\ atm\text{-}of\ `\ set\ a \subseteq atm\text{-}of\ `\ M)$ **by** *auto*
qed

lemma *find-first-unused-var-Some*:
  find-first-unused-var l M = Some a $\Longrightarrow$ $(\exists m \in set\ l.\ a \in set\ m \wedge a \notin M \wedge -a \notin M)$
  by (*induct l*) (*auto split*: *option.splits dest*: *find-some*)

lemma *find-first-unused-var-undefined*:
  find-first-unused-var l (*lits-of Ms*) = *Some a* $\Longrightarrow$ *undefined-lit Ms a*
  using *find-first-unused-var-Some*[*of l lits-of Ms a*] *Marked-Propagated-in-iff-in-lits-of*
  by *blast*

end
theory *DPLL-W-Implementation*
imports *DPLL-CDCL-W-Implementation DPLL-W ~~/src/HOL/Library/Code-Target-Numeral*
begin

6.2 Simple Implementation of DPLL

6.2.1 Combining the propagate and decide: a DPLL step

definition *DPLL-step* :: *int* $dpll_W$*-ann-literals* × *int literal list list*
  ⇒ *int* $dpll_W$*-ann-literals* × *int literal list list* **where**
DPLL-step = (λ(*Ms*, *N*).
  (*case find-first-unit-clause N Ms of*
    Some (*L*, -) ⇒ (*Propagated L* () # *Ms*, *N*)
  | - ⇒
    if $\exists C \in set\ N.\ (\forall c \in set\ C.\ -c \in lits\text{-}of\ Ms)$
    then
      (*case backtrack-split Ms of*
        (-, *L* # *M*) ⇒ (*Propagated* (− (*lit-of L*)) () # *M*, *N*)
      | (-, -) ⇒ (*Ms*, *N*)
)
    else
    (*case find-first-unused-var N* (*lits-of Ms*) *of*
        Some a ⇒ (*Marked a* () # *Ms*, *N*)
      | *None* ⇒ (*Ms*, *N*))))

Example of propagation:

value *DPLL-step* ([*Marked* (*Neg 1*) ()], [[*Pos* (*1*::*int*), *Neg 2*]])

We define the conversion function between the states as defined in *Prop-DPLL* (with multisets) and here (with lists).

abbreviation *toS* ≡ λ(*Ms*::(*int*, *unit*, *unit*) *ann-literal list*)
                    (*N*:: *int literal list list*). (*Ms*, *mset* (*map mset N*))
abbreviation *toS'* ≡ λ(*Ms*::(*int*, *unit*, *unit*) *ann-literal list*,
                    N:: *int literal list list*). (*Ms*, *mset* (*map mset N*))

Proof of correctness of *DPLL-step*

lemma *DPLL-step-is-a-*$dpll_W$*-step*:
  assumes *step*: (*Ms'*, *N'*) = *DPLL-step* (*Ms*, *N*)

```
  and neq: (Ms, N) ≠ (Ms', N')
  shows dpll_W (toS Ms N) (toS Ms' N')
proof −
  let ?S = (Ms, mset (map mset N))
  { fix L E
    assume unit: find-first-unit-clause N Ms = Some (L, E)
    hence Ms'N: (Ms', N') = (Propagated L () # Ms, N)
      using step unfolding DPLL-step-def by auto
    obtain C where
      C: C ∈ set N and
      Ms: Ms ⊨as CNot (mset C − {#L#}) and
      undef: undefined-lit Ms L and
      L ∈ set C using find-first-unit-clause-some[OF unit] by metis
    have dpll_W (Ms, mset (map mset N))
        (Propagated L () # fst (Ms, mset (map mset N)), snd (Ms, mset (map mset N)))
      apply (rule dpll_W.propagate)
      using Ms undef C ‹L ∈ set C› unfolding mem-set-multiset-eq by (auto simp add: C)
    hence ?thesis using Ms'N by auto
  }
  moreover
  { assume unit: find-first-unit-clause N Ms = None
    assume exC: ∃ C ∈ set N. Ms ⊨as CNot (mset C)
    then obtain C where C: C ∈ set N and Ms: Ms ⊨as CNot (mset C) by auto
    then obtain L M M' where bt: backtrack-split Ms  = (M', L # M)
      using step exC neq unfolding DPLL-step-def prod.case unit
      by (cases backtrack-split Ms, rename-tac b, case-tac b) auto
    hence is-marked L using backtrack-split-snd-hd-marked[of Ms] by auto
    have 1: dpll_W (Ms, mset (map mset N))
                (Propagated (− lit-of L) () # M, snd (Ms, mset (map mset N)))
      apply (rule dpll_W.backtrack[OF - ‹is-marked L›, of ])
      using C Ms bt by auto
    moreover have (Ms', N') = (Propagated (− (lit-of L)) () # M, N)
      using step exC unfolding DPLL-step-def bt prod.case unit by auto
    ultimately have ?thesis by auto
  }
  moreover
  { assume unit: find-first-unit-clause N Ms = None
    assume exC: ¬ (∃ C ∈ set N. Ms ⊨as CNot (mset C))
    obtain L where unused: find-first-unused-var N (lits-of Ms) = Some L
      using step exC neq unfolding DPLL-step-def prod.case unit
      by (cases find-first-unused-var N (lits-of Ms)) auto
    have dpll_W (Ms, mset (map mset N))
             (Marked L () # fst (Ms, mset (map mset N)), snd (Ms, mset (map mset N)))
      apply (rule dpll_W.decided[of ?S L])
      using find-first-unused-var-Some[OF unused]
      by (auto simp add: Marked-Propagated-in-iff-in-lits-of atms-of-ms-def)
    moreover have (Ms', N') = (Marked L () # Ms, N)
      using step exC unfolding DPLL-step-def unused prod.case unit by auto
    ultimately have ?thesis by auto
  }
  ultimately show ?thesis by (cases find-first-unit-clause N Ms) auto
qed

lemma DPLL-step-stuck-final-state:
  assumes step: (Ms, N) = DPLL-step (Ms, N)
```

shows *conclusive-dpll$_W$-state* (*toS Ms N*)
proof −
have *unit*: *find-first-unit-clause N Ms* = *None*
using *step* **unfolding** *DPLL-step-def* **by** (*auto split*:*option.splits*)

{ **assume** *n*: ∃ *C* ∈ *set N*. *Ms* ⊨*as CNot* (*mset C*)
hence *Ms*: (*Ms*, *N*) = (*case backtrack-split Ms of* (*x*, []) ⇒ (*Ms*, *N*)
| (*x*, *L* # *M*) ⇒ (*Propagated* (− *lit-of L*) () # *M*, *N*))
using *step* **unfolding** *DPLL-step-def* **by** (*simp add*:*unit*)

have *snd* (*backtrack-split Ms*) = []
proof (*cases backtrack-split Ms*, *cases snd* (*backtrack-split Ms*))
fix *a b*
assume *backtrack-split Ms* = (*a*, *b*) **and** *snd* (*backtrack-split Ms*) = []
thus *snd* (*backtrack-split Ms*) = [] **by** *blast*
next
fix *a b aa list*
assume
bt: *backtrack-split Ms* = (*a*, *b*) **and**
bt': *snd* (*backtrack-split Ms*) = *aa* # *list*
hence *Ms*: *Ms* = *Propagated* (− *lit-of aa*) () # *list* **using** *Ms* **by** *auto*
have *is-marked aa* **using** *backtrack-split-snd-hd-marked*[*of Ms*] *bt bt'* **by** *auto*
moreover have *fst* (*backtrack-split Ms*) @ *aa* # *list* = *Ms*
using *backtrack-split-list-eq*[*of Ms*] *bt'* **by** *auto*
ultimately have *False* **unfolding** *Ms* **by** *auto*
thus *snd* (*backtrack-split Ms*) = [] **by** *blast*
qed

hence *?thesis*
using *n backtrack-snd-empty-not-marked*[*of Ms*] **unfolding** *conclusive-dpll$_W$-state-def*
by (*cases backtrack-split Ms*) *auto*
}
moreover {
assume *n*: ¬ (∃ *C* ∈ *set N*. *Ms* ⊨*as CNot* (*mset C*))
hence *find-first-unused-var N* (*lits-of Ms*) = *None*
using *step* **unfolding** *DPLL-step-def* **by** (*simp add*: *unit split*: *option.splits*)
hence *a*: ∀ *a* ∈ *set N*. *atm-of* ‘ *set a* ⊆ *atm-of* ‘ (*lits-of Ms*) **by** *auto*
have *fst* (*toS Ms N*) ⊨*asm snd* (*toS Ms N*) **unfolding** *true-annots-def CNot-def Ball-def*
proof *clarify*
fix *x*
assume *x*: *x* ∈ *set-mset* (*clauses* (*toS Ms N*))
hence ¬*Ms* ⊨*as CNot x* **using** *n* **unfolding** *true-annots-def CNot-def Ball-def* **by** *auto*
moreover have *total-over-m* (*lits-of Ms*) {*x*}
using *a x image-iff in-mono atms-of-s-def*
unfolding *total-over-m-def total-over-set-def lits-of-def* **by** *fastforce*
ultimately show *fst* (*toS Ms N*) ⊨*a x*
using *total-not-CNot*[*of lits-of Ms x*] **by** (*simp add*: *true-annot-def true-annots-true-cls*)
qed
hence *?thesis* **unfolding** *conclusive-dpll$_W$-state-def* **by** *blast*
}
ultimately show *?thesis* **by** *blast*
qed

6.2.2 Adding invariants

Invariant tested in the function **function** *DPLL-ci* :: *int* $dpll_W$*-ann-literals* ⇒ *int literal list list*
⇒ *int* $dpll_W$*-ann-literals* × *int literal list list* **where**
DPLL-ci Ms N =
(*if* ¬$dpll_W$*-all-inv* (*Ms*, *mset* (*map mset N*))
then (*Ms*, *N*)
else
let (*Ms'*, *N'*) = *DPLL-step* (*Ms*, *N*) *in*
if (*Ms'*, *N'*) = (*Ms*, *N*) *then* (*Ms*, *N*) *else DPLL-ci Ms' N*)
by *fast+*
termination
proof (*relation* {(*S'*, *S*). (*toS' S'*, *toS' S*) ∈ {(*S'*, *S*). $dpll_W$*-all-inv S* ∧ $dpll_W$ *S S'*}})
show *wf* {(*S'*, *S*).(*toS' S'*, *toS' S*) ∈ {(*S'*, *S*). $dpll_W$*-all-inv S* ∧ $dpll_W$ *S S'*}}
using *wf-if-measure-f*[*OF* $dpll_W$*-wf*, *of toS'*] **by** *auto*
next
fix *Ms* :: *int* $dpll_W$*-ann-literals* **and** *N x xa y*
assume¬ ¬ $dpll_W$*-all-inv* (*toS Ms N*)
and *step*: *x* = *DPLL-step* (*Ms*, *N*)
and *x*: (*xa*, *y*) = *x*
and (*xa*, *y*) ≠ (*Ms*, *N*)
thus ((*xa*, *N*), *Ms*, *N*) ∈ {(*S'*, *S*). (*toS' S'*, *toS' S*) ∈ {(*S'*, *S*). $dpll_W$*-all-inv S* ∧ $dpll_W$ *S S'*}}
using *DPLL-step-is-a-*$dpll_W$*-step* $dpll_W$*-same-clauses split-conv* **by** *fastforce*
qed

No invariant tested **function** (*domintros*) *DPLL-part*:: *int* $dpll_W$*-ann-literals* ⇒ *int literal list list* ⇒
int $dpll_W$*-ann-literals* × *int literal list list* **where**
DPLL-part Ms N =
(*let* (*Ms'*, *N'*) = *DPLL-step* (*Ms*, *N*) *in*
if (*Ms'*, *N'*) = (*Ms*, *N*) *then* (*Ms*, *N*) *else DPLL-part Ms' N*)
by *fast+*

lemma *snd-DPLL-step*[*simp*]:
snd (*DPLL-step* (*Ms*, *N*)) = *N*
unfolding *DPLL-step-def* **by** (*auto split*: *split-if option.splits prod.splits list.splits*)

lemma $dpll_W$*-all-inv-implieS-2-eq3-and-dom*:
assumes $dpll_W$*-all-inv* (*Ms*, *mset* (*map mset N*))
shows *DPLL-ci Ms N* = *DPLL-part Ms N* ∧ *DPLL-part-dom* (*Ms*, *N*)
using *assms*
proof (*induct rule*: *DPLL-ci.induct*)
case (*1 Ms N*)
have *snd* (*DPLL-step* (*Ms*, *N*)) = *N* **by** *auto*
then obtain *Ms'* **where** *Ms'*: *DPLL-step* (*Ms*, *N*) = (*Ms'*, *N*) **by** (*cases DPLL-step* (*Ms*, *N*)) *auto*
have *inv'*: $dpll_W$*-all-inv* (*toS Ms' N*) **by** (*metis* (*mono-tags*) *1.prems DPLL-step-is-a-*$dpll_W$*-step*
Ms' $dpll_W$*-all-inv old.prod.inject*)
{ assume (*Ms'*, *N*) ≠ (*Ms*, *N*)
hence *DPLL-ci Ms' N* = *DPLL-part Ms' N* ∧ *DPLL-part-dom* (*Ms'*, *N*) **using** *1*(*1*)[*of - Ms' N*] *Ms'*
1(*2*) *inv'* **by** *auto*
hence *DPLL-part-dom* (*Ms*, *N*) **using** *DPLL-part.domintros Ms'* **by** *fastforce*
moreover have *DPLL-ci Ms N* = *DPLL-part Ms N* **using** *1.prems DPLL-part.psimps Ms'*
‹*DPLL-ci Ms' N* = *DPLL-part Ms' N* ∧ *DPLL-part-dom* (*Ms'*, *N*)› ‹*DPLL-part-dom* (*Ms*, *N*)› **by** *auto*

ultimately have *?case* **by** *blast*
}
moreover {
assume $(Ms', N) = (Ms, N)$
hence *?case* **using** *DPLL-part.domintros DPLL-part.psimps Ms'* **by** *fastforce*
}
ultimately show *?case* **by** *blast*
qed

lemma *DPLL-ci-dpll*$_W$*-rtranclp*:
assumes *DPLL-ci* $Ms\ N = (Ms', N')$
shows $dpll_W{}^{**}$ $(toS\ Ms\ N)$ $(toS\ Ms'\ N)$
using *assms*
proof (*induct Ms N arbitrary*: $Ms'\ N'$ *rule*: *DPLL-ci.induct*)
case $(1\ Ms\ N\ Ms'\ N')$ **note** $IH = this(1)$ **and** $step = this(2)$
obtain $S_1\ S_2$ **where** S: $(S_1, S_2) = DPLL\text{-}step\ (Ms, N)$ **by** (*cases DPLL-step* (Ms, N)) *auto*

{ assume $\neg dpll_W\text{-}all\text{-}inv\ (toS\ Ms\ N)$
hence $(Ms, N) = (Ms', N)$ **using** *step* **by** *auto*
hence *?case* **by** *auto*
}
moreover
{ assume $dpll_W\text{-}all\text{-}inv\ (toS\ Ms\ N)$
and $(S_1, S_2) = (Ms, N)$
hence *?case* **using** *S step* **by** *auto*
}
moreover
{ assume $dpll_W\text{-}all\text{-}inv\ (toS\ Ms\ N)$
and $(S_1, S_2) \neq (Ms, N)$
moreover obtain $S_1'\ S_2'$ **where** $DPLL\text{-}ci\ S_1\ N = (S_1', S_2')$ **by** (*cases DPLL-ci* S_1 N) *auto*
moreover have $DPLL\text{-}ci\ Ms\ N = DPLL\text{-}ci\ S_1\ N$ **using** *DPLL-ci.simps*[*of Ms N*] *calculation*
proof −
have (*case* (S_1, S_2) *of* $(ms, lss) \Rightarrow$
if $(ms, lss) = (Ms, N)$ *then* (Ms, N) *else DPLL-ci ms N*) $=$ *DPLL-ci Ms N*
using *S DPLL-ci.simps*[*of Ms N*] *calculation* **by** *presburger*
hence (*if* $(S_1, S_2) = (Ms, N)$ *then* (Ms, N) *else DPLL-ci* S_1 N) $=$ *DPLL-ci Ms N*
by *fastforce*
thus *?thesis*
using *calculation*(*2*) **by** *presburger*
qed
ultimately have $dpll_W{}^{**}$ $(toS\ S_1'\ N)$ $(toS\ Ms'\ N)$ **using** *IH*[*of* (S_1, S_2) S_1 S_2] *S step* **by** *simp*

moreover have $dpll_W$ $(toS\ Ms\ N)$ $(toS\ S_1\ N)$
by (*metis DPLL-step-is-a-dpll*$_W$*-step S* ‹$(S_1, S_2) \neq (Ms, N)$› *prod.sel*(*2*) *snd-DPLL-step*)
ultimately have *?case* **by** (*metis* (*mono-tags, hide-lams*) *IH S* ‹$(S_1, S_2) \neq (Ms, N)$›
‹*DPLL-ci Ms N* $=$ *DPLL-ci* S_1 N› ‹$dpll_W\text{-}all\text{-}inv\ (toS\ Ms\ N)$› *converse-rtranclp-into-rtranclp*
local.step)
}
ultimately show *?case* **by** *blast*
qed

lemma *dpll*$_W$*-all-inv-dpll*$_W$*-tranclp-irrefl*:
assumes $dpll_W\text{-}all\text{-}inv\ (Ms, N)$
and $dpll_W{}^{++}$ (Ms, N) (Ms, N)
shows *False*

proof −
  have *1*: *wf* $\{(S', S)$. $dpll_W$*-all-inv* $S \wedge dpll_W{}^{++}$ S $S'\}$ **using** $dpll_W$*-wf-tranclp* **by** *auto*
  have $((Ms, N), (Ms, N)) \in \{(S', S)$. $dpll_W$*-all-inv* $S \wedge dpll_W{}^{++}$ S $S'\}$ **using** *assms* **by** *auto*
  thus *False* **using** *wf-not-refl*[*OF 1*] **by** *blast*
qed

lemma *DPLL-ci-final-state*:
  assumes *step*: *DPLL-ci Ms N* = (*Ms*, *N*)
  and *inv*: $dpll_W$*-all-inv* (*toS Ms N*)
  shows *conclusive-*$dpll_W$*-state* (*toS Ms N*)
proof −
  have *st*: $dpll_W{}^{**}$ (*toS Ms N*) (*toS Ms N*) **using** *DPLL-ci-*$dpll_W$*-rtranclp*[*OF step*] **.**
  have *DPLL-step* (*Ms*, *N*) = (*Ms*, *N*)
    proof (*rule ccontr*)
      obtain *Ms′ N′* **where** *Ms′N*: (*Ms′*, *N′*) = *DPLL-step* (*Ms*, *N*)
        by (*cases DPLL-step* (*Ms*, *N*)) *auto*
      assume ¬ *?thesis*
      hence *DPLL-ci Ms′ N* = (*Ms*, *N*) **using** *step inv st Ms′N*[*symmetric*] **by** *fastforce*
      hence $dpll_W{}^{++}$ (*toS Ms N*) (*toS Ms N*)
       by (*metis DPLL-ci-*$dpll_W$*-rtranclp DPLL-step-is-a-*$dpll_W$*-step Ms′N* ‹*DPLL-step* (*Ms*, *N*) ≠ (*Ms*, *N*)›
          prod.sel(*2*) *rtranclp-into-tranclp2 snd-DPLL-step*)
      thus *False* **using** $dpll_W$*-all-inv-*$dpll_W$*-tranclp-irrefl inv* **by** *auto*
    qed
  thus *?thesis* **using** *DPLL-step-stuck-final-state*[*of Ms N*] **by** *simp*
qed

lemma *DPLL-step-obtains*:
  obtains *Ms′* **where** (*Ms′*, *N*) = *DPLL-step* (*Ms*, *N*)
  unfolding *DPLL-step-def* **by** (*metis* (*no-types*, *lifting*) *DPLL-step-def prod.collapse snd-DPLL-step*)

lemma *DPLL-ci-obtains*:
  obtains *Ms′* **where** (*Ms′*, *N*) = *DPLL-ci Ms N*
proof (*induct rule*: *DPLL-ci.induct*)
  case (*1 Ms N*) **note** *IH* = *this*(*1*) **and** *that* = *this*(*2*)
  obtain *S* **where** *SN*: (*S*, *N*) = *DPLL-step* (*Ms*, *N*) **using** *DPLL-step-obtains* **by** *metis*
  { assume ¬ $dpll_W$*-all-inv* (*toS Ms N*)
    hence *?case* **using** *that* **by** *auto*
  }
  moreover {
    assume *n*: (*S*, *N*) ≠ (*Ms*, *N*)
    and *inv*: $dpll_W$*-all-inv* (*toS Ms N*)
    have ∃ *ms*. *DPLL-step* (*Ms*, *N*) = (*ms*, *N*)
      by (*metis* ‹⋀*thesisa*. (⋀*S*. (*S*, *N*) = *DPLL-step* (*Ms*, *N*) ⟹ *thesisa*) ⟹ *thesisa*›)
    hence *?thesis*
      using *IH that* **by** *fastforce*
  }
  moreover {
    assume *n*: (*S*, *N*) = (*Ms*, *N*)
    hence *?case* **using** *SN that* **by** *fastforce*
  }
  ultimately show *?case* **by** *blast*
qed

lemma *DPLL-ci-no-more-step*:
assumes *step*: *DPLL-ci Ms N* = (*Ms'*, *N'*)
shows *DPLL-ci Ms' N'* = (*Ms'*, *N'*)
using *assms*
proof (*induct arbitrary*: *Ms' N' rule*: *DPLL-ci.induct*)
case (*1 Ms N Ms' N'*) **note** *IH* = *this*(*1*) **and** *step* = *this*(*2*)
obtain S_1 **where** *S*: (S_1, *N*) = *DPLL-step* (*Ms*, *N*) **using** *DPLL-step-obtains* **by** *auto*
{ assume ¬$dpll_W$*-all-inv* (*toS Ms N*)
hence *?case* **using** *step* **by** *auto*
}
moreover {
assume $dpll_W$*-all-inv* (*toS Ms N*)
and (S_1, *N*) = (*Ms*, *N*)
hence *?case* **using** *S step* **by** *auto*
}
moreover
{ assume *inv*: $dpll_W$*-all-inv* (*toS Ms N*)
assume *n*: (S_1, *N*) ≠ (*Ms*, *N*)
obtain S_1' **where** *SS*: (S_1', *N*) = *DPLL-ci* S_1 *N* **using** *DPLL-ci-obtains* **by** *blast*
moreover have *DPLL-ci Ms N* = *DPLL-ci* S_1 *N*
proof −
have (*case* (S_1, *N*) *of* (*ms*, *lss*) ⇒ *if* (*ms*, *lss*) = (*Ms*, *N*) *then* (*Ms*, *N*) *else DPLL-ci ms N*)
= *DPLL-ci Ms N*
using *S DPLL-ci.simps*[*of Ms N*] *calculation inv* **by** *presburger*
hence (*if* (S_1, *N*) = (*Ms*, *N*) *then* (*Ms*, *N*) *else DPLL-ci* S_1 *N*) = *DPLL-ci Ms N*
by *fastforce*
thus *?thesis*
using *calculation n* **by** *presburger*
qed
moreover
have *DPLL-ci* S_1' *N* = (S_1', *N*) **using** *step IH*[*OF - - S n SS*[*symmetric*]] *inv* **by** *blast*
ultimately have *?case* **using** *step* **by** *fastforce*
}
ultimately show *?case* **by** *blast*
qed

lemma *DPLL-part-*$dpll_W$*-all-inv-final*:
fixes *M Ms'*:: (*int*, *unit*, *unit*) *ann-literal list* **and**
N :: *int literal list list*
assumes *inv*: $dpll_W$*-all-inv* (*Ms*, *mset* (*map mset N*))
and *MsN*: *DPLL-part Ms N* = (*Ms'*, *N*)
shows *conclusive-*$dpll_W$*-state* (*toS Ms' N*) ∧ $dpll_W^{**}$ (*toS Ms N*) (*toS Ms' N*)
proof −
have *2*: *DPLL-ci Ms N* = *DPLL-part Ms N* **using** *inv* $dpll_W$*-all-inv-implieS-2-eq3-and-dom* **by** *blast*
hence *star*: $dpll_W^{**}$ (*toS Ms N*) (*toS Ms' N*) **unfolding** *MsN* **using** *DPLL-ci-*$dpll_W$*-rtranclp* **by** *blast*
hence *inv'*: $dpll_W$*-all-inv* (*toS Ms' N*) **using** *inv rtranclp-*$dpll_W$*-all-inv* **by** *blast*
show *?thesis* **using** *star DPLL-ci-final-state*[*OF DPLL-ci-no-more-step inv'*] *2* **unfolding** *MsN* **by** *blast*
qed

Embedding the invariant into the type

Defining the type **typedef** $dpll_W$*-state* =

$\{(M::(int,\ unit,\ unit)\ ann\text{-}literal\ list,\ N::int\ literal\ list\ list).$
$dpll_W\text{-}all\text{-}inv\ (toS\ M\ N)\}$
morphisms *rough-state-of state-of*
proof
show $([],[]) \in \{(M,\ N).\ dpll_W\text{-}all\text{-}inv\ (toS\ M\ N)\}$ **by** (*auto simp add*: *dpll$_W$-all-inv-def*)
qed

lemma
DPLL-part-dom ([], *N*)
using *assms* *dpll$_W$-all-inv-implieS-2-eq3-and-dom*[*of* [] *N*] **by** (*simp add*: *dpll$_W$-all-inv-def*)

Some type classes **instantiation** *dpll$_W$-state* :: *equal*
begin
definition *equal-dpll$_W$-state* :: *dpll$_W$-state* ⇒ *dpll$_W$-state* ⇒ *bool* **where**
equal-dpll$_W$-state $S\ S' = (rough\text{-}state\text{-}of\ S = rough\text{-}state\text{-}of\ S')$
instance
by *standard* (*simp add*: *rough-state-of-inject equal-dpll$_W$-state-def*)
end

DPLL **definition** *DPLL-step′* :: *dpll$_W$-state* ⇒ *dpll$_W$-state* **where**
DPLL-step′ S = *state-of* (*DPLL-step* (*rough-state-of S*))

declare *rough-state-of-inverse*[*simp*]

lemma *DPLL-step-dpll$_W$-conc-inv*:
DPLL-step (*rough-state-of S*) $\in \{(M,\ N).\ dpll_W\text{-}all\text{-}inv\ (toS\ M\ N)\}$
by (*smt DPLL-ci.simps DPLL-ci-dpll$_W$-rtranclp case-prodE case-prodI2 rough-state-of mem-Collect-eq old.prod.case prod.sel*(*2*) *rtranclp-dpll$_W$-all-inv snd-DPLL-step*)

lemma *rough-state-of-DPLL-step′-DPLL-step*[*simp*]:
rough-state-of (*DPLL-step′ S*) = *DPLL-step* (*rough-state-of S*)
using *DPLL-step-dpll$_W$-conc-inv DPLL-step′-def state-of-inverse* **by** *auto*

function *DPLL-tot*:: *dpll$_W$-state* ⇒ *dpll$_W$-state* **where**
DPLL-tot S =
(*let S′* = *DPLL-step′ S in*
if S′ = *S then S else DPLL-tot S′*)
by *fast+*
termination
proof (*relation* $\{(T',\ T).$
$(rough\text{-}state\text{-}of\ T',\ rough\text{-}state\text{-}of\ T)$
$\in \{(S',\ S).\ (toS'\ S',\ toS'\ S)$
$\in \{(S',\ S).\ dpll_W\text{-}all\text{-}inv\ S \wedge dpll_W\ \ S\ S'\}\}\})$
show $wf\ \{(b,\ a).$
$(rough\text{-}state\text{-}of\ b,\ rough\text{-}state\text{-}of\ a)$
$\in \{(b,\ a).\ (toS'\ b,\ toS'\ a)$
$\in \{(b,\ a).\ dpll_W\text{-}all\text{-}inv\ a \wedge dpll_W\ \ a\ b\}\}\}$
using *wf-if-measure-f*[*OF wf-if-measure-f*[*OF dpll$_W$-wf*, *of toS′*], *of rough-state-of*] **.**
next
fix *S x*
assume *x*: *x* = *DPLL-step′ S*
and $x \neq S$
have *dpll$_W$-all-inv* (*case rough-state-of S of* (*Ms*, *N*) ⇒ (*Ms*, *mset* (*map mset N*)))
by (*metis* (*no-types*, *lifting*) *case-prodE mem-Collect-eq old.prod.case rough-state-of*)
moreover have *dpll$_W$* (*case rough-state-of S of* (*Ms*, *N*) ⇒ (*Ms*, *mset* (*map mset N*)))

(*case rough-state-of* (*DPLL-step′ S*) *of* (*Ms*, *N*) ⇒ (*Ms*, *mset* (*map mset N*)))
proof −
obtain *Ms N* **where** *Ms*: (*Ms*, *N*) = *rough-state-of S* **by** (*cases rough-state-of S*) *auto*
have $dpll_W$*-all-inv* (*toS′* (*Ms*, *N*)) **using** *calculation* **unfolding** *Ms* **by** *blast*
moreover obtain *Ms′ N′* **where** *Ms′*: (*Ms′*, *N′*) = *rough-state-of* (*DPLL-step′ S*)
by (*cases rough-state-of* (*DPLL-step′ S*)) *auto*
ultimately have $dpll_W$*-all-inv* (*toS′* (*Ms′*, *N′*)) **unfolding** *Ms′*
by (*metis* (*no-types*, *lifting*) *case-prod-unfold mem-Collect-eq rough-state-of*)

have $dpll_W$ (*toS Ms N*) (*toS Ms′ N′*)
apply (*rule DPLL-step-is-a-*$dpll_W$*-step*[*of Ms′ N′ Ms N*])
unfolding *Ms Ms′* **using** ‹*x* ≠ *S*› *rough-state-of-inject x* **by** *fastforce*+
thus *?thesis* **unfolding** *Ms*[*symmetric*] *Ms′*[*symmetric*] **by** *auto*
qed
ultimately show (*x*, *S*) ∈ {(*T′*, *T*). (*rough-state-of T′*, *rough-state-of T*)
∈ {(*S′*, *S*). (*toS′ S′*, *toS′ S*) ∈ {(*S′*, *S*). $dpll_W$*-all-inv S* ∧ $dpll_W$ *S S′*}}}
by (*auto simp add*: *x*)
qed

lemma [*code*]:
DPLL-tot S =
(*let S′* = *DPLL-step′ S in*
if S′ = *S then S else DPLL-tot S′*) **by** *auto*

lemma *DPLL-tot-DPLL-step-DPLL-tot*[*simp*]: *DPLL-tot* (*DPLL-step′ S*) = *DPLL-tot S*
apply (*cases DPLL-step′ S* = *S*)
apply *simp*
unfolding *DPLL-tot.simps*[*of S*] **by** (*simp del*: *DPLL-tot.simps*)

lemma *DOPLL-step′-DPLL-tot*[*simp*]:
DPLL-step′ (*DPLL-tot S*) = *DPLL-tot S*
by (*rule DPLL-tot.induct*[*of* λ*S*. *DPLL-step′* (*DPLL-tot S*) = *DPLL-tot S S*])
(*metis* (*full-types*) *DPLL-tot.simps*)

lemma *DPLL-tot-final-state*:
assumes *DPLL-tot S* = *S*
shows *conclusive-*$dpll_W$*-state* (*toS′* (*rough-state-of S*))
proof −
have *DPLL-step′ S* = *S* **using** *assms*[*symmetric*] *DOPLL-step′-DPLL-tot* **by** *metis*
hence *DPLL-step* (*rough-state-of S*) = (*rough-state-of S*)
unfolding *DPLL-step′-def* **using** *DPLL-step-*$dpll_W$*-conc-inv rough-state-of-inverse*
by (*metis rough-state-of-DPLL-step′-DPLL-step*)
thus *?thesis*
by (*metis* (*mono-tags*, *lifting*) *DPLL-step-stuck-final-state old.prod.exhaust split-conv*)
qed

lemma *DPLL-tot-star*:
assumes *rough-state-of* (*DPLL-tot S*) = *S′*
shows $dpll_W{}^{**}$ (*toS′* (*rough-state-of S*)) (*toS′ S′*)
using *assms*
proof (*induction arbitrary*: *S′ rule*: *DPLL-tot.induct*)
case (*1 S S′*)
let *?x* = *DPLL-step′ S*

{ **assume** *?x* = *S*
then have *?case* **using** *1*(*2*) **by** *simp*
}
moreover {
assume *S*: *?x* ≠ *S*
have *?case*
apply (*cases DPLL-step′ S* = *S*)
using *S* **apply** *blast*
by (*smt 1.IH 1.prems DPLL-step-is-a-dpll*$_W$*-step DPLL-tot.simps case-prodE2 rough-state-of-DPLL-step′-DPLL-step rtranclp.rtrancl-into-rtrancl rtranclp.rtrancl-refl rtranclp-idemp split-conv*)
}
ultimately show *?case* **by** *auto*
qed

lemma *rough-state-of-rough-state-of-nil*[*simp*]:
rough-state-of (*state-of* ([], *N*)) = ([], *N*)
apply (*rule DPLL-W-Implementation.dpll*$_W$*-state.state-of-inverse*)
unfolding *dpll*$_W$*-all-inv-def* **by** *auto*

Theorem of correctness

lemma *DPLL-tot-correct*:
assumes *rough-state-of* (*DPLL-tot* (*state-of* (([], *N*)))) = (*M*, *N′*)
and (*M′*, *N′′*) = *toS′* (*M*, *N′*)
shows *M′* ⊨*asm N′′* ⟷ *satisfiable* (*set-mset N′′*)
proof −
have *dpll*$_W$** (*toS′* ([], *N*)) (*toS′* (*M*, *N′*)) **using** *DPLL-tot-star*[*OF assms*(*1*)] **by** *auto*
moreover have *conclusive-dpll*$_W$*-state* (*toS′* (*M*, *N′*))
using *DPLL-tot-final-state* **by** (*metis* (*mono-tags*, *lifting*) *DOPLL-step′-DPLL-tot DPLL-tot.simps assms*(*1*))
ultimately show *?thesis* **using** *dpll*$_W$*-conclusive-state-correct* **by** (*smt DPLL-ci.simps DPLL-ci-dpll*$_W$*-rtranclp assms*(*2*) *dpll*$_W$*-all-inv-def prod.case prod.sel*(*1*) *prod.sel*(*2*) *rtranclp-dpll*$_W$*-inv*(*3*) *rtranclp-dpll*$_W$*-inv-starting-from-0*)
qed

6.2.3 Code export

A conversion to *DPLL-W-Implementation.dpll*$_W$*-state* **definition** *Con* :: (*int*, *unit*, *unit*) *ann-literal list* × *int literal list list*
⇒ *dpll*$_W$*-state* **where**
Con xs = *state-of* (*if dpll*$_W$*-all-inv* (*toS* (*fst xs*) (*snd xs*)) *then xs else* ([], []))
lemma [*code abstype*]:
Con (*rough-state-of S*) = *S*
using *rough-state-of*[*of S*] **unfolding** *Con-def* **by** *auto*

declare *rough-state-of-DPLL-step′-DPLL-step*[*code abstract*]

lemma *Con-DPLL-step-rough-state-of-state-of*[*simp*]:
Con (*DPLL-step* (*rough-state-of s*)) = *state-of* (*DPLL-step* (*rough-state-of s*))
unfolding *Con-def* **by** (*metis* (*mono-tags*, *lifting*) *DPLL-step-dpll*$_W$*-conc-inv mem-Collect-eq prod.case-eq-if*)

A slightly different version of *DPLL-tot* where the returned boolean indicates the result.

definition *DPLL-tot-rep* **where**
DPLL-tot-rep S =

$(let\ (M,\ N) = (rough\text{-}state\text{-}of\ (DPLL\text{-}tot\ S))\ in\ (\forall\ A \in set\ N.\ (\exists\ a{\in}set\ A.\ a \in lits\text{-}of\ (M)),\ M))$

One version of the generated SML code is here, but not included in the generated document. The only differences are:

- export *'a literal* from the SML Module *Clausal-Logic*;
- export the constructor *Con* from *DPLL-W-Implementation*;
- export the *int* constructor from *Arith*.

  All these allows to test on the code on some examples.

end
theory *CDCL-W-Implementation*
imports *DPLL-CDCL-W-Implementation CDCL-W-Termination*
begin

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

type-synonym $'a\ cdcl_W\text{-}mark = 'a\ clause$
type-synonym $cdcl_W\text{-}marked\text{-}level = nat$

type-synonym $'v\ cdcl_W\text{-}ann\text{-}literal = ('v,\ cdcl_W\text{-}marked\text{-}level,\ 'v\ cdcl_W\text{-}mark)\ ann\text{-}literal$
type-synonym $'v\ cdcl_W\text{-}ann\text{-}literals = ('v,\ cdcl_W\text{-}marked\text{-}level,\ 'v\ cdcl_W\text{-}mark)\ ann\text{-}literals$
type-synonym $'v\ cdcl_W\text{-}state =$
$'v\ cdcl_W\text{-}ann\text{-}literals \times 'v\ clauses \times 'v\ clauses \times nat \times 'v\ clause\ option$

abbreviation $trail :: 'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'a$ **where**
$trail \equiv (\lambda(M,\ \text{-}).\ M)$

abbreviation $cons\text{-}trail :: 'a \Rightarrow 'a\ list \times 'b \times 'c \times 'd \times 'e \Rightarrow 'a\ list \times 'b \times 'c \times 'd \times 'e$
where
$cons\text{-}trail \equiv (\lambda L\ (M,\ S).\ (L\#M,\ S))$

abbreviation $tl\text{-}trail :: 'a\ list \times 'b \times 'c \times 'd \times 'e \Rightarrow 'a\ list \times 'b \times 'c \times 'd \times 'e$ **where**
$tl\text{-}trail \equiv (\lambda(M,\ S).\ (tl\ M,\ S))$

abbreviation $clss :: 'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'b$ **where**
$clss \equiv \lambda(M,\ N,\ \text{-}).\ N$

abbreviation $learned\text{-}clss :: 'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'c$ **where**
$learned\text{-}clss \equiv \lambda(M,\ N,\ U,\ \text{-}).\ U$

abbreviation $backtrack\text{-}lvl :: 'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'd$ **where**
$backtrack\text{-}lvl \equiv \lambda(M,\ N,\ U,\ k,\ \text{-}).\ k$

abbreviation $update\text{-}backtrack\text{-}lvl :: 'd \Rightarrow 'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'a \times 'b \times 'c \times 'd \times 'e$
where
$update\text{-}backtrack\text{-}lvl \equiv \lambda k\ (M,\ N,\ U,\ \text{-},\ S).\ (M,\ N,\ U,\ k,\ S)$

abbreviation $conflicting :: 'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'e$ **where**
$conflicting \equiv \lambda(M,\ N,\ U,\ k,\ D).\ D$

abbreviation $update\text{-}conflicting :: 'e \Rightarrow 'a \times 'b \times 'c \times 'd \times 'e \Rightarrow 'a \times 'b \times 'c \times 'd \times 'e$
where

update-conflicting ≡ λ*S* (*M*, *N*, *U*, *k*, -). (*M*, *N*, *U*, *k*, *S*)

abbreviation *S0-cdcl*$_W$ *N* ≡ (([], *N*, {#}, *0*, *None*):: ′*v cdcl*$_W$*-state*)

abbreviation *add-learned-cls* **where**
add-learned-cls ≡ λ*C* (*M*, *N*, *U*, *S*). (*M*, *N*, {#*C*#} + *U*, *S*)

abbreviation *remove-cls* **where**
remove-cls ≡ λ*C* (*M*, *N*, *U*, *S*). (*M*, *remove-mset C N*, *remove-mset C U*, *S*)

lemma *trail-conv*: *trail* (*M*, *N*, *U*, *k*, *D*) = *M* **and**
clauses-conv: *clss* (*M*, *N*, *U*, *k*, *D*) = *N* **and**
learned-clss-conv: *learned-clss* (*M*, *N*, *U*, *k*, *D*) = *U* **and**
conflicting-conv: *conflicting* (*M*, *N*, *U*, *k*, *D*) = *D* **and**
backtrack-lvl-conv: *backtrack-lvl* (*M*, *N*, *U*, *k*, *D*) = *k*
by *auto*

lemma *state-conv*:
S = (*trail S*, *clss S*, *learned-clss S*, *backtrack-lvl S*, *conflicting S*)
by (*cases S*) *auto*

interpretation *state*$_W$ *trail clss learned-clss backtrack-lvl conflicting*
λ*L* (*M*, *S*). (*L* # *M*, *S*)
λ(*M*, *S*). (*tl M*, *S*)
λ*C* (*M*, *N*, *S*). (*M*, {#*C*#} + *N*, *S*)
λ*C* (*M*, *N*, *U*, *S*). (*M*, *N*, {#*C*#} + *U*, *S*)
λ*C* (*M*, *N*, *U*, *S*). (*M*, *remove-mset C N*, *remove-mset C U*, *S*)
λ(*k*::*nat*) (*M*, *N*, *U*, -, *D*). (*M*, *N*, *U*, *k*, *D*)
λ*D* (*M*, *N*, *U*, *k*, -). (*M*, *N*, *U*, *k*, *D*)
λ*N*. ([], *N*, {#}, *0*, *None*)
λ(-, *N*, *U*, -). ([], *N*, *U*, *0*, *None*)
by *unfold-locales auto*

interpretation *cdcl*$_W$ *trail clss learned-clss backtrack-lvl conflicting*
λ*L* (*M*, *S*). (*L* # *M*, *S*)
λ(*M*, *S*). (*tl M*, *S*)
λ*C* (*M*, *N*, *S*). (*M*, {#*C*#} + *N*, *S*)
λ*C* (*M*, *N*, *U*, *S*). (*M*, *N*, {#*C*#} + *U*, *S*)
λ*C* (*M*, *N*, *U*, *S*). (*M*, *remove-mset C N*, *remove-mset C U*, *S*)
λ(*k*::*nat*) (*M*, *N*, *U*, -, *D*). (*M*, *N*, *U*, *k*, *D*)
λ*D* (*M*, *N*, *U*, *k*, -). (*M*, *N*, *U*, *k*, *D*)
λ*N*. ([], *N*, {#}, *0*, *None*)
λ(-, *N*, *U*, -). ([], *N*, *U*, *0*, *None*)
by *unfold-locales auto*

declare *clauses-def*[*simp*]

lemma *cdcl*$_W$*-state-eq-equality*[*iff*]: *state-eq S T* ⟷ *S* = *T*
unfolding *state-eq-def* **by** (*cases S*, *cases T*) *auto*
declare *state-simp*[*simp del*]

6.3 CDCL Implementation

6.3.1 Definition of the rules

Types **lemma** *true-clss-remdups*[*simp*]:
  $I \models s$ (*mset* ∘ *remdups*) ‘ $N \longleftrightarrow I \models s$ *mset* ‘ N
  by (*simp add*: *true-clss-def*)

lemma *satisfiable-mset-remdups*[*simp*]:
  satisfiable ((*mset* ∘ *remdups*) ‘ N) ⟷ *satisfiable* (*mset* ‘ N)
unfolding *satisfiable-carac*[*symmetric*] **by** *simp*

value *backtrack-split* [*Marked* (*Pos* (*Suc 0*)) ()]
value ∃ C ∈ *set* [[*Pos* (*Suc 0*), *Neg* (*Suc 0*)]]. (∀ c ∈ *set* C. $-c$ ∈ *lits-of* [*Marked* (*Pos* (*Suc 0*)) ()])

type-synonym $cdcl_W$*-state-inv-st* = (*nat*, *nat*, *nat literal list*) *ann-literal list* ×
  nat literal list list × *nat literal list list* × *nat* × *nat literal list option*

We need some functions to convert between our abstract state *nat* $cdcl_W$*-state* and the concrete state $cdcl_W$*-state-inv-st*.

fun *convert* :: ('*a*, '*b*, '*c list*) *ann-literal* ⇒ ('*a*, '*b*, '*c multiset*) *ann-literal* **where**
convert (*Propagated L C*) = *Propagated L* (*mset C*) |
convert (*Marked K i*) = *Marked K i*

abbreviation *convertC* :: '*a list option* ⇒ '*a multiset option* **where**
convertC ≡ *map-option mset*

lemma *convert-Propagated*[*elim*!]:
  convert z = *Propagated L C* ⟹ (∃ C'. *z* = *Propagated L* C' ∧ C = *mset* C')
  by (*cases z*) *auto*

lemma *get-rev-level-map-convert*:
  get-rev-level (*map convert M*) *n x* = *get-rev-level M n x*
  by (*induction M arbitrary*: *n rule*: *ann-literal-list-induct*) *auto*

lemma *get-level-map-convert*[*simp*]:
  get-level (*map convert M*) = *get-level M*
  using *get-rev-level-map-convert*[*of rev M*] **by** (*simp add*: *rev-map*)

lemma *get-maximum-level-map-convert*[*simp*]:
  get-maximum-level (*map convert M*) *D* = *get-maximum-level M D*
  by (*induction D*)
    (*auto simp add*: *get-maximum-level-plus*)

lemma *get-all-levels-of-marked-map-convert*[*simp*]:
  get-all-levels-of-marked (*map convert M*) = (*get-all-levels-of-marked M*)
  by (*induction M rule*: *ann-literal-list-induct*) *auto*

Conversion function

fun *toS* :: $cdcl_W$*-state-inv-st* ⇒ *nat* $cdcl_W$*-state* **where**
toS (*M*, *N*, *U*, *k*, *C*) = (*map convert M*, *mset* (*map mset N*), *mset* (*map mset U*), *k*, *convertC C*)

Definition an abstract type

typedef $cdcl_W$*-state-inv* = {*S*::$cdcl_W$*-state-inv-st*. $cdcl_W$*-all-struct-inv* (*toS S*)}
  morphisms *rough-state-of state-of*
proof

show ([],[], [], *0*, *None*) $\in$ {*S*. *cdcl*$_W$*-all-struct-inv* (*toS S*)}
  by (*auto simp add*: *cdcl*$_W$*-all-struct-inv-def*)
qed

instantiation *cdcl*$_W$*-state-inv* :: *equal*
begin
definition *equal-cdcl*$_W$*-state-inv* :: *cdcl*$_W$*-state-inv* $\Rightarrow$ *cdcl*$_W$*-state-inv* $\Rightarrow$ *bool* **where**
equal-cdcl$_W$*-state-inv S S'* = (*rough-state-of S* = *rough-state-of S'*)
instance
  by *standard* (*simp add*: *rough-state-of-inject equal-cdcl*$_W$*-state-inv-def*)
end

lemma *lits-of-map-convert*[*simp*]: *lits-of* (*map convert M*) = *lits-of M*
  by (*induction M rule*: *ann-literal-list-induct*) *simp-all*

lemma *undefined-lit-map-convert*[*iff*]:
  undefined-lit (*map convert M*) *L* $\longleftrightarrow$ *undefined-lit M L*
  by (*auto simp add*: *Marked-Propagated-in-iff-in-lits-of*)

lemma *true-annot-map-convert*[*simp*]: *map convert M* $\models$*a N* $\longleftrightarrow$ *M* $\models$*a N*
  by (*induction M rule*: *ann-literal-list-induct*) (*simp-all add*: *true-annot-def*)

lemma *true-annots-map-convert*[*simp*]: *map convert M* $\models$*as N* $\longleftrightarrow$ *M* $\models$*as N*
  unfolding *true-annots-def* **by** *auto*

lemmas *propagateE*
lemma *find-first-unit-clause-some-is-propagate*:
  assumes *H*: *find-first-unit-clause* (*N* @ *U*) *M* = *Some* (*L*, *C*)
  shows *propagate* (*toS* (*M*, *N*, *U*, *k*, *None*)) (*toS* (*Propagated L C* # *M*, *N*, *U*, *k*, *None*))
  using *assms*
  by (*auto dest*!: *find-first-unit-clause-some simp add*: *propagate.simps*
    intro!: *exI*[*of - mset C* − {#*L*#}])

6.3.2 The Transitions

Propagate **definition** *do-propagate-step* **where**
do-propagate-step S =
  (*case S of*
    (*M*, *N*, *U*, *k*, *None*) $\Rightarrow$
      (*case find-first-unit-clause* (*N* @ *U*) *M of*
        Some (*L*, *C*) $\Rightarrow$ (*Propagated L C* # *M*, *N*, *U*, *k*, *None*)
      | *None* $\Rightarrow$ (*M*, *N*, *U*, *k*, *None*))
  | *S* $\Rightarrow$ *S*)

lemma *do-propgate-step*:
  do-propagate-step S $\neq$ *S* $\Longrightarrow$ *propagate* (*toS S*) (*toS* (*do-propagate-step S*))
  apply (*cases S*, *cases conflicting S*)
  using *find-first-unit-clause-some-is-propagate*[*of clss S learned-clss S trail S - -*
    backtrack-lvl S]
  by (*auto simp add*: *do-propagate-step-def split*: *option.splits*)

lemma *do-propagate-step-option*[*simp*]:
  conflicting S $\neq$ *None* $\Longrightarrow$ *do-propagate-step S* = *S*
  unfolding *do-propagate-step-def* **by** (*cases S*, *cases conflicting S*) *auto*

lemma *do-propagate-step-no-step*:

assumes *dist*: ∀ *c*∈*set* (*clss S* @ *learned-clss S*). *distinct c* **and**
prop-step: *do-propagate-step S* = *S*
shows *no-step propagate* (*toS S*)
proof (*standard*, *standard*)
fix *T*
assume *propagate* (*toS S*) *T*
then obtain *M N U k C L* **where**
toSS: *toS S* = (*M*, *N*, *U*, *k*, *None*) **and**
T: *T* = (*Propagated L* (*C* + {#*L*#}) # *M*, *N*, *U*, *k*, *None*) **and**
MC: *M* ⊨*as CNot C* **and**
undef: *undefined-lit M L* **and**
CL: *C* + {#*L*#} ∈# *N* + *U*
apply − **by** (*cases toS S*) *auto*
let *?M* = *trail S*
let *?N* = *clss S*
let *?U* = *learned-clss S*
let *?k* = *backtrack-lvl S*
let *?D* = *None*
have *S*: *S* = (*?M*, *?N*, *?U*, *?k*, *?D*)
using *toSS* **by** (*cases S*, *cases conflicting S*) *simp-all*
have *S*: *toS S* = *toS* (*?M*, *?N*, *?U*, *?k*, *?D*)
unfolding *S*[*symmetric*] **by** *simp*

have
M: *M* = *map convert ?M* **and**
N: *N* = *mset* (*map mset ?N*) **and**
U: *U* = *mset* (*map mset ?U*)
using *toSS*[*unfolded S*] **by** *auto*

obtain *D* **where**
DCL: *mset D* = *C* + {#*L*#} **and**
D: *D* ∈ *set* (*?N* @ *?U*)
using *CL* **unfolding** *N U* **by** *auto*
obtain *C′ L′* **where**
setD: *set D* = *set* (*L′* # *C′*) **and**
C′: *mset C′* = *C* **and**
L: *L* = *L′*
using *DCL* **by** (*metis ex-mset mset.simps*(*2*) *mset-eq-setD*)
have *find-first-unit-clause* (*?N* @ *?U*) *?M* ≠ *None*
apply (*rule dist find-first-unit-clause-none*[*of D ?N* @ *?U ?M L*, *OF - D*])
using *D assms*(*1*) **apply** *auto*[*1*]
using *MC setD DCL M MC* **unfolding** *C′*[*symmetric*] **apply** *auto*[*1*]
using *M undef* **apply** *auto*[*1*]
unfolding *setD L* **by** *auto*
then show *False* **using** *prop-step S* **unfolding** *do-propagate-step-def* **by** (*cases S*) *auto*
qed

Conflict **fun** *find-conflict* **where**
find-conflict M [] = *None* |
find-conflict M (*N* # *Ns*) = (*if* (∀ *c* ∈ *set N*. −*c* ∈ *lits-of M*) *then Some N else find-conflict M Ns*)

lemma *find-conflict-Some*:
find-conflict M Ns = *Some N* ⟹ *N* ∈ *set Ns* ∧ *M* ⊨*as CNot* (*mset N*)
by (*induction Ns rule*: *find-conflict.induct*)
(*auto split*: *split-if-asm*)

lemma *find-conflict-None*:
  find-conflict M Ns = None $\longleftrightarrow$ ($\forall$ *N* $\in$ *set Ns*. $\neg$*M* $\models$*as CNot* (*mset N*))
  by (*induction Ns*) *auto*

lemma *find-conflict-None-no-confl*:
  find-conflict M (*N*@*U*) = *None* $\longleftrightarrow$ *no-step conflict* (*toS* (*M*, *N*, *U*, *k*, *None*))
  by (*auto simp add*: *find-conflict-None conflict.simps*)

definition *do-conflict-step* **where**
do-conflict-step S =
  (*case S of*
    (*M*, *N*, *U*, *k*, *None*) $\Rightarrow$
      (*case find-conflict M* (*N* @ *U*) *of*
        Some a $\Rightarrow$ (*M*, *N*, *U*, *k*, *Some a*)
      | *None* $\Rightarrow$ (*M*, *N*, *U*, *k*, *None*))
  | *S* $\Rightarrow$ *S*)

lemma *do-conflict-step*:
  do-conflict-step S $\neq$ *S* $\Longrightarrow$ *conflict* (*toS S*) (*toS* (*do-conflict-step S*))
  apply (*cases S*, *cases conflicting S*)
  unfolding *conflict.simps do-conflict-step-def*
  by (*auto dest*!:*find-conflict-Some split*: *option.splits*)

lemma *do-conflict-step-no-step*:
  do-conflict-step S = *S* $\Longrightarrow$ *no-step conflict* (*toS S*)
  apply (*cases S*, *cases conflicting S*)
  unfolding *do-conflict-step-def*
  using *find-conflict-None-no-confl*[*of trail S clss S learned-clss S*
    backtrack-lvl S]
  by (*auto split*: *option.splits*)

lemma *do-conflict-step-option*[*simp*]:
  conflicting S $\neq$ *None* $\Longrightarrow$ *do-conflict-step S* = *S*
  unfolding *do-conflict-step-def* **by** (*cases S*, *cases conflicting S*) *auto*

lemma *do-conflict-step-conflicting*[*dest*]:
  do-conflict-step S $\neq$ *S* $\Longrightarrow$ *conflicting* (*do-conflict-step S*) $\neq$ *None*
  unfolding *do-conflict-step-def* **by** (*cases S*, *cases conflicting S*) (*auto split*: *option.splits*)

definition *do-cp-step* **where**
do-cp-step S =
  (*do-propagate-step o do-conflict-step*) *S*

lemma *cp-step-is-cdcl$_W$-cp*:
  assumes *H*: *do-cp-step S* $\neq$ *S*
  shows *cdcl$_W$-cp* (*toS S*) (*toS* (*do-cp-step S*))
proof −
  show *?thesis*
  proof (*cases do-conflict-step S* $\neq$ *S*)
    case *True*
    then show *?thesis*
      by (*auto simp add*: *do-conflict-step do-conflict-step-conflicting do-cp-step-def*)
  next
    case *False*

then have *confl*[*simp*]: *do-conflict-step S* = *S* **by** *simp*
show *?thesis*
proof (*cases do-propagate-step S* = *S*)
case *True*
then show *?thesis*
using *H* **by** (*simp add*: *do-cp-step-def*)
next
case *False*
let *?S* = *toS S*
let *?T* = *toS* (*do-propagate-step S*)
let *?U* = *toS* (*do-conflict-step* (*do-propagate-step S*))
have *propa*: *propagate* (*toS S*) *?T* **using** *False do-propgate-step* **by** *blast*
moreover have *ns*: *no-step conflict* (*toS S*) **using** *confl do-conflict-step-no-step* **by** *blast*
ultimately show *?thesis*
using $cdcl_W$*-cp.intros*(*2*)[*of ?S ?T*] *confl* **unfolding** *do-cp-step-def* **by** *auto*
qed
qed
qed

lemma *do-cp-step-eq-no-prop-no-confl*:
do-cp-step S = *S* $\Longrightarrow$ *do-conflict-step S* = *S* $\wedge$ *do-propagate-step S* = *S*
by (*cases S*, *cases conflicting S*)
(*auto simp add*: *do-conflict-step-def do-propagate-step-def do-cp-step-def split*: *option.splits*)

lemma *no-*$cdcl_W$*-cp-iff-no-propagate-no-conflict*:
no-step $cdcl_W$*-cp S* $\longleftrightarrow$ *no-step propagate S* $\wedge$ *no-step conflict S*
by (*auto simp*: $cdcl_W$*-cp.simps*)

lemma *do-cp-step-eq-no-step*:
assumes *H*: *do-cp-step S* = *S* **and** $\forall$ *c* $\in$ *set* (*clss S* @ *learned-clss S*). *distinct c*
shows *no-step* $cdcl_W$*-cp* (*toS S*)
unfolding *no-*$cdcl_W$*-cp-iff-no-propagate-no-conflict*
using *assms* **apply** (*cases S*, *cases conflicting S*)
using *do-propagate-step-no-step*[*of S*]
by (*auto dest*!: *do-cp-step-eq-no-prop-no-confl*[*simplified*] *do-conflict-step-no-step*
split: *option.splits*)

lemma $cdcl_W$*-cp-*$cdcl_W$*-st*: $cdcl_W$*-cp S S'* $\Longrightarrow$ $cdcl_W^{**}$ *S S'*
by (*simp add*: $cdcl_W$*-cp-tranclp-*$cdcl_W$ *tranclp-into-rtranclp*)

lemma $cdcl_W$*-cp-wf-all-inv*:
wf {(*S'*, *S*::*'v*::*linorder* $cdcl_W$*-state*). $cdcl_W$*-all-struct-inv S* $\wedge$ $cdcl_W$*-cp S S'*}
(**is** *wf ?R*)
proof (*rule wf-bounded-measure*[*of -* λS. *card* (*atms-of-msu* (*clss S*))+*1*
λS. *length* (*trail S*) + (*if conflicting S* = *None then 0 else 1*)], *goal-cases*)
case (*1 S S'*)
then have $cdcl_W$*-all-struct-inv S* **and** $cdcl_W$*-cp S S'* **by** *auto*
moreover then have $cdcl_W$*-all-struct-inv S'*
using *rtranclp-*$cdcl_W$*-all-struct-inv-inv* $cdcl_W$*-cp-*$cdcl_W$*-st* **by** *blast*
ultimately show *?case*
by (*auto simp*:$cdcl_W$*-cp.simps elim*!: *conflictE propagateE*
dest: *length-model-le-vars-all-inv*)
qed

lemma $cdcl_W$*-all-struct-inv-rough-state*[*simp*]: $cdcl_W$*-all-struct-inv* (*toS* (*rough-state-of S*))

using *rough-state-of* **by** *auto*

lemma [*simp*]: $cdcl_W$*-all-struct-inv* (*toS S*) $\Longrightarrow$ *rough-state-of* (*state-of S*) = *S*
by (*simp add*: *state-of-inverse*)

lemma *rough-state-of-state-of-do-cp-step*[*simp*]:
rough-state-of (*state-of* (*do-cp-step* (*rough-state-of S*))) = *do-cp-step* (*rough-state-of S*)
proof −
have $cdcl_W$*-all-struct-inv* (*toS* (*do-cp-step* (*rough-state-of S*)))
apply (*cases do-cp-step* (*rough-state-of S*) = (*rough-state-of S*))
apply *simp*
using *cp-step-is-*$cdcl_W$*-cp*[*of rough-state-of S*] $cdcl_W$*-all-struct-inv-rough-state*[*of S*]
$cdcl_W$*-cp-*$cdcl_W$*-st rtranclp-*$cdcl_W$*-all-struct-inv-inv* **by** *blast*
then show *?thesis* **by** *auto*
qed

Skip **fun** *do-skip-step* :: $cdcl_W$*-state-inv-st* ⇒ $cdcl_W$*-state-inv-st* **where**
do-skip-step (*Propagated L C # Ls*,*N*,*U*,*k*, *Some D*) =
(*if* −*L* ∉ *set D* ∧ *D* ≠ []
then (*Ls*, *N*, *U*, *k*, *Some D*)
else (*Propagated L C #Ls*, *N*, *U*, *k*, *Some D*)) |
do-skip-step S = *S*

lemma *do-skip-step*:
do-skip-step S ≠ *S* $\Longrightarrow$ *skip* (*toS S*) (*toS* (*do-skip-step S*))
apply (*induction S rule*: *do-skip-step.induct*)
by (*auto simp add*: *skip.simps*)

lemma *do-skip-step-no*:
do-skip-step S = *S* $\Longrightarrow$ *no-step skip* (*toS S*)
by (*induction S rule*: *do-skip-step.induct*)
(*auto simp add*: *other split*: *split-if-asm*)

lemma *do-skip-step-trail-is-None*[*iff*]:
do-skip-step S = (*a*, *b*, *c*, *d*, *None*) ⟷ *S* = (*a*, *b*, *c*, *d*, *None*)
by (*cases S rule*: *do-skip-step.cases*) *auto*

Resolve **fun** *maximum-level-code*:: *'a literal list* ⇒ (*'a*, *nat*, *'a literal list*) *ann-literal list* ⇒ *nat*
where
maximum-level-code [] - = *0* |
maximum-level-code (*L # Ls*) *M* = *max* (*get-level M L*) (*maximum-level-code Ls M*)

lemma *maximum-level-code-eq-get-maximum-level*[*code*, *simp*]:
maximum-level-code D M = *get-maximum-level M* (*mset D*)
by (*induction D*) (*auto simp add*: *get-maximum-level-plus*)

fun *do-resolve-step* :: $cdcl_W$*-state-inv-st* ⇒ $cdcl_W$*-state-inv-st* **where**
do-resolve-step (*Propagated L C # Ls*, *N*, *U*, *k*, *Some D*) =
(*if* −*L* ∈ *set D* ∧ *maximum-level-code* (*remove1* (−*L*) *D*) (*Propagated L C # Ls*) = *k*
then (*Ls*, *N*, *U*, *k*, *Some* (*remdups* (*remove1 L C* @ *remove1* (−*L*) *D*)))
else (*Propagated L C # Ls*, *N*, *U*, *k*, *Some D*)) |
do-resolve-step S = *S*

lemma *do-resolve-step*:
$cdcl_W$*-all-struct-inv* (*toS S*) $\Longrightarrow$ *do-resolve-step S* ≠ *S*

$\Longrightarrow$ *resolve* (*toS S*) (*toS* (*do-resolve-step S*))
proof (*induction S rule*: *do-resolve-step.induct*)
  case (*1 L C M N U k D*)
  then have
    $-$ *L* ∈ *set D* **and**
    M: *maximum-level-code* (*remove1* (−*L*) *D*) (*Propagated L C # M*) = *k*
    by (*cases mset D* − {#− *L*#} = {#},
      auto dest!: *get-maximum-level-exists-lit-of-max-level*[*of - Propagated L C # M*]
      split: *split-if-asm*)+
  have *every-mark-is-a-conflict* (*toS* (*Propagated L C # M*, *N*, *U*, *k*, *Some D*))
    using *1*(*1*) **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-conflicting-def* **by** *fast*
  then have *L* ∈ *set C* **by** *fastforce*
  then obtain *C′* **where** *C*: *mset C* = *C′* + {#*L*#}
    by (*metis add.commute in-multiset-in-set insert-DiffM*)
  obtain *D′* **where** *D*: *mset D* = *D′* + {#−*L*#}
    using ‹− *L* ∈ *set D*› **by** (*metis add.commute in-multiset-in-set insert-DiffM*)
  have *D′L*: *D′* + {#− *L*#} − {#−*L*#} = *D′* **by** (*auto simp add*: *multiset-eq-iff*)

  have *CL*: *mset C* − {#*L*#} + {#*L*#} = *mset C* **using** ‹*L* ∈ *set C*› **by** (*auto simp add*: *multiset-eq-iff*)
  have *get-maximum-level* (*Propagated L* (*C′* + {#*L*#}) *# map convert M*) *D′* = *k*
    using *M*[*simplified*] **unfolding** *maximum-level-code-eq-get-maximum-level C*[*symmetric*] *CL*
    by (*metis D D′L convert.simps*(*1*) *get-maximum-level-map-convert list.simps*(*9*))
  then have
    resolve
      (*map convert* (*Propagated L C # M*), *mset ‘# mset N*, *mset ‘# mset U*, *k*, *Some* (*mset D*))
      (*map convert M*, *mset ‘# mset N*, *mset ‘# mset U*, *k*,
        Some (((*mset D* − {#−*L*#}) #∪ (*mset C* − {#*L*#}))))
    unfolding *resolve.simps*
      by (*simp add*: *C D*)
  moreover have
    (*map convert* (*Propagated L C # M*), *mset ‘# mset N*, *mset ‘# mset U*, *k*, *Some* (*mset D*))
     = *toS* (*Propagated L C # M*, *N*, *U*, *k*, *Some D*)
    by (*auto simp*: *mset-map*)
  moreover
    have *distinct-mset* (*mset C*) **and** *distinct-mset* (*mset D*)
      using ‹$cdcl_W$*-all-struct-inv* (*toS* (*Propagated L C # M*, *N*, *U*, *k*, *Some D*))›
      unfolding $cdcl_W$*-all-struct-inv-def distinct-*$cdcl_W$*-state-def*
      by *auto*
    then have (*mset C* − {#*L*#}) #∪ (*mset D* − {#− *L*#}) =
      remdups-mset (*mset C* − {#*L*#} + (*mset D* − {#− *L*#}))
      by (*auto simp*: *distinct-mset-rempdups-union-mset*)
    then have (*map convert M*, *mset ‘# mset N*, *mset ‘# mset U*, *k*,
    Some ((*mset D* − {#− *L*#}) #∪ (*mset C* − {#*L*#})))
    = *toS* (*do-resolve-step* (*Propagated L C # M*, *N*, *U*, *k*, *Some D*))
    using ‹− *L* ∈ *set D*› *M* **by** (*auto simp*:*ac-simps mset-map*)
  ultimately show *?case*
    by *simp*
qed *auto*

lemma *do-resolve-step-no*:
  do-resolve-step S = *S* $\Longrightarrow$ *no-step resolve* (*toS S*)
  apply (*cases S*; *cases hd* (*trail S*); *cases conflicting S*)
  by (*auto*
    elim!: *resolveE* *split*: *split-if-asm*
    dest!: *union-single-eq-member*

simp del: *in-multiset-in-set get-maximum-level-map-convert*
simp: *in-multiset-in-set*[*symmetric*] *get-maximum-level-map-convert*[*symmetric*])

lemma *rough-state-of-state-of-resolve*[*simp*]:
$cdcl_W$*-all-struct-inv* (*toS S*) $\Longrightarrow$ *rough-state-of* (*state-of* (*do-resolve-step S*)) = *do-resolve-step S*
apply (*rule state-of-inverse*)
apply (*cases do-resolve-step S* = *S*)
apply *simp*
by (*blast dest*: *other resolve bj do-resolve-step* $cdcl_W$*-all-struct-inv-inv*)

lemma *do-resolve-step-trail-is-None*[*iff*]:
do-resolve-step S = (*a*, *b*, *c*, *d*, *None*) $\longleftrightarrow$ *S* = (*a*, *b*, *c*, *d*, *None*)
by (*cases S rule*: *do-resolve-step.cases*) *auto*

Backjumping **fun** *find-level-decomp* **where**
find-level-decomp M [] *D k* = *None* |
find-level-decomp M (*L* # *Ls*) *D k* =
(*case* (*get-level M L*, *maximum-level-code* (*D* @ *Ls*) *M*) *of*
(*i*, *j*) $\Rightarrow$ *if i* = *k* $\wedge$ *j* < *i then Some* (*L*, *j*) *else find-level-decomp M Ls* (*L*#*D*) *k*
)

lemma *find-level-decomp-some*:
assumes *find-level-decomp M Ls D k* = *Some* (*L*, *j*)
shows *L* $\in$ *set Ls* $\wedge$ *get-maximum-level M* (*mset* (*remove1 L* (*Ls* @ *D*))) = *j* $\wedge$ *get-level M L* = *k*
using *assms*
proof (*induction Ls arbitrary*: *D*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons L' Ls*) **note** *IH* = *this*(*1*) **and** *H* = *this*(*2*)

def *find* ≡ (*if get-level M L'* ≠ *k* ∨ ¬ *get-maximum-level M* (*mset D* + *mset Ls*) < *get-level M L'*
then find-level-decomp M Ls (*L'* # *D*) *k*
else Some (*L'*, *get-maximum-level M* (*mset D* + *mset Ls*)))
have *a1*: ⋀*D*. *find-level-decomp M Ls D k* = *Some* (*L*, *j*) $\Longrightarrow$
L ∈ *set Ls* ∧ *get-maximum-level M* (*mset Ls* + *mset D* − {#*L*#}) = *j* ∧ *get-level M L* = *k*
using *IH* **by** *simp*
have *a2*: *find* = *Some* (*L*, *j*)
using *H* **unfolding** *find-def* **by** (*auto split*: *split-if-asm*)
{ assume *Some* (*L'*, *get-maximum-level M* (*mset D* + *mset Ls*)) ≠ *find*
then have *f3*: *L* ∈ *set Ls* **and** *get-maximum-level M* (*mset Ls* + *mset* (*L'* # *D*) − {#*L*#}) = *j*
using *a1 IH a2* **unfolding** *find-def* **by** *meson+*
moreover then have *mset Ls* + *mset D* − {#*L*#} + {#*L'*#} = {#*L'*#} + *mset D* + (*mset Ls* − {#*L*#})
by (*auto simp*: *ac-simps multiset-eq-iff Suc-leI*)
ultimately have *f4*: *get-maximum-level M* (*mset Ls* + *mset D* − {#*L*#} + {#*L'*#}) = *j*
by (*metis* (*no-types*) *diff-union-single-conv mem-set-multiset-eq mset.simps*(*2*) *union-commute*)
} note *f4* = *this*
have {#*L'*#} + (*mset Ls* + *mset D*) = *mset Ls* + (*mset D* + {#*L'*#})
by (*auto simp*: *ac-simps*)
then have
(*L* = *L'* ⟶ *get-maximum-level M* (*mset Ls* + *mset D*) = *j* ∧ *get-level M L'* = *k*) **and**
(*L* ≠ *L'* ⟶ *L* ∈ *set Ls* ∧ *get-maximum-level M* (*mset Ls* + *mset D* − {#*L*#} + {#*L'*#}) = *j* ∧
get-level M L = *k*)

using *f4 a2 a1*[*of* L' # *D*] **unfolding** *find-def* **by** (*metis* (*no-types*) *add-diff-cancel-left'* *mset.simps*(*2*) *option.inject prod.inject union-commute*)+
then show *?case* **by** *simp*
qed

lemma *find-level-decomp-none*:
assumes *find-level-decomp M Ls E k* = *None* **and** *mset* (*L#D*) = *mset* (*Ls* @ *E*)
shows $\neg$(*L* $\in$ *set Ls* $\wedge$ *get-maximum-level M* (*mset D*) < *k* $\wedge$ *k* = *get-level M L*)
using *assms*
proof (*induction Ls arbitrary*: *E L D*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons* L' *Ls*) **note** *IH* = *this*(*1*) **and** *find-none* = *this*(*2*) **and** *LD* = *this*(*3*)
have *mset D* + {#L'#} = *mset E* + (*mset Ls* + {#L'#}) $\Longrightarrow$ *mset D* = *mset E* + *mset Ls*
by (*metis add-right-imp-eq union-assoc*)
then show *?case*
using *find-none IH*[*of* L' # *E L D*] *LD* **by** (*auto simp add*: *ac-simps split*: *split-if-asm*)
qed

fun *bt-cut* **where**
bt-cut i (*Propagated - -* # *Ls*) = *bt-cut i Ls* |
bt-cut i (*Marked K k* # *Ls*) = (*if k* = *Suc i then Some* (*Marked K k* # *Ls*) *else bt-cut i Ls*) |
bt-cut i [] = *None*

lemma *bt-cut-some-decomp*:
bt-cut i M = *Some* M' $\Longrightarrow$ $\exists$ *K M2 M1*. *M* = *M2* @ M' $\wedge$ M' = *Marked K* (*i+1*) # *M1*
by (*induction i M rule*: *bt-cut.induct*) (*auto split*: *split-if-asm*)

lemma *bt-cut-not-none*: *M* = *M2* @ *Marked K* (*Suc i*) # M' $\Longrightarrow$ *bt-cut i M* $\neq$ *None*
by (*induction M2 arbitrary*: *M rule*: *ann-literal-list-induct*) *auto*

lemma *get-all-marked-decomposition-ex*:
$\exists$ *N*. (*Marked K* (*Suc i*) # M', *N*) $\in$ *set* (*get-all-marked-decomposition* (*M2*@*Marked K* (*Suc i*) # M'))
apply (*induction M2 rule*: *ann-literal-list-induct*)
apply *auto*[*2*]
by (*rename-tac L m xs*, *case-tac get-all-marked-decomposition* (*xs* @ *Marked K* (*Suc i*) # M'))
auto

lemma *bt-cut-in-get-all-marked-decomposition*:
bt-cut i M = *Some* M' $\Longrightarrow$ $\exists$ *M2*. (M', *M2*) $\in$ *set* (*get-all-marked-decomposition M*)
by (*auto dest*!: *bt-cut-some-decomp simp add*: *get-all-marked-decomposition-ex*)

fun *do-backtrack-step* **where**
do-backtrack-step (*M*, *N*, *U*, *k*, *Some D*) =
(*case find-level-decomp M D* [] *k of*
None ⇒ (*M*, *N*, *U*, *k*, *Some D*)
| *Some* (*L*, *j*) ⇒
(*case bt-cut j M of*
Some (*Marked - -* # *Ls*) ⇒ (*Propagated L D* # *Ls*, *N*, *D* # *U*, *j*, *None*)
| *-* ⇒ (*M*, *N*, *U*, *k*, *Some D*))
) |
do-backtrack-step S = *S*

lemma *get-all-marked-decomposition-map-convert*:
  (*get-all-marked-decomposition* (*map convert M*)) =
    map (λ(*a*, *b*). (*map convert a*, *map convert b*)) (*get-all-marked-decomposition M*)
  apply (*induction M rule*: *ann-literal-list-induct*)
    apply *simp*
  by (*rename-tac L l xs*, *case-tac get-all-marked-decomposition xs*; *auto*)+

lemma *do-backtrack-step*:
  assumes
    db: *do-backtrack-step S* ≠ *S* **and**
    inv: $cdcl_W$*-all-struct-inv* (*toS S*)
  shows *backtrack* (*toS S*) (*toS* (*do-backtrack-step S*))
  proof (*cases S*, *cases conflicting S*, *goal-cases*)
    case (*1 M N U k E*)
    then show *?case* **using** *db* **by** *auto*
  next
    case (*2 M N U k E C*) **note** *S* = *this*(*1*) **and** *confl* = *this*(*2*)
    have *E*: *E* = *Some C* **using** *S confl* **by** *auto*

    obtain *L j* **where** *fd*: *find-level-decomp M C* [] *k* = *Some* (*L*, *j*)
      using *db* **unfolding** *S E* **by** (*cases C*) (*auto split*: *split-if-asm option.splits*)
    have *L* ∈ *set C* **and** *get-maximum-level M* (*mset* (*remove1 L C*)) = *j* **and**
      levL: *get-level M L* = *k*
      using *find-level-decomp-some*[*OF fd*] **by** *auto*
    obtain *C'* **where** *C*: *mset C* = *mset C'* + {#*L*#}
      using ‹*L* ∈ *set C*› **by** (*metis add.commute ex-mset in-multiset-in-set insert-DiffM*)
    obtain M_2 **where** M_2: *bt-cut j M* = *Some* M_2
      using *db fd* **unfolding** *S E* **by** (*auto split*: *option.splits*)
    obtain *M1 K* **where** *M1*: M_2 = *Marked K* (*Suc j*) # *M1*
      using *bt-cut-some-decomp*[*OF* M_2] **by** (*cases* M_2) *auto*
    obtain *c* **where** *c*: *M* = *c* @ *Marked K* (*Suc j*) # *M1*
       using *bt-cut-in-get-all-marked-decomposition*[*OF* M_2]
       unfolding *M1* **by** *fastforce*
    have *get-all-levels-of-marked* (*map convert M*) = *rev* [*1..<Suc k*]
      using *inv* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-def S* **by** *auto*
    from *arg-cong*[*OF this*, *of* λ*a*. *Suc j* ∈ *set a*] **have** *j* ≤ *k* **unfolding** *c* **by** *auto*
    have *max-l-j*: *maximum-level-code C' M* = *j*
      using *db fd* M_2 *C* **unfolding** *S E* **by** (*auto*
          split: *option.splits list.splits ann-literal.splits*
          dest!: *find-level-decomp-some*)[*1*]
    have *get-maximum-level M* (*mset C*) ≥ *k*
      using ‹*L* ∈ *set C*› *get-maximum-level-ge-get-level levL* **by** *blast*
    moreover have *get-maximum-level M* (*mset C*) ≤ *k*
      using *get-maximum-level-exists-lit-of-max-level*[*of mset C M*] *inv*
        $cdcl_W$*-M-level-inv-get-level-le-backtrack-lvl*[*of toS S*]
      unfolding *C* $cdcl_W$*-all-struct-inv-def S* **by** (*auto dest*: *sym*[*of get-level - -*])
    ultimately have *get-maximum-level M* (*mset C*) = *k* **by** *auto*

    obtain *M2* **where** *M2*: (M_2, *M2*) ∈ *set* (*get-all-marked-decomposition M*)
      using *bt-cut-in-get-all-marked-decomposition*[*OF* M_2] **by** *metis*
    have *H*: (*reduce-trail-to* (*map convert M1*)
      (*add-learned-cls* (*mset C'* + {#*L*#})
        (*map convert M*, *mset* (*map mset N*), *mset* (*map mset U*), *j*, *None*))) =
       (*map convert M1*, *mset* (*map mset N*), {#*mset C'* + {#*L*#}#} + *mset* (*map mset U*), *j*, *None*)
        apply (*subst state-conv*[*of reduce-trail-to - -*])

using *M2* **unfolding** *M1* **by** *auto*
have
backtrack
(*map convert M*, *mset '# mset N*, *mset '# mset U*, *k*, *Some* (*mset C*))
(*Propagated L* (*mset C*) # *map convert M1*, *mset '# mset N*, *mset '# mset U* + {#*mset C*#},
j,
None)
apply (*rule backtrack-rule*)
unfolding *C* **apply** *simp*
using *Set.imageI*[*of* (M_2, *M2*) *set* (*get-all-marked-decomposition M*)
(λ(*a*, *b*). (*map convert a*, *map convert b*))] *M2*
apply (*auto simp*: *get-all-marked-decomposition-map-convert M1*)[*1*]
using *max-l-j levL* ‹$j \leq k$› **apply** (*simp add*: *get-maximum-level-plus*)
using *C* ‹*get-maximum-level M* (*mset C*) = *k*› *levL* **apply** *auto*[*1*]
using *max-l-j* **apply** *simp*
apply (*cases reduce-trail-to* (*map convert M1*)
(*add-learned-cls* (*mset C'* + {#*L*#})
(*map convert M*, *mset* (*map mset N*), *mset* (*map mset U*), *j*, *None*)))
using *M2 M1 H* **by** (*auto simp*: *ac-simps mset-map*)
then show *?case*
using M_2 *fd* **unfolding** *S E M1* **by** (*auto simp*: *mset-map*)
obtain *M2* **where** (M_2, *M2*) ∈ *set* (*get-all-marked-decomposition M*)
using *bt-cut-in-get-all-marked-decomposition*[*OF* M_2] **by** *metis*
qed

lemma *do-backtrack-step-no*:
assumes *db*: *do-backtrack-step S* = *S*
and *inv*: $cdcl_W$*-all-struct-inv* (*toS S*)
shows *no-step backtrack* (*toS S*)
proof (*rule ccontr*, *cases S*, *cases conflicting S*, *goal-cases*)
case *1*
then show *?case* **using** *db* **by** (*auto split*: *option.splits*)
next
case (*2 M N U k E C*) **note** *bt* = *this*(*1*) **and** *S* = *this*(*2*) **and** *confl* = *this*(*3*)
obtain *D L K b z M1 j* **where**
levL: *get-level M L* = *get-maximum-level M* (*D* + {#*L*#}) **and**
k: *k* = *get-maximum-level M* (*D* + {#*L*#}) **and**
j: *j* = *get-maximum-level M D* **and**
CE: *convertC E* = *Some* (*D* + {#*L*#}) **and**
decomp: (*z* # *M1*, *b*) ∈ *set* (*get-all-marked-decomposition M*) **and**
z: *Marked K* (*Suc j*) = *convert z* **using** *bt* **unfolding** *S*
by (*auto split*: *option.splits elim*!: *backtrackE*
simp: *get-all-marked-decomposition-map-convert*)
have *z*: *z* = *Marked K* (*Suc j*) **using** *z* **by** (*cases z*) *auto*
obtain *c* **where** *c*: *M* = *c* @ *b* @ *Marked K* (*Suc j*) # *M1*
using *decomp* **unfolding** *z* **by** *blast*
have *get-all-levels-of-marked* (*map convert M*) = *rev* [*1*..<*Suc k*]
using *inv* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-def S* **by** *auto*
from *arg-cong*[*OF this*, *of* λ*a*. *Suc j* ∈ *set a*] **have** *k* > *j* **unfolding** *c* **by** *auto*
obtain *C D'* **where**
E: *E* = *Some C* **and**
C: *mset C* = *mset* (*L* # *D'*)
using *CE* **apply** (*cases E*)
apply *simp*
by (*metis ex-mset mset.simps*(*2*) *option.inject option.simps*(*9*))

have $D'D$: *mset* $D' = D$
  using *C CE E* by *auto*
have *find-level-decomp M C* [] $k \neq$ *None*
  apply *rule*
  apply (*drule find-level-decomp-none*[*of - - - - L* D'])
  using *C* ‹$k > j$› *mset-eq-setD* unfolding *k*[*symmetric*] $D'D$ *j*[*symmetric*] *levL* by *fastforce+*
then obtain L' j' where *fd-some*: *find-level-decomp M C* [] k = *Some* (L', j')
  by (*cases find-level-decomp M C* [] *k*) *auto*
have L': $L' = L$
  proof (*rule ccontr*)
    assume ¬ *?thesis*
    then have $L' \in\# D$
      by (*metis C* $D'D$ *fd-some find-level-decomp-some in-multiset-in-set insert-iff list.simps*(*15*))
    then have *get-level M* $L' \leq$ *get-maximum-level M D*
      using *get-maximum-level-ge-get-level* by *blast*
    then show *False* using ‹$k > j$› *j find-level-decomp-some*[*OF fd-some*] by *auto*
  qed
then have j': $j' = j$ using *find-level-decomp-some*[*OF fd-some*] *j C* $D'D$ by *auto*

have *btc-none*: *bt-cut j M* $\neq$ *None*
  apply (*rule bt-cut-not-none*[*of M* - @ -])
  using *c* by *simp*
show *?case* using *db* unfolding *S E*
  by (*auto split*: *option.splits list.splits ann-literal.splits*
    simp add: *fd-some* L' j' *btc-none*
    dest: *bt-cut-some-decomp*)
qed

lemma *rough-state-of-state-of-backtrack*[*simp*]:
  assumes *inv*: $cdcl_W$*-all-struct-inv* (*toS S*)
  shows *rough-state-of* (*state-of* (*do-backtrack-step S*))= *do-backtrack-step S*
proof (*rule state-of-inverse*)
  have *f2*: *backtrack* (*toS S*) (*toS* (*do-backtrack-step S*)) ∨ *do-backtrack-step S* = *S*
    using *do-backtrack-step inv* by *blast*
  have $\bigwedge p$. ¬ $cdcl_W$*-o* (*toS S*) *p* ∨ $cdcl_W$*-all-struct-inv p*
    using *inv* $cdcl_W$*-all-struct-inv-inv other* by *blast*
  then have *do-backtrack-step S* = *S* ∨ $cdcl_W$*-all-struct-inv* (*toS* (*do-backtrack-step S*))
    using *f2* by *blast*
  then show *do-backtrack-step S* ∈ {*S*. $cdcl_W$*-all-struct-inv* (*toS S*)}
    using *inv* by *fastforce*
qed

Decide fun *do-decide-step* where
do-decide-step (*M*, *N*, *U*, *k*, *None*) =
  (*case find-first-unused-var N* (*lits-of M*) *of*
    None ⇒ (*M*, *N*, *U*, *k*, *None*)
  | *Some L* ⇒ (*Marked L* (*Suc k*) # *M*, *N*, *U*, *k+1*, *None*)) |
do-decide-step S = *S*

lemma *do-decide-step*:
  do-decide-step S $\neq$ *S* $\Longrightarrow$ *decide* (*toS S*) (*toS* (*do-decide-step S*))
  apply (*cases S*, *cases conflicting S*)
  defer
  apply (*auto split*: *option.splits simp add*: *decide.simps Marked-Propagated-in-iff-in-lits-of*
      dest: *find-first-unused-var-undefined find-first-unused-var-Some*

    intro: *atms-of-atms-of-ms-mono*)[*1*]
proof −
  fix *a* :: (*nat*, *nat*, *nat literal list*) *ann-literal list* **and**
      b :: *nat literal list list* **and** *c* :: *nat literal list list* **and**
      d :: *nat* **and** *e* :: *nat literal list option*
  {
    fix *a* :: (*nat*, *nat*, *nat literal list*) *ann-literal list* **and**
      b :: *nat literal list list* **and** *c* :: *nat literal list list* **and**
      d :: *nat* **and** *x2* :: *nat literal* **and** *m* :: *nat literal list*
    assume *a1*: $m \in set\ b$
    assume $x2 \in set\ m$
    then have *f2*: $atm\text{-}of\ x2 \in atms\text{-}of\ (mset\ m)$
      by *simp*
    have $\bigwedge f.\ (f\ m{::}nat\ literal\ multiset) \in f\ `\ set\ b$
      using *a1* **by** *blast*
    then have $\bigwedge f.\ (atms\text{-}of\ (f\ m){::}nat\ set) \subseteq atms\text{-}of\text{-}ms\ (f\ `\ set\ b)$
     using *atms-of-atms-of-ms-mono* **by** *blast*
    then have $\bigwedge n\ f.\ (n{::}nat) \in atms\text{-}of\text{-}ms\ (f\ `\ set\ b) \vee n \notin atms\text{-}of\ (f\ m)$
      by (*meson contra-subsetD*)
    then have $atm\text{-}of\ x2 \in atms\text{-}of\text{-}ms\ (mset\ `\ set\ b)$
      using *f2* **by** *blast*
  } **note** $H = this$
  {
    fix *m* :: *nat literal list* **and** *x2*
    have $m \in set\ b \Longrightarrow x2 \in set\ m \Longrightarrow x2 \notin lits\text{-}of\ a \Longrightarrow -\ x2 \notin lits\text{-}of\ a \Longrightarrow$
      $\exists\, aa{\in}set\ b.\ \neg\ atm\text{-}of\ `\ set\ aa \subseteq atm\text{-}of\ `\ lits\text{-}of\ a$
      by (*meson atm-of-in-atm-of-set-in-uminus contra-subsetD rev-image-eqI*)
  } **note** $H' = this$

  assume $do\text{-}decide\text{-}step\ S \neq S$ **and**
     $S = (a, b, c, d, e)$ **and**
     $conflicting\ S = None$
  then show *decide* (*toS S*) (*toS* (*do-decide-step S*))
    using *H H′* **by** (*auto split*: *option.splits simp*: *decide.simps Marked-Propagated-in-iff-in-lits-of*
      dest!: *find-first-unused-var-Some*)
qed

lemma *do-decide-step-no*:
  $do\text{-}decide\text{-}step\ S = S \Longrightarrow no\text{-}step\ decide\ (toS\ S)$
  by (*cases S*, *cases conflicting S*)
   (*fastforce simp*: *atms-of-ms-mset-unfold atm-of-eq-atm-of Marked-Propagated-in-iff-in-lits-of*
     split: *option.splits*)+

lemma *rough-state-of-state-of-do-decide-step*[*simp*]:
  $cdcl_W\text{-}all\text{-}struct\text{-}inv\ (toS\ S) \Longrightarrow rough\text{-}state\text{-}of\ (state\text{-}of\ (do\text{-}decide\text{-}step\ S)) = do\text{-}decide\text{-}step\ S$
proof (*subst state-of-inverse*, *goal-cases*)
  case *1*
  then show *?case*
    by (*cases do-decide-step S = S*)
     (*auto dest*: *do-decide-step decide other intro*: $cdcl_W$*-all-struct-inv-inv*)
qed *simp*

lemma *rough-state-of-state-of-do-skip-step*[*simp*]:
  $cdcl_W\text{-}all\text{-}struct\text{-}inv\ (toS\ S) \Longrightarrow rough\text{-}state\text{-}of\ (state\text{-}of\ (do\text{-}skip\text{-}step\ S)) = do\text{-}skip\text{-}step\ S$
  apply (*subst state-of-inverse*, *cases do-skip-step S = S*)

apply *simp*
by (*blast dest*: *other skip bj do-skip-step* $cdcl_W$*-all-struct-inv-inv*)+

6.3.3 Code generation

Type definition There are two invariants: one while applying conflict and propagate and one for the other rules

declare *rough-state-of-inverse*[*simp add*]
definition *Con* **where**
Con xs = *state-of* (*if* $cdcl_W$*-all-struct-inv* (*toS* (*fst xs*, *snd xs*)) *then xs*
else ([], [], [], *0*, *None*))

lemma [*code abstype*]:
Con (*rough-state-of S*) = *S*
using *rough-state-of*[*of S*] **unfolding** *Con-def* **by** *simp*

definition *do-cp-step'* **where**
do-cp-step' S = *state-of* (*do-cp-step* (*rough-state-of S*))

typedef $cdcl_W$*-state-inv-from-init-state* = {*S*::$cdcl_W$*-state-inv-st*. $cdcl_W$*-all-struct-inv* (*toS S*)
∧ $cdcl_W$*-stgy*** (*S0-*$cdcl_W$ (*clss* (*toS S*))) (*toS S*)}
morphisms *rough-state-from-init-state-of state-from-init-state-of*
proof
show ([],[], [], *0*, *None*) ∈ {*S*. $cdcl_W$*-all-struct-inv* (*toS S*)
∧ $cdcl_W$*-stgy*** (*S0-*$cdcl_W$ (*clss* (*toS S*))) (*toS S*)}
by (*auto simp add*: $cdcl_W$*-all-struct-inv-def*)
qed

instantiation $cdcl_W$*-state-inv-from-init-state* :: *equal*
begin
definition *equal-*$cdcl_W$*-state-inv-from-init-state* :: $cdcl_W$*-state-inv-from-init-state* ⇒
$cdcl_W$*-state-inv-from-init-state* ⇒ *bool* **where**
equal-$cdcl_W$*-state-inv-from-init-state S S'* ⟷
(*rough-state-from-init-state-of S* = *rough-state-from-init-state-of S'*)
instance
by *standard* (*simp add*: *rough-state-from-init-state-of-inject*
equal-$cdcl_W$*-state-inv-from-init-state-def*)
end

definition *ConI* **where**
ConI S = *state-from-init-state-of* (*if* $cdcl_W$*-all-struct-inv* (*toS* (*fst S*, *snd S*))
∧ $cdcl_W$*-stgy*** (*S0-*$cdcl_W$ (*clss* (*toS S*))) (*toS S*) *then S else* ([], [], [], *0*, *None*))

lemma [*code abstype*]:
ConI (*rough-state-from-init-state-of S*) = *S*
using *rough-state-from-init-state-of*[*of S*] **unfolding** *ConI-def*
by (*simp add*: *rough-state-from-init-state-of-inverse*)

definition *id-of-I-to*:: $cdcl_W$*-state-inv-from-init-state* ⇒ $cdcl_W$*-state-inv* **where**
id-of-I-to S = *state-of* (*rough-state-from-init-state-of S*)

lemma [*code abstract*]:
rough-state-of (*id-of-I-to S*) = *rough-state-from-init-state-of S*
unfolding *id-of-I-to-def* **using** *rough-state-from-init-state-of* **by** *auto*

Conflict and Propagate **function** *do-full1-cp-step* :: $cdcl_W$*-state-inv* ⇒ $cdcl_W$*-state-inv* **where**
do-full1-cp-step S =
  (*let S′ = do-cp-step′ S in*
   if S = S′ then S else do-full1-cp-step S′)
by *auto*
termination
proof (*relation* {(*T′*, *T*). (*rough-state-of T′*, *rough-state-of T*) ∈ {(*S′*, *S*).
  (*toS S′*, *toS S*) ∈ {(*S′*, *S*). $cdcl_W$*-all-struct-inv S* ∧ $cdcl_W$*-cp S S′*}}}, *goal-cases*)
  case *1*
  show *?case*
    using *wf-if-measure-f*[*OF wf-if-measure-f*[*OF* $cdcl_W$*-cp-wf-all-inv*, *of toS*], *of rough-state-of*] **.**
next
  case (*2 S′ S*)
  then show *?case*
    unfolding *do-cp-step′-def*
    apply *simp*
    by (*metis cp-step-is-*$cdcl_W$*-cp rough-state-of-inverse*)
qed

lemma *do-full1-cp-step-fix-point-of-do-full1-cp-step*:
  do-cp-step(*rough-state-of* (*do-full1-cp-step S*)) = (*rough-state-of* (*do-full1-cp-step S*))
  by (*rule do-full1-cp-step.induct*[*of* λ*S*. *do-cp-step*(*rough-state-of* (*do-full1-cp-step S*))
      = (*rough-state-of* (*do-full1-cp-step S*))]])
    (*metis* (*full-types*) *do-full1-cp-step.elims rough-state-of-state-of-do-cp-step do-cp-step′-def*)

lemma *in-clauses-rough-state-of-is-distinct*:
  c∈*set* (*clss* (*rough-state-of S*) @ *learned-clss* (*rough-state-of S*)) ⟹ *distinct c*
  apply (*cases rough-state-of S*)
  using *rough-state-of*[*of S*] **by** (*auto simp add*: *distinct-mset-set-distinct* $cdcl_W$*-all-struct-inv-def*
    distinct-$cdcl_W$*-state-def*)

lemma *do-full1-cp-step-full*:
  full $cdcl_W$*-cp* (*toS* (*rough-state-of S*))
    (*toS* (*rough-state-of* (*do-full1-cp-step S*)))
  unfolding *full-def*
proof (*rule conjI*, *induction S rule*: *do-full1-cp-step.induct*)
  case (*1 S*)
  then have *f1*:
      $cdcl_W$*-cp*** (*toS* (*do-cp-step* (*rough-state-of S*))) (
        toS (*rough-state-of* (*do-full1-cp-step* (*state-of* (*do-cp-step* (*rough-state-of S*))))))
      ∨ *state-of* (*do-cp-step* (*rough-state-of S*)) = *S*
    using *do-cp-step′-def rough-state-of-state-of-do-cp-step* **by** *fastforce*
  have *f2*: ⋀*c*. (*if c = state-of* (*do-cp-step* (*rough-state-of c*))
      then c else do-full1-cp-step (*state-of* (*do-cp-step* (*rough-state-of c*))))
     = *do-full1-cp-step c*
    by (*metis* (*full-types*) *do-cp-step′-def do-full1-cp-step.simps*)
  have *f3*: ¬ $cdcl_W$*-cp* (*toS* (*rough-state-of S*)) (*toS* (*do-cp-step* (*rough-state-of S*)))
    ∨ *state-of* (*do-cp-step* (*rough-state-of S*)) = *S*
    ∨ $cdcl_W$*-cp*$^{++}$ (*toS* (*rough-state-of S*))
       (*toS* (*rough-state-of* (*do-full1-cp-step* (*state-of* (*do-cp-step* (*rough-state-of S*))))))
    using *f1* **by** (*meson rtranclp-into-tranclp2*)
  { assume *do-full1-cp-step S* ≠ *S*
    then have *do-cp-step* (*rough-state-of S*) = *rough-state-of S*
        ⟶ $cdcl_W$*-cp*** (*toS* (*rough-state-of S*)) (*toS* (*rough-state-of* (*do-full1-cp-step S*)))
      ∨ *do-cp-step* (*rough-state-of S*) ≠ *rough-state-of S*

$\wedge$ *state-of* (*do-cp-step* (*rough-state-of* S)) $\neq S$
using *f2 f1* **by** (*metis* (*no-types*))
then have *do-cp-step* (*rough-state-of* S) $\neq$ *rough-state-of* S
$\wedge$ *state-of* (*do-cp-step* (*rough-state-of* S)) $\neq S$
$\vee$ $cdcl_W$*-cp*** (*toS* (*rough-state-of* S)) (*toS* (*rough-state-of* (*do-full1-cp-step* S)))
by (*metis rough-state-of-state-of-do-cp-step*)
then have $cdcl_W$*-cp*** (*toS* (*rough-state-of* S)) (*toS* (*rough-state-of* (*do-full1-cp-step* S)))
using *f3 f2* **by** (*metis* (*no-types*) *cp-step-is-*$cdcl_W$*-cp tranclp-into-rtranclp*) **}**
then show *?case*
by *fastforce*
next
show *no-step* $cdcl_W$*-cp* (*toS* (*rough-state-of* (*do-full1-cp-step* S)))
apply (*rule do-cp-step-eq-no-step*[*OF do-full1-cp-step-fix-point-of-do-full1-cp-step*[*of S*]])
using *in-clauses-rough-state-of-is-distinct* **unfolding** *do-cp-step'-def* **by** *blast*
qed

lemma [*code abstract*]:
rough-state-of (*do-cp-step'* S) = *do-cp-step* (*rough-state-of* S)
unfolding *do-cp-step'-def* **by** *auto*

The other rules **fun** *do-other-step* **where**
do-other-step S =
(*let* T = *do-skip-step* S *in*
if $T \neq S$
then T
else
(*let* U = *do-resolve-step* T *in*
if $U \neq T$
then U *else*
(*let* V = *do-backtrack-step* U *in*
if $V \neq U$ *then* V *else do-decide-step* V)))

lemma *do-other-step*:
assumes *inv*: $cdcl_W$*-all-struct-inv* (*toS* S) **and**
st: *do-other-step* $S \neq S$
shows $cdcl_W$*-o* (*toS* S) (*toS* (*do-other-step* S))
using *st inv* **by** (*auto split*: *split-if-asm*
simp add: *Let-def*
intro: *do-skip-step do-resolve-step do-backtrack-step do-decide-step*)

lemma *do-other-step-no*:
assumes *inv*: $cdcl_W$*-all-struct-inv* (*toS* S) **and**
st: *do-other-step* $S = S$
shows *no-step* $cdcl_W$*-o* (*toS* S)
using *st inv* **by** (*auto split*: *split-if-asm elim*: $cdcl_W$*-bjE*
simp add: *Let-def* $cdcl_W$*-bj.simps elim!*: $cdcl_W$*-o.cases*
dest!: *do-skip-step-no do-resolve-step-no do-backtrack-step-no do-decide-step-no*)

lemma *rough-state-of-state-of-do-other-step*[*simp*]:
rough-state-of (*state-of* (*do-other-step* (*rough-state-of* S))) = *do-other-step* (*rough-state-of* S)
proof (*cases do-other-step* (*rough-state-of* S) = *rough-state-of* S)
case *True*
then show *?thesis* **by** *simp*
next
case *False*

have *cdcl*$_W$*-o* (*toS* (*rough-state-of S*)) (*toS* (*do-other-step* (*rough-state-of S*)))
by (*metis False cdcl*$_W$*-all-struct-inv-rough-state do-other-step*[*of rough-state-of S*])
then have *cdcl*$_W$*-all-struct-inv* (*toS* (*do-other-step* (*rough-state-of S*)))
using *cdcl*$_W$*-all-struct-inv-inv cdcl*$_W$*-all-struct-inv-rough-state other* **by** *blast*
then show *?thesis*
by (*simp add*: *CollectI state-of-inverse*)
qed

definition *do-other-step'* **where**
do-other-step' S =
state-of (*do-other-step* (*rough-state-of S*))

lemma *rough-state-of-do-other-step'*[*code abstract*]:
rough-state-of (*do-other-step' S*) = *do-other-step* (*rough-state-of S*)
apply (*cases do-other-step* (*rough-state-of S*) = *rough-state-of S*)
unfolding *do-other-step'-def* **apply** *simp*
using *do-other-step*[*of rough-state-of S*] **by** (*auto intro*: *cdcl*$_W$*-all-struct-inv-inv*
cdcl$_W$*-all-struct-inv-rough-state other state-of-inverse*)

definition *do-cdcl*$_W$*-stgy-step* **where**
do-cdcl$_W$*-stgy-step S* =
(*let T* = *do-full1-cp-step S in*
if T $\neq$ *S*
then T
else
(*let U* = (*do-other-step' T*) *in*
(*do-full1-cp-step U*)))

definition *do-cdcl*$_W$*-stgy-step'* **where**
do-cdcl$_W$*-stgy-step' S* = *state-from-init-state-of* (*rough-state-of* (*do-cdcl*$_W$*-stgy-step* (*id-of-I-to S*)))

lemma *toS-do-full1-cp-step-not-eq*: *do-full1-cp-step S* $\neq$ *S* $\Longrightarrow$
toS (*rough-state-of S*) $\neq$ *toS* (*rough-state-of* (*do-full1-cp-step S*))
proof −
assume *a1*: *do-full1-cp-step S* $\neq$ *S*
then have *S* $\neq$ *do-cp-step' S*
by *fastforce*
then show *?thesis*
by (*metis* (*no-types*) *cp-step-is-cdcl*$_W$*-cp do-cp-step'-def do-cp-step-eq-no-step*
do-full1-cp-step-fix-point-of-do-full1-cp-step in-clauses-rough-state-of-is-distinct
rough-state-of-inverse)
qed

do-full1-cp-step should not be unfolded anymore:

declare *do-full1-cp-step.simps*[*simp del*]

Correction of the transformation **lemma** *do-cdcl*$_W$*-stgy-step*:
assumes *do-cdcl*$_W$*-stgy-step S* $\neq$ *S*
shows *cdcl*$_W$*-stgy* (*toS* (*rough-state-of S*)) (*toS* (*rough-state-of* (*do-cdcl*$_W$*-stgy-step S*)))
proof (*cases do-full1-cp-step S* = *S*)
case *False*
then show *?thesis*
using *assms do-full1-cp-step-full*[*of S*] **unfolding** *full-unfold do-cdcl*$_W$*-stgy-step-def*
by (*auto intro*!: *cdcl*$_W$*-stgy.intros dest*: *toS-do-full1-cp-step-not-eq*)
next

case *True*
have $cdcl_W$*-o* (*toS* (*rough-state-of S*)) (*toS* (*rough-state-of* (*do-other-step′ S*)))
by (*smt True assms* $cdcl_W$*-all-struct-inv-rough-state do-*$cdcl_W$*-stgy-step-def do-other-step rough-state-of-do-other-step′ rough-state-of-inverse*)
moreover
have
np: *no-step propagate* (*toS* (*rough-state-of S*)) **and**
nc: *no-step conflict* (*toS* (*rough-state-of S*))
apply (*metis True do-cp-step-eq-no-prop-no-confl do-full1-cp-step-fix-point-of-do-full1-cp-step do-propagate-step-no-step in-clauses-rough-state-of-is-distinct*)
by (*metis True do-conflict-step-no-step do-cp-step-eq-no-prop-no-confl do-full1-cp-step-fix-point-of-do-full1-cp-step*)
then have *no-step* $cdcl_W$*-cp* (*toS* (*rough-state-of S*))
by (*simp add*: $cdcl_W$*-cp.simps*)
moreover have *full* $cdcl_W$*-cp* (*toS* (*rough-state-of* (*do-other-step′ S*)))
(*toS* (*rough-state-of* (*do-full1-cp-step* (*do-other-step′ S*))))
using *do-full1-cp-step-full* **by** *auto*
ultimately show *?thesis*
using *assms True* **unfolding** *do-*$cdcl_W$*-stgy-step-def*
by (*auto intro*!: $cdcl_W$*-stgy.other′ dest*: *toS-do-full1-cp-step-not-eq*)
qed

lemma *length-trail-toS*[*simp*]:
length (*trail* (*toS S*)) = *length* (*trail S*)
by (*cases S*) *auto*

lemma *conflicting-noTrue-iff-toS*[*simp*]:
conflicting (*toS S*) ≠ *None* ⟷ *conflicting S* ≠ *None*
by (*cases S*) *auto*

lemma *trail-toS-neq-imp-trail-neq*:
trail (*toS S*) ≠ *trail* (*toS S′*) ⟹ *trail S* ≠ *trail S′*
by (*cases S*, *cases S′*) *auto*

lemma *do-skip-step-trail-changed-or-conflict*:
assumes *d*: *do-other-step S* ≠ *S*
and *inv*: $cdcl_W$*-all-struct-inv* (*toS S*)
shows *trail S* ≠ *trail* (*do-other-step S*)
proof −
have *M*: ⋀*M K M1 c*. *M* = *c* @ *K* # *M1* ⟹ *Suc* (*length M1*) ≤ *length M*
by *auto*
have $cdcl_W$*-M-level-inv* (*toS S*)
using *inv* **unfolding** $cdcl_W$*-all-struct-inv-def* **by** *auto*
have $cdcl_W$*-o* (*toS S*) (*toS* (*do-other-step S*)) **using** *do-other-step*[*OF inv d*] **.**
then show *?thesis*
using ‹$cdcl_W$*-M-level-inv* (*toS S*)›
proof (*induction toS* (*do-other-step S*) *rule*: $cdcl_W$*-o-induct-lev2*)
case *decide*
then show *?thesis*
by (*auto simp add*: *trail-toS-neq-imp-trail-neq*)[]
next
case (*skip*)
then show *?case*
by (*cases S*; *cases do-other-step S*) *force*

next
case (*resolve*)
then show *?case*
by (*cases S*, *cases do-other-step S*) *force*
next
case (*backtrack K i M1 M2 L D*) **note** *decomp* = *this*(*1*) **and** *confl-S* = *this*(*3*) **and** *undef* = *this*(*6*)
and *U* = *this*(*7*)
have [*simp*]: *cons-trail* (*Propagated L* (*D* + {#*L*#}))
(*reduce-trail-to M1*
(*add-learned-cls* (*D* + {#*L*#})
(*update-backtrack-lvl* (*get-maximum-level* (*trail* (*toS S*)) *D*)
(*update-conflicting None* (*toS S*)))))
=
(*Propagated L* (*D* + {#*L*#})# *M1*,*mset* (*map mset* (*clss S*)),
{#*D* + {#*L*#}#} + *mset* (*map mset* (*learned-clss S*)),
get-maximum-level (*trail* (*toS S*)) *D*, *None*)
apply (*subst state-conv*[*of cons-trail* - -])
using *decomp undef* **by** (*cases S*) *auto*
then show *?case*
apply (*cases do-other-step S*)
apply (*auto split*: *split-if-asm simp*: *Let-def*)
apply (*cases S rule*: *do-skip-step.cases*; *auto split*: *split-if-asm*)
apply (*cases S rule*: *do-skip-step.cases*; *auto split*: *split-if-asm*)

apply (*cases S rule*: *do-backtrack-step.cases*;
auto split: *split-if-asm option.splits list.splits ann-literal.splits*
dest!: *bt-cut-some-decomp simp*: *Let-def*)
using *d* **apply** (*cases S rule*: *do-decide-step.cases*; *auto split*: *option.splits*)[]
done
qed
qed

lemma *do-full1-cp-step-induct*:
($\bigwedge$*S*. (*S* $\neq$ *do-cp-step' S* $\Longrightarrow$ *P* (*do-cp-step' S*)) $\Longrightarrow$ *P S*) $\Longrightarrow$ *P a0*
using *do-full1-cp-step.induct* **by** *metis*

lemma *do-cp-step-neq-trail-increase*:
$\exists$ *c*. *trail* (*do-cp-step S*) = *c* @ *trail S* $\wedge$($\forall$ *m* $\in$ *set c*. $\neg$ *is-marked m*)
by (*cases S*, *cases conflicting S*)
(*auto simp add*: *do-cp-step-def do-conflict-step-def do-propagate-step-def split*: *option.splits*)

lemma *do-full1-cp-step-neq-trail-increase*:
$\exists$ *c*. *trail* (*rough-state-of* (*do-full1-cp-step S*)) = *c* @ *trail* (*rough-state-of S*)
$\wedge$ ($\forall$ *m* $\in$ *set c*. $\neg$ *is-marked m*)
apply (*induction rule*: *do-full1-cp-step-induct*)
apply (*rename-tac S*, *case-tac do-cp-step' S* = *S*)
apply (*simp add*: *do-full1-cp-step.simps*)
by (*smt Un-iff append-assoc do-cp-step'-def do-cp-step-neq-trail-increase do-full1-cp-step.simps*
rough-state-of-state-of-do-cp-step set-append)

lemma *do-cp-step-conflicting*:
conflicting (*rough-state-of S*) $\neq$ *None* $\Longrightarrow$ *do-cp-step' S* = *S*
unfolding *do-cp-step'-def do-cp-step-def* **by** *simp*

```
lemma do-full1-cp-step-conflicting:
  conflicting (rough-state-of S) ≠ None ⟹ do-full1-cp-step S = S
  unfolding do-cp-step'-def do-cp-step-def
  apply (induction rule: do-full1-cp-step-induct)
  by (rename-tac S, case-tac S ≠ do-cp-step' S)
   (auto simp add: do-full1-cp-step.simps do-cp-step-conflicting)

lemma do-decide-step-not-conflicting-one-more-decide:
  assumes
    conflicting S = None and
    do-decide-step S ≠ S
  shows Suc (length (filter is-marked (trail S)))
    = length (filter is-marked (trail (do-decide-step S)))
  using assms unfolding do-other-step'-def
  by (cases S) (auto simp: Let-def split: split-if-asm option.splits
     dest!: find-first-unused-var-Some-not-all-incl)

lemma do-decide-step-not-conflicting-one-more-decide-bt:
  assumes conflicting S ≠ None and
  do-decide-step S ≠ S
  shows length (filter is-marked (trail S)) < length (filter is-marked (trail (do-decide-step S)))
  using assms unfolding do-other-step'-def by (cases S, cases conflicting S)
   (auto simp add: Let-def split: split-if-asm option.splits)

lemma do-other-step-not-conflicting-one-more-decide-bt:
  assumes
    conflicting (rough-state-of S) ≠ None and
    conflicting (rough-state-of (do-other-step' S)) = None and
    do-other-step' S ≠ S
  shows length (filter is-marked (trail (rough-state-of S)))
    > length (filter is-marked (trail (rough-state-of (do-other-step' S))))
proof (cases S, goal-cases)
  case (1 y) note S = this(1) and inv = this(2)
  obtain M N U k E where y: y = (M, N, U, k, Some E)
    using assms(1) S inv by (cases y, cases conflicting y) auto
  have M: rough-state-of (state-of (M, N, U, k,  Some E)) = (M, N, U, k,  Some E)
    using inv y by (auto simp add: state-of-inverse)
  have bt: do-other-step' S = state-of (do-backtrack-step (rough-state-of S))
    proof (cases rough-state-of S rule: do-decide-step.cases)
      case 1
      then show ?thesis
        using assms(1,2) by auto[]
    next
      case (2 v vb vd vf vh)
      have f3: ⋀c. (if do-skip-step (rough-state-of c) ≠ rough-state-of c
        then do-skip-step (rough-state-of c)
        else if do-resolve-step (do-skip-step (rough-state-of c)) ≠ do-skip-step (rough-state-of c)
             then do-resolve-step (do-skip-step (rough-state-of c))
             else if do-backtrack-step (do-resolve-step (do-skip-step (rough-state-of c)))
               ≠ do-resolve-step (do-skip-step (rough-state-of c))
             then do-backtrack-step (do-resolve-step (do-skip-step (rough-state-of c)))
             else do-decide-step (do-backtrack-step (do-resolve-step
               (do-skip-step (rough-state-of c)))))
        = rough-state-of (do-other-step' c)
        by (simp add: rough-state-of-do-other-step')
```

have (*trail* (*rough-state-of* (*do-other-step′ S*)), *clss* (*rough-state-of* (*do-other-step′ S*)),
learned-clss (*rough-state-of* (*do-other-step′ S*)),
backtrack-lvl (*rough-state-of* (*do-other-step′ S*)), *None*)
= *rough-state-of* (*do-other-step′ S*)
using *assms*(*2*) **by** (*metis* (*no-types*) *state-conv*)
then show *?thesis*
using *f3 2* **by** (*metis* (*no-types*) *do-decide-step.simps*(*2*) *do-resolve-step-trail-is-None*
do-skip-step-trail-is-None rough-state-of-inverse)
qed
show *?case*
using *assms*(*2*) *S* **unfolding** *bt y inv*
apply *simp*
by (*auto simp add*: *M bt-cut-not-none*
split: *option.splits*
dest!: *bt-cut-some-decomp*)
qed

lemma *do-other-step-not-conflicting-one-more-decide*:
assumes *conflicting* (*rough-state-of S*) = *None* **and**
do-other-step′ S ≠ *S*
shows *1* + *length* (*filter is-marked* (*trail* (*rough-state-of S*)))
= *length* (*filter is-marked* (*trail* (*rough-state-of* (*do-other-step′ S*))))
proof (*cases S*, *goal-cases*)
case (*1 y*) **note** *S* = *this*(*1*) **and** *inv* = *this*(*2*)
obtain *M N U k* **where** *y*: *y* = (*M*, *N*, *U*, *k*, *None*) **using** *assms*(*1*) *S inv* **by** (*cases y*) *auto*
have *M*: *rough-state-of* (*state-of* (*M*, *N*, *U*, *k*, *None*)) = (*M*, *N*, *U*, *k*, *None*)
using *inv y* **by** (*auto simp add*: *state-of-inverse*)
have *state-of* (*do-decide-step* (*M*, *N*, *U*, *k*, *None*)) ≠ *state-of* (*M*, *N*, *U*, *k*, *None*)
using *assms*(*2*) **unfolding** *do-other-step′-def y inv S* **by** (*auto simp add*: *M*)
then have *f4*: *do-skip-step* (*rough-state-of S*) = *rough-state-of S*
unfolding *S M y* **by** (*metis* (*full-types*) *do-skip-step.simps*(*4*))
have *f5*: *do-resolve-step* (*rough-state-of S*) = *rough-state-of S*
unfolding *S M y* **by** (*metis* (*no-types*) *do-resolve-step.simps*(*4*))
have *f6*: *do-backtrack-step* (*rough-state-of S*) = *rough-state-of S*
unfolding *S M y* **by** (*metis* (*no-types*) *do-backtrack-step.simps*(*2*))
have *do-other-step* (*rough-state-of S*) ≠ *rough-state-of S*
using *assms*(*2*) **unfolding** *S M y do-other-step′-def* **by** (*metis* (*no-types*))
then show *?case*
using *f6 f5 f4* **by** (*simp add*: *assms*(*1*) *do-decide-step-not-conflicting-one-more-decide*
do-other-step′-def)
qed

lemma *rough-state-of-state-of-do-skip-step-rough-state-of*[*simp*]:
rough-state-of (*state-of* (*do-skip-step* (*rough-state-of S*))) = *do-skip-step* (*rough-state-of S*)
by (*smt do-other-step.simps rough-state-of-inverse rough-state-of-state-of-do-other-step*)

lemma *conflicting-do-resolve-step-iff*[*iff*]:
conflicting (*do-resolve-step S*) = *None* ⟷ *conflicting S* = *None*
by (*cases S rule*: *do-resolve-step.cases*)
(*auto simp add*: *Let-def split*: *option.splits*)

lemma *conflicting-do-skip-step-iff*[*iff*]:
conflicting (*do-skip-step S*) = *None* ⟷ *conflicting S* = *None*
by (*cases S rule*: *do-skip-step.cases*)
(*auto simp add*: *Let-def split*: *option.splits*)

lemma *conflicting-do-decide-step-iff*[*iff*]:
conflicting (*do-decide-step* S) = *None* $\longleftrightarrow$ *conflicting* S = *None*
by (*cases* S *rule*: *do-decide-step.cases*)
(*auto simp add*: *Let-def split*: *option.splits*)

lemma *conflicting-do-backtrack-step-imp*[*simp*]:
do-backtrack-step $S \neq S \Longrightarrow$ *conflicting* (*do-backtrack-step* S) = *None*
by (*cases* S *rule*: *do-backtrack-step.cases*)
(*auto simp add*: *Let-def split*: *list.splits option.splits ann-literal.splits*)

lemma *do-skip-step-eq-iff-trail-eq*:
do-skip-step $S = S \longleftrightarrow$ *trail* (*do-skip-step* S) = *trail* S
by (*cases* S *rule*: *do-skip-step.cases*) *auto*

lemma *do-decide-step-eq-iff-trail-eq*:
do-decide-step $S = S \longleftrightarrow$ *trail* (*do-decide-step* S) = *trail* S
by (*cases* S *rule*: *do-decide-step.cases*) (*auto split*: *option.split*)

lemma *do-backtrack-step-eq-iff-trail-eq*:
do-backtrack-step $S = S \longleftrightarrow$ *trail* (*do-backtrack-step* S) = *trail* S
by (*cases* S *rule*: *do-backtrack-step.cases*)
(*auto split*: *option.split list.splits ann-literal.splits*
dest!: *bt-cut-in-get-all-marked-decomposition*)

lemma *do-resolve-step-eq-iff-trail-eq*:
do-resolve-step $S = S \longleftrightarrow$ *trail* (*do-resolve-step* S) = *trail* S
by (*cases* S *rule*: *do-resolve-step.cases*) *auto*

lemma *do-other-step-eq-iff-trail-eq*:
trail (*do-other-step* S) = *trail* $S \longleftrightarrow$ *do-other-step* $S = S$
by (*auto simp add*: *Let-def do-skip-step-eq-iff-trail-eq*[*symmetric*]
do-decide-step-eq-iff-trail-eq[*symmetric*] *do-backtrack-step-eq-iff-trail-eq*[*symmetric*]
do-resolve-step-eq-iff-trail-eq[*symmetric*])

lemma *do-full1-cp-step-do-other-step′-normal-form*[*dest*!]:
assumes H: *do-full1-cp-step* (*do-other-step′* S) = S
shows *do-other-step′* $S = S \wedge$ *do-full1-cp-step* $S = S$
proof −
let *?T* = *do-other-step′* S
{ assume *confl*: *conflicting* (*rough-state-of ?T*) $\neq$ *None*
then have *tr*: *trail* (*rough-state-of* (*do-full1-cp-step ?T*)) = *trail* (*rough-state-of ?T*)
using *do-full1-cp-step-conflicting* **by** *auto*
have *trail* (*rough-state-of* (*do-full1-cp-step* (*do-other-step′* S))) = *trail* (*rough-state-of* S)
using *arg-cong*[*OF* H, *of* λS. *trail* (*rough-state-of* S)] **.**
then have *trail* (*rough-state-of* (*do-other-step′* S)) = *trail* (*rough-state-of* S)
by (*auto simp add*: *do-full1-cp-step-conflicting confl*)
then have *do-other-step′* $S = S$
by (*simp add*: *do-other-step-eq-iff-trail-eq do-other-step′-def*
del: *do-other-step.simps*)
}
moreover {
assume *eq*[*simp*]: *do-other-step′* $S = S$
obtain c **where** c: *trail* (*rough-state-of* (*do-full1-cp-step* S)) = c @ *trail* (*rough-state-of* S)

using *do-full1-cp-step-neq-trail-increase* **by** *auto*

moreover have *trail* (*rough-state-of* (*do-full1-cp-step S*)) = *trail* (*rough-state-of S*)
using *arg-cong*[*OF H*, *of* λ*S*. *trail* (*rough-state-of S*)] **by** *simp*
finally have *c* = [] **by** *blast*
then have *do-full1-cp-step S* = *S* **using** *assms* **by** *auto*
}
moreover {
assume *confl*: *conflicting* (*rough-state-of ?T*) = *None* **and** *neq*: *do-other-step′ S* ≠ *S*
obtain *c* **where**
c: *trail* (*rough-state-of* (*do-full1-cp-step ?T*)) = *c* @ *trail* (*rough-state-of ?T*) **and**
nm: ∀ *m*∈*set c*. ¬ *is-marked m*
using *do-full1-cp-step-neq-trail-increase* **by** *auto*
have *length* (*filter is-marked* (*trail* (*rough-state-of* (*do-full1-cp-step ?T*))))
= *length* (*filter is-marked* (*trail* (*rough-state-of ?T*))) **using** *nm* **unfolding** *c* **by** *force*
moreover have *length* (*filter is-marked* (*trail* (*rough-state-of S*)))
≠ *length* (*filter is-marked* (*trail* (*rough-state-of ?T*)))
using *do-other-step-not-conflicting-one-more-decide*[*OF* - *neq*]
do-other-step-not-conflicting-one-more-decide-bt[*of S*, *OF* - *confl neq*]
by *linarith*
finally have *False* **unfolding** *H* **by** *blast*
}
ultimately show *?thesis* **by** *blast*
qed

lemma *do-cdcl*$_W$*-stgy-step-no*:
assumes *S*: *do-cdcl*$_W$*-stgy-step S* = *S*
shows *no-step cdcl*$_W$*-stgy* (*toS* (*rough-state-of S*))
proof −
{
fix *S′*
assume *full1 cdcl*$_W$*-cp* (*toS* (*rough-state-of S*)) *S′*
then have *False*
using *do-full1-cp-step-full*[*of S*] **unfolding** *full-def S rtranclp-unfold full1-def*
by (*smt assms do-cdcl*$_W$*-stgy-step-def tranclpD*)
}
moreover {
fix *S′ S″*
assume *cdcl*$_W$*-o* (*toS* (*rough-state-of S*)) *S′* **and**
no-step propagate (*toS* (*rough-state-of S*)) **and**
no-step conflict (*toS* (*rough-state-of S*)) **and**
full cdcl$_W$*-cp S′ S″*
then have *False*
using *assms* **unfolding** *do-cdcl*$_W$*-stgy-step-def*
by (*smt cdcl*$_W$*-all-struct-inv-rough-state do-full1-cp-step-do-other-step′-normal-form*
do-other-step-no rough-state-of-do-other-step′)
}
ultimately show *?thesis* **using** *assms* **by** (*force simp*: *cdcl*$_W$*-cp.simps cdcl*$_W$*-stgy.simps*)
qed

lemma *toS-rough-state-of-state-of-rough-state-from-init-state-of*[*simp*]:
toS (*rough-state-of* (*state-of* (*rough-state-from-init-state-of S*)))
= *toS* (*rough-state-from-init-state-of S*)
using *rough-state-from-init-state-of*[*of S*] **by** (*auto simp add*: *state-of-inverse*)

lemma *cdcl$_W$-cp-is-rtranclp-cdcl$_W$*: *cdcl$_W$-cp S T* $\Longrightarrow$ *cdcl$_W$*** *S T*
apply (*induction rule*: *cdcl$_W$-cp.induct*)
using *conflict* **apply** *blast*
using *propagate* **by** *blast*

lemma *rtranclp-cdcl$_W$-cp-is-rtranclp-cdcl$_W$*: *cdcl$_W$-cp*** *S T* $\Longrightarrow$ *cdcl$_W$*** *S T*
apply (*induction rule*: *rtranclp-induct*)
apply *simp*
by (*fastforce dest*!: *cdcl$_W$-cp-is-rtranclp-cdcl$_W$*)

lemma *cdcl$_W$-stgy-is-rtranclp-cdcl$_W$*:
cdcl$_W$-stgy S T $\Longrightarrow$ *cdcl$_W$*** *S T*
apply (*induction rule*: *cdcl$_W$-stgy.induct*)
using *cdcl$_W$-stgy.conflict*$'$ *rtranclp-cdcl$_W$-stgy-rtranclp-cdcl$_W$* **apply** *blast*
unfolding *full-def* **by** (*fastforce dest*!:*other rtranclp-cdcl$_W$-cp-is-rtranclp-cdcl$_W$*)

lemma *cdcl$_W$-stgy-init-clss*: *cdcl$_W$-stgy S T* $\Longrightarrow$ *cdcl$_W$-M-level-inv S* $\Longrightarrow$ *clss S* = *clss T*
using *rtranclp-cdcl$_W$-init-clss cdcl$_W$-stgy-is-rtranclp-cdcl$_W$* **by** *fast*

lemma *clauses-toS-rough-state-of-do-cdcl$_W$-stgy-step*[*simp*]:
clss (*toS* (*rough-state-of* (*do-cdcl$_W$-stgy-step* (*state-of* (*rough-state-from-init-state-of S*)))))
= *clss* (*toS* (*rough-state-from-init-state-of S*)) (**is** - = *clss* (*toS ?S*))
apply (*cases do-cdcl$_W$-stgy-step* (*state-of ?S*) = *state-of ?S*)
apply *simp*
by (*smt cdcl$_W$-all-struct-inv-def cdcl$_W$-all-struct-inv-rough-state cdcl$_W$-stgy-no-more-init-clss*
do-cdcl$_W$-stgy-step toS-rough-state-of-state-of-rough-state-from-init-state-of)

lemma *rough-state-from-init-state-of-do-cdcl$_W$-stgy-step*$'$[*code abstract*]:
rough-state-from-init-state-of (*do-cdcl$_W$-stgy-step*$'$ *S*) =
rough-state-of (*do-cdcl$_W$-stgy-step* (*id-of-I-to S*))
proof −
let *?S* = (*rough-state-from-init-state-of S*)
have *cdcl$_W$-stgy*** (*S0-cdcl$_W$* (*clss* (*toS* (*rough-state-from-init-state-of S*))))
(*toS* (*rough-state-from-init-state-of S*))
using *rough-state-from-init-state-of*[*of S*] **by** *auto*
moreover have *cdcl$_W$-stgy***
(*toS* (*rough-state-from-init-state-of S*))
(*toS* (*rough-state-of* (*do-cdcl$_W$-stgy-step*
(*state-of* (*rough-state-from-init-state-of S*)))))
using *do-cdcl$_W$-stgy-step*[*of state-of ?S*]
by (*cases do-cdcl$_W$-stgy-step* (*state-of ?S*) = *state-of ?S*) *auto*
ultimately show *?thesis*
unfolding *do-cdcl$_W$-stgy-step*$'$*-def id-of-I-to-def*
by (*auto intro*!: *state-from-init-state-of-inverse*)
qed

All rules together **function** *do-all-cdcl$_W$-stgy* **where**
do-all-cdcl$_W$-stgy S =
(*let T* = *do-cdcl$_W$-stgy-step*$'$ *S in*
if T = *S then S else do-all-cdcl$_W$-stgy T*)
by *fast+*
termination
proof (*relation* {(*T*, *S*).
(*cdcl$_W$-measure* (*toS* (*rough-state-from-init-state-of T*)),
cdcl$_W$-measure (*toS* (*rough-state-from-init-state-of S*)))

```
    ∈ lexn {(a, b). a < b} 3}, goal-cases)
  case 1
  show ?case by (rule wf-if-measure-f) (auto intro!: wf-lexn wf-less)
next
  case (2 S T) note T = this(1) and ST = this(2)
  let ?S = rough-state-from-init-state-of S
  have S: cdcl_W-stgy** (S0-cdcl_W (clss (toS ?S))) (toS ?S)
    using rough-state-from-init-state-of[of S] by auto
  moreover have cdcl_W-stgy (toS (rough-state-from-init-state-of S))
    (toS (rough-state-from-init-state-of T))
    proof –
      have ⋀c. rough-state-of (state-of (rough-state-from-init-state-of c)) =
        rough-state-from-init-state-of c
        using rough-state-from-init-state-of by force
      then have do-cdcl_W-stgy-step (state-of (rough-state-from-init-state-of S))
        ≠ state-of (rough-state-from-init-state-of S)
        using ST T by (metis (no-types) id-of-I-to-def rough-state-from-init-state-of-inject
          rough-state-from-init-state-of-do-cdcl_W-stgy-step')
      then show ?thesis
        using do-cdcl_W-stgy-step id-of-I-to-def rough-state-from-init-state-of-do-cdcl_W-stgy-step' T
        by fastforce
    qed
  moreover
    have cdcl_W-all-struct-inv (toS (rough-state-from-init-state-of S))
      using rough-state-from-init-state-of[of S] by auto
    then have cdcl_W-all-struct-inv (S0-cdcl_W (clss (toS (rough-state-from-init-state-of S))))
      by (cases rough-state-from-init-state-of S)
         (auto simp add: cdcl_W-all-struct-inv-def distinct-cdcl_W-state-def)
  ultimately show ?case
    by (auto intro!: cdcl_W-stgy-step-decreasing[of - - S0-cdcl_W (clss (toS ?S))]
      simp del: cdcl_W-measure.simps)
qed

thm do-all-cdcl_W-stgy.induct
lemma do-all-cdcl_W-stgy-induct:
  (⋀S. (do-cdcl_W-stgy-step' S ≠ S ⟹ P (do-cdcl_W-stgy-step' S)) ⟹ P S) ⟹ P a0
 using do-all-cdcl_W-stgy.induct by metis

lemma no-step-cdcl_W-stgy-cdcl_W-all:
  no-step cdcl_W-stgy (toS (rough-state-from-init-state-of (do-all-cdcl_W-stgy S)))
  apply (induction S rule:do-all-cdcl_W-stgy-induct)
  apply (rename-tac S, case-tac do-cdcl_W-stgy-step' S ≠ S)
proof –
  fix Sa :: cdcl_W-state-inv-from-init-state
  assume a1: ¬ do-cdcl_W-stgy-step' Sa ≠ Sa
  { fix pp
    have (if True then Sa else do-all-cdcl_W-stgy Sa) = do-all-cdcl_W-stgy Sa
      using a1 by auto
    then have ¬ cdcl_W-stgy (toS (rough-state-from-init-state-of (do-all-cdcl_W-stgy Sa))) pp
      using a1 by (metis (no-types) do-cdcl_W-stgy-step-no id-of-I-to-def
        rough-state-from-init-state-of-do-cdcl_W-stgy-step' rough-state-of-inverse) }
  then show no-step cdcl_W-stgy (toS (rough-state-from-init-state-of (do-all-cdcl_W-stgy Sa)))
    by fastforce
next
  fix Sa :: cdcl_W-state-inv-from-init-state
```

assume *a1*: *do-cdcl*$_W$*-stgy-step′ Sa* $\neq$ *Sa*
$\Longrightarrow$ *no-step cdcl*$_W$*-stgy* (*toS* (*rough-state-from-init-state-of*
(*do-all-cdcl*$_W$*-stgy* (*do-cdcl*$_W$*-stgy-step′ Sa*))))
assume *a2*: *do-cdcl*$_W$*-stgy-step′ Sa* $\neq$ *Sa*
have *do-all-cdcl*$_W$*-stgy Sa* = *do-all-cdcl*$_W$*-stgy* (*do-cdcl*$_W$*-stgy-step′ Sa*)
by (*metis* (*full-types*) *do-all-cdcl*$_W$*-stgy.simps*)
then show *no-step cdcl*$_W$*-stgy* (*toS* (*rough-state-from-init-state-of* (*do-all-cdcl*$_W$*-stgy Sa*)))
using *a2 a1* **by** *presburger*
qed

lemma *do-all-cdcl*$_W$*-stgy-is-rtranclp-cdcl*$_W$*-stgy*:
cdcl$_W$*-stgy*** (*toS* (*rough-state-from-init-state-of S*))
(*toS* (*rough-state-from-init-state-of* (*do-all-cdcl*$_W$*-stgy S*)))
proof (*induction S rule*: *do-all-cdcl*$_W$*-stgy-induct*)
case (*1 S*) **note** *IH* = *this*(*1*)
show *?case*
proof (*cases do-cdcl*$_W$*-stgy-step′ S* = *S*)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
have *f2*: *do-cdcl*$_W$*-stgy-step* (*id-of-I-to S*) = *id-of-I-to S* $\longrightarrow$
rough-state-from-init-state-of (*do-cdcl*$_W$*-stgy-step′ S*)
= *rough-state-of* (*state-of* (*rough-state-from-init-state-of S*))
using *id-of-I-to-def rough-state-from-init-state-of-do-cdcl*$_W$*-stgy-step′* **by** *presburger*
have *f3*: *do-all-cdcl*$_W$*-stgy S* = *do-all-cdcl*$_W$*-stgy* (*do-cdcl*$_W$*-stgy-step′ S*)
by (*metis* (*full-types*) *do-all-cdcl*$_W$*-stgy.simps*)
have *cdcl*$_W$*-stgy* (*toS* (*rough-state-from-init-state-of S*))
(*toS* (*rough-state-from-init-state-of* (*do-cdcl*$_W$*-stgy-step′ S*)))
= *cdcl*$_W$*-stgy* (*toS* (*rough-state-of* (*id-of-I-to S*)))
(*toS* (*rough-state-of* (*do-cdcl*$_W$*-stgy-step* (*id-of-I-to S*))))
using *id-of-I-to-def rough-state-from-init-state-of-do-cdcl*$_W$*-stgy-step′*
toS-rough-state-of-state-of-rough-state-from-init-state-of **by** *presburger*
then show *?thesis*
using *f3 f2 IH do-cdcl*$_W$*-stgy-step* **by** *fastforce*
qed
qed

Final theorem:

lemma *DPLL-tot-correct*:
assumes
r: *rough-state-from-init-state-of* (*do-all-cdcl*$_W$*-stgy* (*state-from-init-state-of*
(([], *map remdups N*, [], *0*, *None*)))) = *S* **and**
S: (*M′*, *N′*, *U′*, *k*, *E*) = *toS S*
shows (*E* $\neq$ *Some* {#} $\wedge$ *satisfiable* (*set* (*map mset N*)))
$\vee$ (*E* = *Some* {#} $\wedge$ *unsatisfiable* (*set* (*map mset N*)))
proof −
let *?N* = *map remdups N*
have *inv*: *cdcl*$_W$*-all-struct-inv* (*toS* ([], *map remdups N*, [], *0*, *None*))
unfolding *cdcl*$_W$*-all-struct-inv-def distinct-cdcl*$_W$*-state-def distinct-mset-set-def* **by** *auto*
then have *S0*: *rough-state-of* (*state-of* ([], *map remdups N*, [], *0*, *None*))
= ([], *map remdups N*, [], *0*, *None*) **by** *simp*
have *1*: *full cdcl*$_W$*-stgy* (*toS* ([], *?N*, [], *0*, *None*)) (*toS S*)
unfolding *full-def* **apply** *rule*
using *do-all-cdcl*$_W$*-stgy-is-rtranclp-cdcl*$_W$*-stgy*[*of*

state-from-init-state-of ([], *map remdups N*, [], *0*, *None*)] *inv*
no-step-cdcl$_W$-stgy-cdcl$_W$-all
by (*auto simp del*: *do-all-cdcl$_W$-stgy.simps simp*: *state-from-init-state-of-inverse*
r[*symmetric*])+
moreover have *2*: *finite* (*set* (*map mset ?N*)) **by** *auto*
moreover have *3*: *distinct-mset-set* (*set* (*map mset ?N*))
unfolding *distinct-mset-set-def* **by** *auto*
moreover
have *cdcl$_W$-all-struct-inv* (*toS S*)
by (*metis* (*no-types*) *cdcl$_W$-all-struct-inv-rough-state r*
toS-rough-state-of-state-of-rough-state-from-init-state-of)
then have *cons*: *consistent-interp* (*lits-of M'*)
unfolding *cdcl$_W$-all-struct-inv-def cdcl$_W$-M-level-inv-def S*[*symmetric*] **by** *auto*
moreover
have *clss* (*toS* ([], *?N*, [], *0*, *None*)) = *clss* (*toS S*)
apply (*rule rtranclp-cdcl$_W$-init-clss*)
using *1* **unfolding** *full-def* **by** (*auto simp add*: *rtranclp-cdcl$_W$-stgy-rtranclp-cdcl$_W$*)
then have *N'*: *mset* (*map mset ?N*) = *N'*
using *S*[*symmetric*] **by** *auto*
have ($E \neq$ *Some* {#} $\wedge$ *satisfiable* (*set* (*map mset ?N*)))
$\vee$ ($E =$ *Some* {#} $\wedge$ *unsatisfiable* (*set* (*map mset ?N*)))
using *full-cdcl$_W$-stgy-final-state-conclusive* **unfolding** *N'* **apply** *rule*
using *1* **apply** *simp*
using *2* **apply** *simp*
using *3* **apply** *simp*
using *S*[*symmetric*] *N'* **apply** *auto*[*1*]
using *S*[*symmetric*] *N' cons* **by** (*fastforce simp*: *true-annots-true-cls*)
then show *?thesis* **by** *auto*
qed

The Code The SML code is skipped in the documentation, but stays to ensure that some version of the exported code is working. The only difference between the generated code and the one used here is the export of the constructor ConI.

end
theory *CDCL-WNOT*
imports *CDCL-W-Termination CDCL-NOT*
begin

7 Link between Weidenbach's and NOT's CDCL

7.1 Inclusion of the states

declare *upt.simps*(*2*)[*simp del*]
sledgehammer-params[*verbose*]

context *cdcl$_W$*
begin

lemma *backtrack-levE*:
backtrack S S' $\Longrightarrow$ *cdcl$_W$-M-level-inv S* $\Longrightarrow$
($\bigwedge$*D L K M1 M2*.
(*Marked K* (*Suc* (*get-maximum-level* (*trail S*) *D*)) # *M1*, *M2*)
$\in$ *set* (*get-all-marked-decomposition* (*trail S*)) $\Longrightarrow$
get-level (*trail S*) *L* = *get-maximum-level* (*trail S*) (*D* + {#*L*#}) $\Longrightarrow$

undefined-lit M1 L $\Longrightarrow$
S' $\sim$ *cons-trail* (*Propagated L* (*D* + {#*L*#}))
(*reduce-trail-to M1* (*add-learned-cls* (*D* + {#*L*#})
(*update-backtrack-lvl* (*get-maximum-level* (*trail S*) *D*) (*update-conflicting None S*)))) $\Longrightarrow$
backtrack-lvl S = *get-maximum-level* (*trail S*) (*D* + {#*L*#}) $\Longrightarrow$
conflicting S = *Some* (*D* + {#*L*#}) $\Longrightarrow$ *P*) $\Longrightarrow$
P
using *assms* **by** (*induction rule*: *backtrack-induction-lev2*) *metis*

lemma *backtrack-no-cdcl*$_W$*-bj*:
assumes *cdcl*: *cdcl*$_W$*-bj T U* **and** *inv*: *cdcl*$_W$*-M-level-inv V*
shows $\neg$*backtrack V T*
using *cdcl inv*
apply (*induction rule*: *cdcl*$_W$*-bj.induct*)
apply (*elim skipE*, *force elim*!: *backtrack-levE*[*OF - inv*] *simp*: *cdcl*$_W$*-M-level-inv-def*)
apply (*elim resolveE*, *force elim*!: *backtrack-levE*[*OF - inv*] *simp*: *cdcl*$_W$*-M-level-inv-def*)
apply *standard*
apply (*elim backtrack-levE*[*OF - inv*], *elim backtrackE*)
apply (*force simp del*: *state-simp simp add*: *state-eq-conflicting cdcl*$_W$*-M-level-inv-decomp*)
done

abbreviation *skip-or-resolve* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **where**
skip-or-resolve $\equiv$ (λ*S T*. *skip S T* $\vee$ *resolve S T*)

lemma *rtranclp-cdcl*$_W$*-bj-skip-or-resolve-backtrack*:
assumes *cdcl*$_W$*-bj*** *S U* **and** *inv*: *cdcl*$_W$*-M-level-inv S*
shows *skip-or-resolve*** *S U* $\vee$ ($\exists$ *T*. *skip-or-resolve*** *S T* $\wedge$ *backtrack T U*)
using *assms*
proof (*induction*)
case *base*
then show *?case* **by** *simp*
next
case (*step U V*) **note** *st* = *this*(*1*) **and** *bj* = *this*(*2*) **and** *IH* = *this*(*3*)[*OF this*(*4*)]
consider
(*SU*) *S* = *U*
| (*SUp*) *cdcl*$_W$*-bj*$^{++}$ *S U*
using *st* **unfolding** *rtranclp-unfold* **by** *blast*
then show *?case*
proof *cases*
case *SUp*
have $\bigwedge$*T*. *skip-or-resolve*** *S T* $\Longrightarrow$ *cdcl*$_W$** *S T*
using *mono-rtranclp*[*of skip-or-resolve cdcl*$_W$] *other* **by** *blast*
then have *skip-or-resolve*** *S U*
using *bj IH inv backtrack-no-cdcl*$_W$*-bj rtranclp-cdcl*$_W$*-consistent-inv*[*OF - inv*] **by** *meson*
then show *?thesis*
using *bj* **by** (*metis* (*no-types*, *lifting*) *cdcl*$_W$*-bj.cases rtranclp.simps*)
next
case *SU*
then show *?thesis*
using *bj* **by** (*metis* (*no-types*, *lifting*) *cdcl*$_W$*-bj.cases rtranclp.simps*)
qed
qed

lemma *rtranclp-skip-or-resolve-rtranclp-cdcl*$_W$:

skip-or-resolve** *S T* $\Longrightarrow$ *cdcl*$_W$** *S T*
by (*induction rule*: *rtranclp-induct*) (*auto dest*!: *cdcl*$_W$*-bj.intros cdcl*$_W$*.intros cdcl*$_W$*-o.intros*)

definition *backjump-l-cond* :: *'v clause* $\Rightarrow$ *'v clause* $\Rightarrow$ *'v literal* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **where**
backjump-l-cond $\equiv$ λ*C C' L' S. True*

definition *inv*$_{NOT}$ :: *'st* $\Rightarrow$ *bool* **where**
inv$_{NOT}$ $\equiv$ λ*S. no-dup* (*trail S*)

declare *inv*$_{NOT}$*-def*[*simp*]
end

fun *convert-ann-literal-from-W* **where**
convert-ann-literal-from-W (*Propagated L -*) = *Propagated L* () |
convert-ann-literal-from-W (*Marked L -*) = *Marked L* ()

abbreviation *convert-trail-from-W* ::
('*v*, '*lvl*, '*a*) *ann-literal list*
$\Rightarrow$ ('*v*, *unit*, *unit*) *ann-literal list* **where**
convert-trail-from-W $\equiv$ *map convert-ann-literal-from-W*

lemma *lits-of-convert-trail-from-W*[*simp*]:
lits-of (*convert-trail-from-W M*) = *lits-of M*
by (*induction rule*: *ann-literal-list-induct*) *simp-all*

lemma *lit-of-convert-trail-from-W*[*simp*]:
lit-of (*convert-ann-literal-from-W L*) = *lit-of L*
by (*cases L*) *auto*

lemma *no-dup-convert-from-W*[*simp*]:
no-dup (*convert-trail-from-W M*) $\longleftrightarrow$ *no-dup M*
by (*auto simp*: *comp-def*)

lemma *convert-trail-from-W-true-annots*[*simp*]:
convert-trail-from-W M $\models$*as C* $\longleftrightarrow$ *M* $\models$*as C*
by (*auto simp*: *true-annots-true-cls*)

lemma *defined-lit-convert-trail-from-W*[*simp*]:
defined-lit (*convert-trail-from-W S*) *L* $\longleftrightarrow$ *defined-lit S L*
by (*auto simp*: *defined-lit-map image-comp*)

The values *0* and {#} are dummy values.

fun *convert-ann-literal-from-NOT*
:: ('*a*, '*e*, '*b*) *ann-literal* $\Rightarrow$ ('*a*, *nat*, '*a literal multiset*) *ann-literal* **where**
convert-ann-literal-from-NOT (*Propagated L -*) = *Propagated L* {#} |
convert-ann-literal-from-NOT (*Marked L -*) = *Marked L 0*

abbreviation *convert-trail-from-NOT* **where**
convert-trail-from-NOT $\equiv$ *map convert-ann-literal-from-NOT*

lemma *undefined-lit-convert-trail-from-NOT*[*simp*]:
undefined-lit (*convert-trail-from-NOT F*) *L* $\longleftrightarrow$ *undefined-lit F L*
by (*induction F rule*: *ann-literal-list-induct*) (*auto simp*: *defined-lit-map*)

lemma *lits-of-convert-trail-from-NOT*:

lits-of (*convert-trail-from-NOT F*) = *lits-of F*
by (*induction F rule*: *ann-literal-list-induct*) *auto*

lemma *convert-trail-from-W-from-NOT*[*simp*]:
convert-trail-from-W (*convert-trail-from-NOT M*) = *M*
by (*induction rule*: *ann-literal-list-induct*) *auto*

lemma *convert-trail-from-W-convert-lit-from-NOT*[*simp*]:
convert-ann-literal-from-W (*convert-ann-literal-from-NOT L*) = *L*
by (*cases L*) *auto*

abbreviation $trail_{NOT}$ **where**
$trail_{NOT}$ *S* ≡ *convert-trail-from-W* (*fst S*)

lemma *undefined-lit-convert-trail-from-W*[*iff*]:
undefined-lit (*convert-trail-from-W M*) *L* ⟷ *undefined-lit M L*
by (*auto simp*: *defined-lit-map image-comp*)

lemma *lit-of-convert-ann-literal-from-NOT*[*iff*]:
lit-of (*convert-ann-literal-from-NOT L*) = *lit-of L*
by (*cases L*) *auto*

sublocale $state_W$ ⊆ *dpll-state*
λ*S*. *convert-trail-from-W* (*trail S*)
clauses
λ*L S*. *cons-trail* (*convert-ann-literal-from-NOT L*) *S*
λ*S*. *tl-trail S*
λ*C S*. *add-learned-cls C S*
λ*C S*. *remove-cls C S*
by *unfold-locales* (*auto simp*: *map-tl o-def*)

context $state_W$
begin
declare $state\text{-}simp_{NOT}$[*simp del*]
end

sublocale $cdcl_W$ ⊆ $cdcl_{NOT}$*-merge-bj-learn-ops*
λ*S*. *convert-trail-from-W* (*trail S*)
clauses
λ*L S*. *cons-trail* (*convert-ann-literal-from-NOT L*) *S*
λ*S*. *tl-trail S*
λ*C S*. *add-learned-cls C S*
λ*C S*. *remove-cls C S*
λ- -. *True*
λ- *S*. *conflicting S* = *None*
λ*C C' L' S*. *backjump-l-cond C C' L' S* ∧ *distinct-mset* (*C'* + {#*L'*#}) ∧ ¬*tautology* (*C'* + {#*L'*#})
by *unfold-locales*

sublocale $cdcl_W$ ⊆ $cdcl_{NOT}$*-merge-bj-learn-proxy*
λ*S*. *convert-trail-from-W* (*trail S*)
clauses
λ*L S*. *cons-trail* (*convert-ann-literal-from-NOT L*) *S*
λ*S*. *tl-trail S*
λ*C S*. *add-learned-cls C S*
λ*C S*. *remove-cls C S*

λ- -. True
λ- S. conflicting S = None backjump-l-cond inv_{NOT}
proof (*unfold-locales*, *goal-cases*)
case *2*
then show *?case* **using** $cdcl_{NOT}$*-merged-bj-learn-no-dup-inv* **by** (*auto simp*: *comp-def*)
next
case (*1 C′ S C F′ K F L*)
moreover
let *?C′ = remdups-mset C′*
have *L ∉# C′*
using ‹*F ⊨as CNot C′*› ‹*undefined-lit F L*› *Marked-Propagated-in-iff-in-lits-of in-CNot-implies-uminus*(*2*) **by** *blast*
then have *distinct-mset* (*?C′ + {#L#}*)
by (*metis count-mset-set*(*3*) *distinct-mset-remdups-mset distinct-mset-single-add less-irrefl-nat mem-set-mset-iff remdups-mset-def*)
moreover
have *no-dup F*
using ‹inv_{NOT} *S*› ‹*convert-trail-from-W* (*trail S*) = *F′ @ Marked K* () *# F*›
unfolding inv_{NOT}*-def*
by (*smt comp-apply distinct.simps*(*2*) *distinct-append list.simps*(*9*) *map-append no-dup-convert-from-W*)
then have *consistent-interp* (*lits-of F*)
using *distinctconsistent-interp* **by** *blast*
then have ¬ *tautology* (*C′*)
using ‹*F ⊨as CNot C′*› *consistent-CNot-not-tautology true-annots-true-cls* **by** *blast*
then have ¬ *tautology* (*?C′ + {#L#}*)
using ‹*F ⊨as CNot C′*› ‹*undefined-lit F L*› **by** (*metis CNot-remdups-mset Marked-Propagated-in-iff-in-lits-of add.commute in-CNot-uminus tautology-add-single tautology-remdups-mset true-annot-singleton true-annots-def*)
show *?case*
proof −
have *f2*: *no-dup* (*convert-trail-from-W* (*trail S*))
using ‹inv_{NOT} *S*› **unfolding** inv_{NOT}*-def* **by** (*simp add*: *o-def*)
have *f3*: *atm-of L ∈ atms-of-msu* (*clauses S*)
∪ *atm-of ‘ lits-of* (*convert-trail-from-W* (*trail S*))
using ‹*convert-trail-from-W* (*trail S*) = *F′ @ Marked K* () *# F*›
‹*atm-of L ∈ atms-of-msu* (*clauses S*) ∪ *atm-of ‘ lits-of* (*F′ @ Marked K* () *# F*)› **by** *auto*
have *f4*: *clauses S ⊨pm remdups-mset C′ + {#L#}*
by (*metis* (*no-types*) ‹*L ∉# C′*› ‹*clauses S ⊨pm C′ + {#L#}*› *remdups-mset-singleton-sum*(*2*) *true-clss-cls-remdups-mset union-commute*)
have *F ⊨as CNot* (*remdups-mset C′*)
by (*simp add*: ‹*F ⊨as CNot C′*›)
then show *?thesis*
using *f4 f3 f2* ‹¬ *tautology* (*remdups-mset C′ + {#L#}*)›
backjump-l.intros[*OF - f2*] *calculation*(*2−5,9*)
state-eq$_{NOT}$*-ref* **unfolding** *backjump-l-cond-def* **by** *blast*
qed
qed

sublocale $cdcl_W$ ⊆ $cdcl_{NOT}$*-merge-bj-learn-proxy2*
λS. convert-trail-from-W (*trail S*)
clauses
λL S. cons-trail (*convert-ann-literal-from-NOT L*) *S*
λS. tl-trail S
λC S. add-learned-cls C S

$\lambda C\ S.$ *remove-cls C S* λ- -. *True* inv_{NOT}
λ- S. *conflicting S = None backjump-l-cond*
by *unfold-locales*

sublocale $cdcl_W \subseteq cdcl_{NOT}$*-merge-bj-learn*
$\lambda S.$ *convert-trail-from-W* (*trail S*)
clauses
$\lambda L\ S.$ *cons-trail* (*convert-ann-literal-from-NOT L*) *S*
$\lambda S.$ *tl-trail S*
$\lambda C\ S.$ *add-learned-cls C S*
$\lambda C\ S.$ *remove-cls C S* λ- -. *True* inv_{NOT}
λ- S. *conflicting S = None backjump-l-cond*
apply *unfold-locales*
using *dpll-bj-no-dup* **apply** (*simp add*: *comp-def*)
using $cdcl_{NOT}$*-no-dup* **by** (*auto simp add*: *comp-def* $cdcl_{NOT}$*.simps*)

context $cdcl_W$
begin

Notations are lost while proving locale inclusion:

notation $state\text{-}eq_{NOT}$ (**infix** $\sim_{NOT}$ *50*)

7.2 Additional Lemmas between NOT and W states

lemma $trail_W$*-eq-reduce-trail-to*$_{NOT}$*-eq*:
trail S = trail T $\Longrightarrow$ *trail* (*reduce-trail-to*$_{NOT}$ *F S*) = *trail* (*reduce-trail-to*$_{NOT}$ *F T*)
proof (*induction F S arbitrary*: *T rule*: *reduce-trail-to*$_{NOT}$*.induct*)
case (*1 F S T*) **note** *IH = this*(*1*) **and** *tr = this*(*2*)
then have [] = *convert-trail-from-W* (*trail S*)
$\vee$ *length F = length* (*convert-trail-from-W* (*trail S*))
$\vee$ *trail* (*reduce-trail-to*$_{NOT}$ *F* (*tl-trail S*)) = *trail* (*reduce-trail-to*$_{NOT}$ *F* (*tl-trail T*))
using *IH* **by** (*metis* (*no-types*) *trail-tl-trail*)
then show *trail* (*reduce-trail-to*$_{NOT}$ *F S*) = *trail* (*reduce-trail-to*$_{NOT}$ *F T*)
using *tr* **by** (*metis* (*no-types*) *reduce-trail-to*$_{NOT}$*.elims*)
qed

lemma *trail-reduce-trail-to*$_{NOT}$*-add-learned-cls*:
no-dup (*trail S*) $\Longrightarrow$
trail (*reduce-trail-to*$_{NOT}$ *M* (*add-learned-cls D S*)) = *trail* (*reduce-trail-to*$_{NOT}$ *M S*)
by (*rule* $trail_W$*-eq-reduce-trail-to*$_{NOT}$*-eq*) *simp*

lemma *reduce-trail-to*$_{NOT}$*-reduce-trail-convert*:
reduce-trail-to$_{NOT}$ *C S = reduce-trail-to* (*convert-trail-from-NOT C*) *S*
apply (*induction C S rule*: *reduce-trail-to*$_{NOT}$*.induct*)
apply (*subst reduce-trail-to*$_{NOT}$*.simps*, *subst reduce-trail-to.simps*)
by *auto*

lemma *reduce-trail-to-length*:
length M = length M' $\Longrightarrow$ *reduce-trail-to M S = reduce-trail-to M' S*
apply (*induction M S arbitrary*: *rule*: *reduce-trail-to.induct*)
apply (*rename-tac F S*; *case-tac trail S* $\neq$ [] ; *case-tac length* (*trail S*) $\neq$ *length M'*)
by (*simp-all add*: *reduce-trail-to-length-ne*)

7.3 More lemmas conflict–propagate and backjumping

7.3.1 Termination

lemma $cdcl_W$*-cp-normalized-element-all-inv*:
assumes *inv*: $cdcl_W$*-all-struct-inv S*
obtains *T* **where** *full* $cdcl_W$*-cp S T*
using *assms* $cdcl_W$*-cp-normalized-element* **unfolding** $cdcl_W$*-all-struct-inv-def* **by** *blast*
thm *backtrackE*

lemma $cdcl_W$*-bj-measure*:
assumes $cdcl_W$*-bj S T* **and** $cdcl_W$*-M-level-inv S*
shows *length* (*trail S*) + (*if conflicting S = None then 0 else 1*)
$>$ *length* (*trail T*) + (*if conflicting T = None then 0 else 1*)
using *assms* **by** (*induction rule*: $cdcl_W$*-bj.induct*)
(*force dest*:*arg-cong*[*of - - length*]
intro: *get-all-marked-decomposition-exists-prepend*
elim!: *backtrack-levE*
simp: $cdcl_W$*-M-level-inv-def*)+

lemma *wf-*$cdcl_W$*-bj*:
wf {(*b*,*a*). $cdcl_W$*-bj a b* ∧ $cdcl_W$*-M-level-inv a*}
apply (*rule wfP-if-measure*[*of* λ-. *True*
\- λ*T*. *length* (*trail T*) + (*if conflicting T = None then 0 else 1*), *simplified*])
using $cdcl_W$*-bj-measure* **by** *blast*

lemma $cdcl_W$*-bj-exists-normal-form*:
assumes *lev*: $cdcl_W$*-M-level-inv S*
shows ∃ *T*. *full* $cdcl_W$*-bj S T*
proof −
obtain *T* **where** *T*: *full* (λ*a b*. $cdcl_W$*-bj a b* ∧ $cdcl_W$*-M-level-inv a*) *S T*
using *wf-exists-normal-form-full*[*OF wf-*$cdcl_W$*-bj*] **by** *auto*
then have $cdcl_W$*-bj*** *S T*
by (*auto dest*: *rtranclp-and-rtranclp-left simp*: *full-def*)
moreover
then have $cdcl_W{}^{**}$ *S T*
using *mono-rtranclp*[*of* $cdcl_W$*-bj* $cdcl_W$] $cdcl_W$*.simps* **by** *blast*
then have $cdcl_W$*-M-level-inv T*
using *rtranclp-*$cdcl_W$*-consistent-inv lev* **by** *auto*
ultimately show *?thesis* **using** *T* **unfolding** *full-def* **by** *auto*
qed

lemma *rtranclp-skip-state-decomp*:
assumes *skip*** *S T* **and** *no-dup* (*trail S*)
shows
∃ *M*. *trail S* = *M* @ *trail T* ∧ (∀ *m*∈*set M*. ¬*is-marked m*) **and**
T ∼ *delete-trail-and-rebuild* (*trail T*) *S*
using *assms* **by** (*induction rule*: *rtranclp-induct*)
(*auto simp del*: *state-simp simp*: *state-eq-def state-access-simp*)

7.3.2 More backjumping

Backjumping after skipping or jump directly **lemma** *rtranclp-skip-backtrack-backtrack*:
assumes
skip** *S T* **and**
backtrack T W **and**

cdcl$_W$*-all-struct-inv S*
shows *backtrack S W*
using *assms*
proof *induction*
case *base*
then show *?case* **by** *simp*
next
case (*step T V*) **note** *st* = *this*(*1*) **and** *skip* = *this*(*2*) **and** *IH* = *this*(*3*) **and** *bt* = *this*(*4*) **and** *inv* = *this*(*5*)
have *skip*** *S V*
using *st skip* **by** *auto*
then have *cdcl*$_W$*-all-struct-inv V*
using *rtranclp-mono*[*of skip cdcl*$_W$] *assms*(*3*) *rtranclp-cdcl*$_W$*-all-struct-inv-inv mono-rtranclp*
by (*auto dest*!: *bj other cdcl*$_W$*-bj.skip*)
then have *cdcl*$_W$*-M-level-inv V*
unfolding *cdcl*$_W$*-all-struct-inv-def* **by** *auto*
then obtain *N k M1 M2 K D L U i* **where**
V: *state V* = (*trail V*, *N*, *U*, *k*, *Some* (*D* + {#*L*#})) **and**
W: *state W* = (*Propagated L* (*D* + {#*L*#}) # *M1*, *N*, {#*D* + {#*L*#}#} + *U*,
get-maximum-level (*trail V*) *D*, *None*) **and**
decomp: (*Marked K* (*Suc i*) # *M1*, *M2*)
∈ *set* (*get-all-marked-decomposition* (*trail V*)) **and**
k = *get-maximum-level* (*trail V*) (*D* + {#*L*#}) **and**
lev-L: *get-level* (*trail V*) *L* = *k* **and**
undef: *undefined-lit M1 L* **and**
W ∼ *cons-trail* (*Propagated L* (*D* + {#*L*#}))
(*reduce-trail-to M1* (*add-learned-cls* (*D* + {#*L*#})
(*update-backtrack-lvl* (*get-maximum-level* (*trail V*) *D*) (*update-conflicting None V*))))**and**
lev-l-D: *backtrack-lvl V* = *get-maximum-level* (*trail V*) (*D* + {#*L*#}) **and**
conflicting V = *Some* (*D* + {#*L*#}) **and**
i: *i* = *get-maximum-level* (*trail V*) *D*
using *bt* **by** (*elim backtrack-levE*)
(*auto simp*: *cdcl*$_W$*-M-level-inv-decomp state-eq-def simp del*: *state-simp*)+
let *?D* = (*D* + {#*L*#})
obtain *L*′ *C*′ **where**
T: *state T* = (*Propagated L*′ *C*′ # *trail V*, *N*, *U*, *k*, *Some ?D*) **and**
V ∼ *tl-trail T* **and**
−*L*′ ∉# *?D* **and**
?D ≠ {#}
using *skip V* **by** *force*

let *?M* = *Propagated L*′ *C*′ # *trail V*
have *cdcl*$_W$** *S T* **using** *bj cdcl*$_W$*-bj.skip mono-rtranclp*[*of skip cdcl*$_W$ *S T*] *other st* **by** *meson*
then have *inv*′: *cdcl*$_W$*-all-struct-inv T*
using *rtranclp-cdcl*$_W$*-all-struct-inv-inv inv* **by** *blast*
have *M-lev*: *cdcl*$_W$*-M-level-inv T* **using** *inv*′ **unfolding** *cdcl*$_W$*-all-struct-inv-def* **by** *auto*
then have *n-d*′: *no-dup ?M*
using *T* **unfolding** *cdcl*$_W$*-M-level-inv-def* **by** *auto*

have *k* > *0*
using *decomp M-lev T V* **unfolding** *cdcl*$_W$*-M-level-inv-def* **by** *auto*
then have *atm-of L* ∈ *atm-of* ‘ *lits-of* (*trail V*)
using *lev-L get-rev-level-ge-0-atm-of-in V* **by** *fastforce*
then have *L-L*′: *atm-of L* ≠ *atm-of L*′
using *n-d*′ **unfolding** *lits-of-def* **by** *auto*

```
  have L'-M: atm-of L' ∉ atm-of ` lits-of (trail V)
    using n-d' unfolding lits-of-def by auto
  have ?M ⊨as CNot ?D
    using inv' T unfolding cdcl_W-conflicting-def cdcl_W-all-struct-inv-def by auto
  then have L' ∉# ?D
    using L-L' L'-M unfolding true-annots-def by (auto simp add: true-annot-def true-cls-def
      atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set Ball-mset-def
      split: split-if-asm)
  have [simp]: trail (reduce-trail-to M1 T) = M1
    by (metis (mono-tags, lifting) One-nat-def Pair-inject T ‹V ∼ tl-trail T› decomp
      diff-less in-get-all-marked-decomposition-trail-update-trail length-greater-0-conv
      length-tl lessI list.distinct(1) reduce-trail-to-length-ne state-eq-trail
      trail-reduce-trail-to-length-le trail-tl-trail)
  have skip** S V
    using st skip by auto
  have no-dup (trail S)
    using inv unfolding cdcl_W-all-struct-inv-def cdcl_W-M-level-inv-def by auto
  then have [simp]: init-clss S = N and [simp]: learned-clss S = U
    using rtranclp-skip-state-decomp[OF ‹skip** S V›] V
    by (auto simp del: state-simp simp: state-eq-def state-access-simp)
  then have W-S: W ∼ cons-trail (Propagated L (D + {#L#})) (reduce-trail-to M1
  (add-learned-cls (D + {#L#}) (update-backtrack-lvl i (update-conflicting None T))))
    using W i T undef M-lev by (auto simp del: state-simp simp: state-eq-def cdcl_W-M-level-inv-def)

  obtain M2' where
    (Marked K (i+1) # M1, M2') ∈ set (get-all-marked-decomposition ?M)
    using decomp V by (cases hd (get-all-marked-decomposition (trail V)),
      cases get-all-marked-decomposition (trail V)) auto
  moreover
    from L-L' have get-level ?M L = k
      using lev-L ‹-L' ∉# ?D› V by (auto split: split-if-asm)
  moreover
    have atm-of L' ∉ atms-of D
      using ‹L' ∉# ?D› ‹-L' ∉# ?D› by (simp add: atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set
        atms-of-def)
    then have get-level ?M L = get-maximum-level ?M (D+{#L#})
      using lev-l-D[symmetric] L-L' V lev-L by simp
  moreover have i = get-maximum-level ?M D
    using i ‹atm-of L' ∉ atms-of D› by auto
  moreover

  ultimately have backtrack T W
    using T(1) W-S by blast
  then show ?thesis using IH inv by blast
qed

lemma fst-get-all-marked-decomposition-prepend-not-marked:
  assumes ∀ m∈set MS. ¬ is-marked m
  shows set (map fst (get-all-marked-decomposition M))
    = set (map fst (get-all-marked-decomposition (MS @ M)))
    using assms apply (induction MS rule: ann-literal-list-induct)
    apply auto[2]
    by (rename-tac L m xs; case-tac get-all-marked-decomposition (xs @ M)) simp-all
```

See also ⟦*skip*** *?S ?T*; *backtrack ?T ?W*; $cdcl_W$*-all-struct-inv ?S*⟧ ⟹ *backtrack ?S ?W*

lemma *rtranclp-skip-backtrack-backtrack-end*:
  assumes
    skip: *skip*** *S T* **and**
    bt: *backtrack S W* **and**
    inv: $cdcl_W$*-all-struct-inv S*
  shows *backtrack T W*
  using *assms*
proof −
  have *M-lev*: $cdcl_W$*-M-level-inv S*
    using *bt inv* **unfolding** $cdcl_W$*-all-struct-inv-def* **by** (*auto elim*!: *backtrack-levE*)
  then obtain *k M M1 M2 K i D L N U* **where**
    S: *state S* = (*M*, *N*, *U*, *k*, *Some* (*D* + {#*L*#})) **and**
    W: *state W* = (*Propagated L* (*D* + {#*L*#}) # *M1*, *N*, {#*D* + {#*L*#}#} + *U*, *get-maximum-level M D*,
      None) **and**
    decomp: (*Marked K* (*i*+*1*) # *M1*, *M2*) ∈ *set* (*get-all-marked-decomposition M*) **and**
    lev-l: *get-level M L* = *k* **and**
    lev-l-D: *get-level M L* = *get-maximum-level M* (*D*+{#*L*#}) **and**
    i: *i* = *get-maximum-level M D* **and**
    undef: *undefined-lit M1 L*
    using *bt* **by** (*elim backtrack-levE*)
    (*simp-all add*: $cdcl_W$*-M-level-inv-decomp state-eq-def del*: *state-simp*)
  let *?D* = (*D* + {#*L*#})

  have [*simp*]: *no-dup* (*trail S*)
    using *M-lev* **by** (*auto simp*: $cdcl_W$*-M-level-inv-decomp*)
  have $cdcl_W$*-all-struct-inv T*
    using *mono-rtranclp*[*of skip* $cdcl_W$] **by** (*smt bj* $cdcl_W$*-bj.skip inv local.skip other*
      rtranclp-$cdcl_W$*-all-struct-inv-inv*)
  then have [*simp*]: *no-dup* (*trail T*)
    unfolding $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-def* **by** *auto*

  obtain *MS* M_T **where** *M*: *M* = *MS* @ M_T **and** M_T: M_T = *trail T* **and** *nm*: ∀ *m*∈*set MS*. ¬*is-marked m*
    using *rtranclp-skip-state-decomp*(*1*)[*OF skip*] *S M-lev* **by** *auto*
  have *T*: *state T* = (M_T, *N*, *U*, *k*, *Some ?D*)
    using M_T *rtranclp-skip-state-decomp*(*2*)[*of S T*] *skip S*
    by (*auto simp del*: *state-simp simp*: *state-eq-def state-access-simp*)

  have $cdcl_W$*-all-struct-inv T*
    apply (*rule rtranclp*-$cdcl_W$*-all-struct-inv-inv*[*OF - inv*])
    using *bj* $cdcl_W$*-bj.skip local.skip other rtranclp-mono*[*of skip* $cdcl_W$] **by** *blast*
  then have M_T ⊨*as CNot ?D*
    unfolding $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-conflicting-def* **using** *T* **by** *blast*
  have ∀ *L*∈#*?D*. *atm-of L* ∈ *atm-of* ‘ *lits-of* M_T
    proof −
      have *f1*: ⋀*l*. ¬ M_T ⊨*a* {#− *l*#} ∨ *atm-of l* ∈ *atm-of* ‘ *lits-of* M_T
        by (*simp add*: *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set in-lit-of-true-annot*
          lits-of-def)
      have ⋀*l*. *l* ∉# *D* ∨ − *l* ∈ *lits-of* M_T
        using ‹M_T ⊨*as CNot* (*D* + {#*L*#})› *multi-member-split* **by** *fastforce*
      then show *?thesis*
        using *f1* **by** (*meson* ‹M_T ⊨*as CNot* (*D* + {#*L*#})› *ball-msetI true-annots-CNot-all-atms-defined*)
    qed
  moreover have *no-dup M*

using *inv S* **unfolding** *cdcl$_W$-all-struct-inv-def cdcl$_W$-M-level-inv-def* **by** *auto*
ultimately have $\forall$ *L*$\in$#*?D. atm-of L* $\notin$ *atm-of ‘ lits-of MS*
unfolding *M* **unfolding** *lits-of-def* **by** *auto*
then have *H*: $\bigwedge$*L. L*$\in$#*?D* $\Longrightarrow$ *get-level M L = get-level M_T L*
unfolding *M* **by** (*fastforce simp*: *lits-of-def*)
have [*simp*]: *get-maximum-level M ?D = get-maximum-level M_T ?D*
by (*metis* ‹*M_T* $\models$*as CNot* (*D* + {#*L*#})› *M nm ball-msetI true-annots-CNot-all-atms-defined get-maximum-level-skip-un-marked-not-present*)

have *lev-l′*: *get-level M_T L = k*
using *lev-l* **by** (*auto simp*: *H*)
have [*simp*]: *trail* (*reduce-trail-to M1 T*) = *M1*
using *T decomp M nm* **by** (*smt M_T append-assoc beginning-not-marked-invert get-all-marked-decomposition-exists-prepend reduce-trail-to-trail-tl-trail-decomp*)
have *W*: *W* $\sim$ *cons-trail* (*Propagated L* (*D* + {#*L*#})) (*reduce-trail-to M1* (*add-learned-cls* (*D* + {#*L*#}) (*update-backtrack-lvl i* (*update-conflicting None T*))))
using *W T i decomp undef* **by** (*auto simp del*: *state-simp simp*: *state-eq-def*)

have *lev-l-D′*: *get-level M_T L = get-maximum-level M_T* (*D*+{#*L*#})
using *lev-l-D* **by** (*auto simp*: *H*)
have [*simp*]: *get-maximum-level M D = get-maximum-level M_T D*
proof −
have $\bigwedge$*ms m.* $\neg$ (*ms*::(′*v*, *nat*, ′*v literal multiset*) *ann-literal list*) $\models$*as CNot m* $\vee$ ($\forall$ *l*$\in$#*m. atm-of l* $\in$ *atm-of ‘ lits-of ms*)
by (*simp add*: *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set in-CNot-implies-uminus*(*2*))
then have $\forall$ *l*$\in$#*D. atm-of l* $\in$ *atm-of ‘ lits-of M_T*
using ‹*M_T* $\models$*as CNot* (*D* + {#*L*#})› **by** *auto*
then show *?thesis*
by (*metis M get-maximum-level-skip-un-marked-not-present nm*)
qed
then have *i′*: *i = get-maximum-level M_T D*
using *i* **by** *auto*
have *Marked K* (*i* + *1*) # *M1* $\in$ *set* (*map fst* (*get-all-marked-decomposition M*))
using *Set.imageI*[*OF decomp*, *of fst*] **by** *auto*
then have *Marked K* (*i* + *1*) # *M1* $\in$ *set* (*map fst* (*get-all-marked-decomposition M_T*))
using *fst-get-all-marked-decomposition-prepend-not-marked*[*OF nm*] **unfolding** *M* **by** *auto*
then obtain *M2′* **where** *decomp′*:(*Marked K* (*i*+*1*) # *M1*, *M2′*) $\in$ *set* (*get-all-marked-decomposition M_T*)
by *auto*
then show *backtrack T W*
using *backtrack.intros*[*OF T decomp′ lev-l′*] *lev-l-D′ i′ W* **by** *force*
qed

lemma *cdcl$_W$-bj-decomp-resolve-skip-and-bj*:
assumes *cdcl$_W$-bj** S T* **and** *inv*: *cdcl$_W$-M-level-inv S*
shows (*skip-or-resolve** S T* $\vee$ ($\exists$ *U. skip-or-resolve** S U* $\wedge$ *backtrack U T*))
using *assms*
proof *induction*
case *base*
then show *?case* **by** *simp*
next
case (*step T U*) **note** *st* = *this*(*1*) **and** *bj* = *this*(*2*) **and** *IH* = *this*(*3*)
have *IH*: *skip-or-resolve** S T*
proof −

{ **assume** ($\exists$ *U*. *skip-or-resolve*** *S U* $\wedge$ *backtrack U T*)
then obtain *V* **where**
bt: *backtrack V T* **and**
skip-or-resolve** *S V*
by *blast*
have *cdcl*$_W$** *S V*
using ‹*skip-or-resolve*** *S V*› *rtranclp-skip-or-resolve-rtranclp-cdcl*$_W$ **by** *blast*
then have *cdcl*$_W$*-M-level-inv V* **and** *cdcl*$_W$*-M-level-inv S*
using *rtranclp-cdcl*$_W$*-consistent-inv inv* **by** *blast*+
with *bj bt* **have** *False* **using** *backtrack-no-cdcl*$_W$*-bj* **by** *simp*
}
then show *?thesis* **using** *IH inv* **by** *blast*
qed
show *?case*
using *bj*
proof (*cases rule*: *cdcl*$_W$*-bj.cases*)
case *backtrack*
then show *?thesis* **using** *IH* **by** *blast*
qed (*metis* (*no-types*, *lifting*) *IH rtranclp.simps*)+
qed

lemma *resolve-skip-deterministic*:
resolve S T $\Longrightarrow$ *skip S U* $\Longrightarrow$ *False*
by *fastforce*

lemma *backtrack-unique*:
assumes
bt-T: *backtrack S T* **and**
bt-U: *backtrack S U* **and**
inv: *cdcl*$_W$*-all-struct-inv S*
shows *T* $\sim$ *U*
proof –
have *lev*: *cdcl*$_W$*-M-level-inv S*
using *inv* **unfolding** *cdcl*$_W$*-all-struct-inv-def* **by** *auto*
then obtain *M N U' k D L i K M1 M2* **where**
S: *state S* = (*M*, *N*, *U'*, *k*, *Some* (*D* + {#*L*#})) **and**
decomp: (*Marked K* (*i+1*) # *M1*, *M2*) $\in$ *set* (*get-all-marked-decomposition M*) **and**
get-level M L = *k* **and**
get-level M L = *get-maximum-level M* (*D*+{#*L*#}) **and**
get-maximum-level M D = *i* **and**
T: *state T* = (*Propagated L* ((*D*+{#*L*#})) # *M1* , *N*, {#*D* + {#*L*#}#} + *U'*, *i*, *None*) **and**
undef: *undefined-lit M1 L*
using *bt-T* **by** (*elim backtrack-levE*)
(*force simp*: *cdcl*$_W$*-M-level-inv-def state-eq-def simp del*: *state-simp*)+

obtain *D' L' i' K' M1' M2'* **where**
S': *state S* = (*M*, *N*, *U'*, *k*, *Some* (*D'* + {#*L'*#})) **and**
decomp': (*Marked K'* (*i'+1*) # *M1'*, *M2'*) $\in$ *set* (*get-all-marked-decomposition M*) **and**
get-level M L' = *k* **and**
get-level M L' = *get-maximum-level M* (*D'*+{#*L'*#}) **and**
get-maximum-level M D' = *i'* **and**
U: *state U* = (*Propagated L'* (*D'*+{#*L'*#}) # *M1'*, *N*, {#*D'* + {#*L'*#}#} +*U'*, *i'*, *None*) **and**
undef: *undefined-lit M1' L'*
using *bt-U lev S* **by** (*elim backtrack-levE*)
(*force simp*: *cdcl*$_W$*-M-level-inv-def state-eq-def simp del*: *state-simp*)+

obtain *c* **where** *M*: *M* = *c* @ *M2* @ *Marked K* (*i* + *1*) # *M1*
  using *decomp* **by** *auto*
obtain *c′* **where** *M′*: *M* = *c′* @ *M2′* @ *Marked K′* (*i′* + *1*) # *M1′*
  using *decomp′* **by** *auto*
have *marked*: *get-all-levels-of-marked M* = *rev* [*1*..<*1*+*k*]
  using *inv S* **unfolding** *cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-M-level-inv-def* **by** *auto*
then have *i* < *k*
  unfolding *M*
  by (*force simp add*: *rev-swap*[*symmetric*] *dest*!: *arg-cong*[*of - - set*])

have [*simp*]: *L* = *L′*
  proof (*rule ccontr*)
    assume ¬ *?thesis*
    then have *L′* ∈# *D*
      using *S* **unfolding** *S′* **by** (*fastforce simp*: *multiset-eq-iff split*: *split-if-asm*)
    then have *get-maximum-level M D* ≥ *k*
      using ‹*get-level M L′* = *k*› *get-maximum-level-ge-get-level* **by** *blast*
    then show *False* **using** ‹*get-maximum-level M D* = *i*› ‹*i* < *k*› **by** *auto*
  qed
then have [*simp*]: *D* = *D′*
  using *S S′* **by** *auto*
have [*simp*]: *i*=*i′* **using** ‹*get-maximum-level M D′* = *i′*› ‹*get-maximum-level M D* = *i*› **by** *auto*

Automation in a step later...

have *H*: ⋀*a A B*. *insert a A* = *B* ⟹ *a* : *B*
  by *blast*
have *get-all-levels-of-marked* (*c*@*M2*) = *rev* [*i*+*2*..<*1*+*k*] **and**
  get-all-levels-of-marked (*c′*@*M2′*) = *rev* [*i*+*2*..<*1*+*k*]
  using *marked* **unfolding** *M*
  using *marked* **unfolding** *M′*
  unfolding *rev-swap*[*symmetric*] **by** (*auto dest*: *append-cons-eq-upt-length-i-end*)
from *arg-cong*[*OF this*(*1*), *of set*] *arg-cong*[*OF this*(*2*), *of set*]
have
  dropWhile (λ*L*. ¬*is-marked L* ∨ *level-of L* ≠ *Suc i*) (*c* @ *M2*) = [] **and**
  dropWhile (λ*L*. ¬*is-marked L* ∨ *level-of L* ≠ *Suc i*) (*c′* @ *M2′*) = []
    unfolding *dropWhile-eq-Nil-conv Ball-def*
    by (*intro allI*; *rename-tac x*; *case-tac x*; *auto dest*!: *H simp add*: *in-set-conv-decomp*)+

then have *M1* = *M1′*
  using *arg-cong*[*OF M*, *of dropWhile* (λ*L*. ¬*is-marked L* ∨ *level-of L* ≠ *Suc i*)]
  unfolding *M′* **by** *auto*
then show *?thesis* **using** *T U* **by** (*auto simp del*: *state-simp simp*: *state-eq-def*)
qed

lemma *if-can-apply-backtrack-no-more-resolve*:
  assumes
    skip: *skip*** *S U* **and**
    bt: *backtrack S T* **and**
    inv: *cdcl*$_W$*-all-struct-inv S*
  shows ¬*resolve U V*
proof (*rule ccontr*)
  assume *resolve*: ¬¬*resolve U V*

  obtain *L C M N U′ k D* **where**
    U: *state U* = (*Propagated L* ((*C* + {#*L*#})) # *M*, *N*, *U′*, *k*, *Some* (*D* + {#−*L*#}))**and**

get-maximum-level (*Propagated L* (*C* + {#*L*#}) # *M*) *D* = *k* **and**
state V = (*M*, *N*, *U'*, *k*, *Some* (*D* #∪ *C*))
using *resolve* **by** *auto*
have $cdcl_W$*-all-struct-inv U*
using *mono-rtranclp*[*of skip* $cdcl_W$] **by** (*meson bj* $cdcl_W$*-bj.skip inv local.skip other rtranclp-*$cdcl_W$*-all-struct-inv-inv*)
then have [*iff*]: *no-dup* (*trail S*) $cdcl_W$*-M-level-inv S* **and** [*iff*]: *no-dup* (*trail U*)
using *inv* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-def* **by** *blast+*
then have
S: *init-clss S* = *N*
learned-clss S = *U'*
backtrack-lvl S = *k*
conflicting S = *Some* (*D* + {#−*L*#})
using *rtranclp-skip-state-decomp*(*2*)[*OF skip*] *U*
by (*auto simp del*: *state-simp simp*: *state-eq-def state-access-simp*)
obtain M_0 **where**
tr-S: *trail S* = M_0 @ *trail U* **and**
nm: ∀ *m*∈*set* M_0. ¬*is-marked m*
using *rtranclp-skip-state-decomp*[*OF skip*] **by** *blast*

obtain *M' D' L' i K M1 M2* **where**
S': *state S* = (*M'*, *N*, *U'*, *k*, *Some* (*D'* + {#*L'*#})) **and**
decomp: (*Marked K* (*i+1*) # *M1*, *M2*) ∈ *set* (*get-all-marked-decomposition M'*) **and**
get-level M' L' = *k* **and**
get-level M' L' = *get-maximum-level M'* (*D'*+{#*L'*#}) **and**
get-maximum-level M' D' = *i* **and**
undef: *undefined-lit M1 L'* **and**
T: *state T* = (*Propagated L'* (*D'*+{#*L'*#}) # *M1* , *N*, {#*D'* + {#*L'*#}#}+*U'*, *i*, *None*)
using *bt* **by** (*elim backtrack-levE*) (*fastforce simp*: *S state-eq-def simp del*:*state-simp*)+
obtain *c* **where** *M*: *M'* = *c* @ *M2* @ *Marked K* (*i* + *1*) # *M1*
using *get-all-marked-decomposition-exists-prepend*[*OF decomp*] **by** *auto*
have *marked*: *get-all-levels-of-marked M'* = *rev* [*1..<1+k*]
using *inv S'* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-def* **by** *auto*
then have *i* < *k*
unfolding *M* **by** (*force simp add*: *rev-swap*[*symmetric*] *dest*!: *arg-cong*[*of - - set*])

have *DD'*: *D'* + {#*L'*#} = *D* + {#−*L*#}
using *S S'* **by** *auto*
have [*simp*]: *L'* = −*L*
proof (*rule ccontr*)
assume ¬ *?thesis*
then have −*L* ∈# *D'*
using *DD'* **by** (*metis add-diff-cancel-right' diff-single-trivial diff-union-swap multi-self-add-other-not-self*)
moreover
have *M'*: *M'* = M_0 @ *Propagated L* ((*C* + {#*L*#})) # *M*
using *tr-S U S S'* **by** (*auto simp*: *lits-of-def*)
have *no-dup M'*
using *inv U S'* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-def* **by** *auto*
have *atm-L-notin-M*: *atm-of L* ∉ *atm-of* ‘ (*lits-of M*)
using ‹*no-dup M'*› *M' U S S'* **by** (*auto simp*: *lits-of-def*)
have *get-all-levels-of-marked M'* = *rev* [*1..<1+k*]
using *inv U S'* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-def* **by** *auto*
then have *get-all-levels-of-marked M* = *rev* [*1..<1+k*]
using *nm M' S' U* **by** (*simp add*: *get-all-levels-of-marked-no-marked*)

then have *get-lev-L*:
get-level(*Propagated L* (C + {#*L*#}) # *M*) *L* = *k*
using *get-level-get-rev-level-get-all-levels-of-marked*[*OF atm-L-notin-M*,
of [*Propagated L* ((C + {#*L*#}))]] **by** *simp*
have *atm-of L* $\notin$ *atm-of* ‘ (*lits-of* (*rev* M_0))
using ‹*no-dup M*′› *M*′ *U S*′ **by** (*auto simp*: *lits-of-def*)
then have *get-level M*′ *L* = *k*
using *get-rev-level-notin-end*[*of L rev* M_0
rev M @ *Propagated L* (C + {#*L*#}) # [] *0*]
using *tr-S get-lev-L M*′ *U S*′ **by** (*simp add*:*nm lits-of-def*)
ultimately have *get-maximum-level M*′ *D*′ $\geq$ *k*
by (*metis get-maximum-level-ge-get-level get-rev-level-uminus*)
then show *False*
using ‹*i* < *k*› **unfolding** ‹*get-maximum-level M*′ *D*′ = *i*› **by** *auto*
qed
have [*simp*]: *D* = *D*′ **using** *DD*′ **by** *auto*
have $cdcl_W{}^{**}$ *S U*
using *bj* $cdcl_W$*-bj.skip local.skip mono-rtranclp*[*of skip* $cdcl_W$ *S U*] *other* **by** *meson*
then have $cdcl_W$*-all-struct-inv U*
using *inv rtranclp-*$cdcl_W$*-all-struct-inv-inv* **by** *blast*
then have *Propagated L* ((C + {#*L*#})) # *M* $\models$*as CNot* (*D*′ + {#*L*′#})
using $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-conflicting-def U* **by** *auto*
then have $\forall$ *L*′$\in$#*D*. *atm-of L*′ $\in$ *atm-of* ‘ *lits-of* (*Propagated L* ((C + {#*L*#})) # *M*)
by (*metis CNot-plus CNot-singleton Un-insert-right* ‹*D* = *D*′› *true-annots-insert ball-msetI*
atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set in-CNot-implies-uminus(*2*)
sup-bot.comm-neutral)
then have *get-maximum-level M*′ *D* = *k*
using *tr-S nm U S*′
get-maximum-level-skip-un-marked-not-present[*of D*
Propagated L (C + {#*L*#}) # *M* M_0]
unfolding ‹*get-maximum-level* (*Propagated L* (C + {#*L*#}) # *M*) *D* = *k*›
unfolding ‹*D* = *D*′›
by *simp*
then show *False*
using ‹*get-maximum-level M*′ *D*′ = *i*› ‹*i* < *k*› **by** *auto*
qed

lemma *if-can-apply-resolve-no-more-backtrack*:
assumes
skip: $skip^{**}$ *S U* **and**
resolve: *resolve S T* **and**
inv: $cdcl_W$*-all-struct-inv S*
shows ¬*backtrack U V*
using *assms*
by (*meson if-can-apply-backtrack-no-more-resolve rtranclp.rtrancl-refl*
rtranclp-skip-backtrack-backtrack)

lemma *if-can-apply-backtrack-skip-or-resolve-is-skip*:
assumes
bt: *backtrack S T* **and**
skip: $skip\text{-}or\text{-}resolve^{**}$ *S U* **and**
inv: $cdcl_W$*-all-struct-inv S*
shows $skip^{**}$ *S U*
using *assms*(*2*,*3*,*1*)
by *induction* (*simp-all add*: *if-can-apply-backtrack-no-more-resolve*)

lemma $cdcl_W$*-bj-bj-decomp*:
  assumes $cdcl_W$*-bj*** S W **and** $cdcl_W$*-all-struct-inv* S
  shows
    $(\exists\, T\ U\ V.\ (\lambda S\ T.\ \textit{skip-or-resolve}\ S\ T \wedge \textit{no-step backtrack}\ S)^{**}\ S\ T$
        $\wedge\ (\lambda T\ U.\ \textit{resolve}\ T\ U \wedge \textit{no-step backtrack}\ T)\ T\ U$
        $\wedge\ \textit{skip}^{**}\ U\ V\ \wedge \textit{backtrack}\ V\ W)$
    $\vee\ (\exists\, T\ U.\ (\lambda S\ T.\ \textit{skip-or-resolve}\ S\ T \wedge \textit{no-step backtrack}\ S)^{**}\ S\ T$
        $\wedge\ (\lambda T\ U.\ \textit{resolve}\ T\ U \wedge \textit{no-step backtrack}\ T)\ T\ U \wedge \textit{skip}^{**}\ U\ W)$
    $\vee\ (\exists\, T.\ \textit{skip}^{**}\ S\ T\ \wedge \textit{backtrack}\ T\ W)$
    $\vee\ \textit{skip}^{**}\ S\ W$ (**is** *?RB S W* $\vee$ *?R S W* $\vee$ *?SB S W* $\vee$ *?S S W*)
  using *assms*
proof *induction*
  case *base*
  then show *?case* **by** *simp*
next
  case (*step W X*) **note** *st* = *this*(*1*) **and** *bj* = *this*(*2*) **and** *IH* = *this*(*3*)[*OF this*(*4*)] **and** *inv* = *this*(*4*)

  have ¬*?RB S W* **and** ¬*?SB S W*
    proof (*clarify*, *goal-cases*)
      case (*1 T U V*)
      have *skip-or-resolve*** *S T*
        using *1*(*1*) **by** (*auto dest*!: *rtranclp-and-rtranclp-left*)
      then show *False*
        by (*metis* (*no-types*, *lifting*) *1*(*2*) *1*(*4*) *1*(*5*) *backtrack-no-*$cdcl_W$*-bj*
          $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-all-struct-inv-inv* $cdcl_W$*-o.bj local.bj other*
          resolve rtranclp-$cdcl_W$*-all-struct-inv-inv rtranclp-skip-backtrack-backtrack*
          rtranclp-skip-or-resolve-rtranclp-$cdcl_W$ *step.prems*)
    next
      case *2*
      then show *?case* **by** (*meson assms*(*2*) $cdcl_W$*-all-struct-inv-def backtrack-no-*$cdcl_W$*-bj*
        local.bj rtranclp-skip-backtrack-backtrack)
    qed
  then have *IH*: *?R S W* $\vee$ *?S S W* **using** *IH* **by** *blast*

  have $cdcl_W{}^{**}$ *S W* **by** (*metis* $cdcl_W$*-o.bj mono-rtranclp other st*)
  then have *inv-W*: $cdcl_W$*-all-struct-inv W* **by** (*simp add*: *rtranclp-*$cdcl_W$*-all-struct-inv-inv*
    step.prems)
  consider
      (*BT*) X' **where** *backtrack W* X'
    | (*skip*) *no-step backtrack W* **and** *skip W X*
    | (*resolve*) *no-step backtrack W* **and** *resolve W X*
    using *bj* $cdcl_W$*-bj.cases* **by** *meson*
  then show *?case*
    proof *cases*
      case (*BT* X')
      then consider
          (*bt*) *backtrack W X*
        | (*sk*) *skip W X*
        using *bj if-can-apply-backtrack-no-more-resolve*[*of W W* X' *X*] *inv-W* $cdcl_W$*-bj.cases* **by** *fast*
      then show *?thesis*
        proof *cases*
          case *bt*
          then show *?thesis* **using** *IH* **by** *auto*
        next

```
      case sk
      then show ?thesis using IH by (meson rtranclp-trans r-into-rtranclp)
    qed
next
  case skip
  then show ?thesis using IH  by (meson rtranclp.rtrancl-into-rtrancl)
next
  case resolve note no-bt = this(1) and res = this(2)
  consider
      (RS) T U where
        (λS T. skip-or-resolve S T ∧ no-step backtrack S)** S T and
        resolve T U and
        no-step backtrack T and
        skip** U W
    | (S)  skip** S W
    using IH by auto
  then show ?thesis
    proof cases
      case (RS T U)
      have cdcl_W** S T
        using  RS(1) cdcl_W-bj.resolve cdcl_W-o.bj  other skip
        mono-rtranclp[of  (λS T. skip-or-resolve S T ∧ no-step backtrack S) cdcl_W  S T]
        by meson
      then have cdcl_W-all-struct-inv U
        by (meson RS(2) cdcl_W-all-struct-inv-inv cdcl_W-bj.resolve cdcl_W-o.bj other
          rtranclp-cdcl_W-all-struct-inv-inv step.prems)
      { fix U'
        assume skip** U U' and skip** U' W
        have cdcl_W-all-struct-inv U'
          using ‹cdcl_W-all-struct-inv U› ‹skip** U U'› rtranclp-cdcl_W-all-struct-inv-inv
             cdcl_W-o.bj rtranclp-mono[of skip cdcl_W] other skip by blast
        then have no-step backtrack U'
          using if-can-apply-backtrack-no-more-resolve[OF ‹skip** U' W› ] res by blast
      }
      with ‹skip** U W›
      have (λS T. skip-or-resolve S T ∧ no-step backtrack S)** U W
         proof induction
           case base
           then show ?case by simp
         next
          case (step V W) note st = this(1) and skip = this(2) and IH = this(3) and H = this(4)
           have ⋀U'. skip** U' V ⟹ skip** U' W
             using skip by auto
           then have (λS T. skip-or-resolve S T ∧ no-step backtrack S)** U V
             using IH H by blast
           moreover have (λS T. skip-or-resolve S T ∧ no-step backtrack S)** V W

             by (simp add: local.skip r-into-rtranclp st step.prems)
           ultimately show ?case by simp
         qed
      then show ?thesis
        proof -
          have f1: ∀p pa pb pc. ¬ p (pa) pb ∨ ¬ p** pb pc ∨ p** pa pc
            by (meson converse-rtranclp-into-rtranclp)
          have skip-or-resolve T U ∧ no-step backtrack T
```

using $RS(2)$ $RS(3)$ **by** *force*
then have $(\lambda p\ pa.\ \textit{skip-or-resolve}\ p\ pa \wedge \textit{no-step backtrack}\ p)^{**}\ T\ W$
proof −
have $(\exists\, vr19\ vr16\ vr17\ vr18.\ vr19\ (vr16{::}'st)\ vr17 \wedge vr19^{**}\ vr17\ vr18$
$\wedge \neg\ vr19^{**}\ vr16\ vr18)$
$\vee \neg\ (\textit{skip-or-resolve}\ T\ U \wedge \textit{no-step backtrack}\ T)$
$\vee \neg\ (\lambda uu\ uua.\ \textit{skip-or-resolve}\ uu\ uua \wedge \textit{no-step backtrack}\ uu)^{**}\ U\ W$
$\vee\ (\lambda uu\ uua.\ \textit{skip-or-resolve}\ uu\ uua \wedge \textit{no-step backtrack}\ uu)^{**}\ T\ W$
by *force*
then show *?thesis*
by (*metis* (*no-types*) ‹$(\lambda S\ T.\ \textit{skip-or-resolve}\ S\ T \wedge \textit{no-step backtrack}\ S)^{**}\ U\ W$›
‹*skip-or-resolve* T U ∧ *no-step backtrack* T› *f1*)
qed
then have $(\lambda p\ pa.\ \textit{skip-or-resolve}\ p\ pa \wedge \textit{no-step backtrack}\ p)^{**}\ S\ W$
using $RS(1)$ **by** *force*
then show *?thesis*
using *no-bt res* **by** *blast*
qed
next
case S
{ fix U'
assume $\textit{skip}^{**}\ S\ U'$ **and** $\textit{skip}^{**}\ U'\ W$
then have $\textit{cdcl}_W{}^{**}\ S\ U'$
using *mono-rtranclp*[*of skip* $\textit{cdcl}_W$ S U'] **by** (*simp add*: $\textit{cdcl}_W$*-o.bj other skip*)
then have $\textit{cdcl}_W$*-all-struct-inv* U'
by (*metis* (*no-types*, *hide-lams*) ‹$\textit{cdcl}_W$*-all-struct-inv* S›
rtranclp-$\textit{cdcl}_W$*-all-struct-inv-inv*)
then have *no-step backtrack* U'
using *if-can-apply-backtrack-no-more-resolve*[*OF* ‹$\textit{skip}^{**}\ U'\ W$›] *res* **by** *blast*
}
with S
have $(\lambda S\ T.\ \textit{skip-or-resolve}\ S\ T \wedge \textit{no-step backtrack}\ S)^{**}\ S\ W$
proof *induction*
case *base*
then show *?case* **by** *simp*
next
case (*step* V W) **note** $st = this(1)$ **and** $skip = this(2)$ **and** $IH = this(3)$ **and** $H = this(4)$
have $\bigwedge U'.\ \textit{skip}^{**}\ U'\ V \Longrightarrow \textit{skip}^{**}\ U'\ W$
using *skip* **by** *auto*
then have $(\lambda S\ T.\ \textit{skip-or-resolve}\ S\ T \wedge \textit{no-step backtrack}\ S)^{**}\ S\ V$
using *IH H* **by** *blast*
moreover have $(\lambda S\ T.\ \textit{skip-or-resolve}\ S\ T \wedge \textit{no-step backtrack}\ S)^{**}\ V\ W$

by (*simp add*: *local.skip r-into-rtranclp st step.prems*)
ultimately show *?case* **by** *simp*
qed
then show *?thesis* **using** *res no-bt* **by** *blast*
qed
qed
qed

The case distinction is needed, since $T \sim V$ does not imply that $R^{**}\ T\ V$.

lemma $\textit{cdcl}_W$*-bj-strongly-confluent*:
assumes
$\textit{cdcl}_W\textit{-bj}^{**}\ S\ V$ **and**

    $cdcl_W$-bj^{**} *S T* **and**
    n-s: *no-step* $cdcl_W$-*bj V* **and**
    inv: $cdcl_W$-*all-struct-inv S*
  shows $T \sim V \vee cdcl_W\text{-}bj^{**}\ T\ V$
  using *assms(2)*
proof *induction*
 case *base*
 then show *?case* **by** (*simp add*: *assms(1)*)
next
 case (*step T U*) **note** *st* = *this(1)* **and** *s-o-r* = *this(2)* **and** *IH* = *this(3)*
 have $cdcl_W^{**}$ *S T*
  using *st mono-rtranclp*[*of* $cdcl_W$-*bj* $cdcl_W$] *other* **by** *blast*
 then have *lev-T*: $cdcl_W$-*M-level-inv T*
  using *inv rtranclp*-$cdcl_W$-*consistent-inv*[*of S T*]
  unfolding $cdcl_W$-*all-struct-inv-def* **by** *auto*

 consider
    (*TV*) $T \sim V$
   | (*bj-TV*) $cdcl_W$-bj^{**} *T V*
  using *IH* **by** *blast*
 then show *?case*
  proof *cases*
   case *TV*
   have *no-step* $cdcl_W$-*bj T*
    using ‹$cdcl_W$-*M-level-inv T*› *n-s* $cdcl_W$-*bj-state-eq-compatible*[*of T - V*] *TV* **by** *auto*
   then show *?thesis*
    using *s-o-r* **by** *auto*
  next
   case *bj-TV*
   then obtain *U'* **where**
    T-U': $cdcl_W$-*bj T U'* **and**
    $cdcl_W$-bj^{**} *U' V*
    using *IH n-s s-o-r* **by** (*metis rtranclp-unfold tranclpD*)
   have $cdcl_W^{**}$ *S T*
    by (*metis* (*no-types*, *hide-lams*) *bj mono-rtranclp*[*of* $cdcl_W$-*bj* $cdcl_W$] *other st*)
   then have *inv-T*: $cdcl_W$-*all-struct-inv T*
    by (*metis* (*no-types*, *hide-lams*) *inv rtranclp*-$cdcl_W$-*all-struct-inv-inv*)

   have *lev-U*: $cdcl_W$-*M-level-inv U*
    using *s-o-r* $cdcl_W$-*consistent-inv lev-T other* **by** *blast*
   show *?thesis*
    using *s-o-r*
    proof *cases*
     case *backtrack*
     then obtain *V0* **where** $skip^{**}$ *T V0* **and** *backtrack V0 V*
      using *IH if-can-apply-backtrack-skip-or-resolve-is-skip*[*OF backtrack - inv-T*]
       $cdcl_W$-*bj-decomp-resolve-skip-and-bj*
       by (*meson bj-TV* $cdcl_W$-*bj.backtrack inv-T lev-T n-s*
        rtranclp-skip-backtrack-backtrack-end)
     then have $cdcl_W$-bj^{**} *T V0* **and** $cdcl_W$-*bj V0 V*
      using *rtranclp-mono*[*of skip* $cdcl_W$-*bj*] **by** *blast+*
     then show *?thesis*
      using ‹*backtrack V0 V*› ‹$skip^{**}$ *T V0*› *backtrack-unique inv-T local.backtrack*
      rtranclp-skip-backtrack-backtrack **by** *auto*
    next

case *resolve*
then have $U \sim U'$
by (*meson* T-U' $cdcl_W$*-bj.simps if-can-apply-backtrack-no-more-resolve inv-T resolve-skip-deterministic resolve-unique rtranclp.rtrancl-refl*)
then show *?thesis*
using ⟨$cdcl_W$*-bj*** U' V⟩ **unfolding** *rtranclp-unfold*
by (*meson* T-U' *bj* $cdcl_W$*-consistent-inv lev-T other state-eq-ref state-eq-sym tranclp-*$cdcl_W$*-bj-state-eq-compatible*)
next
case *skip*
consider
(sk) *skip* T U'
| (bt) *backtrack* T U'
using T-U' **by** (*meson* $cdcl_W$*-bj.cases local.skip resolve-skip-deterministic*)
then show *?thesis*
proof *cases*
case *sk*
then show *?thesis*
using ⟨$cdcl_W$*-bj*** U' V⟩ **unfolding** *rtranclp-unfold*
by (*meson* T-U' *bj* $cdcl_W$*-all-inv*(*3*) $cdcl_W$*-all-struct-inv-def inv-T local.skip other tranclp-*$cdcl_W$*-bj-state-eq-compatible skip-unique state-eq-ref*)
next
case *bt*
have *skip*$^{++}$ T U
using *local.skip* **by** *blast*
then show *?thesis*
using *bt* **by** (*metis* ⟨$cdcl_W$*-bj*** U' V⟩ *backtrack inv-T tranclp-unfold-begin rtranclp-skip-backtrack-backtrack-end tranclp-into-rtranclp*)
qed
qed
qed
qed

lemma $cdcl_W$*-bj-unique-normal-form*:
assumes
ST: $cdcl_W$*-bj*** S T **and** SU: $cdcl_W$*-bj*** S U **and**
n-s-U: *no-step* $cdcl_W$*-bj* U **and**
n-s-T: *no-step* $cdcl_W$*-bj* T **and**
inv: $cdcl_W$*-all-struct-inv* S
shows $T \sim U$
proof −
have $T \sim U \lor$ $cdcl_W$*-bj*** T U
using ST SU $cdcl_W$*-bj-strongly-confluent inv n-s-U* **by** *blast*
then show *?thesis*
by (*metis* (*no-types*) *n-s-T rtranclp-unfold state-eq-ref tranclp-unfold-begin*)
qed

lemma *full-*$cdcl_W$*-bj-unique-normal-form*:
assumes *full* $cdcl_W$*-bj* S T **and** *full* $cdcl_W$*-bj* S U **and**
inv: $cdcl_W$*-all-struct-inv* S
shows $T \sim U$
using $cdcl_W$*-bj-unique-normal-form assms* **unfolding** *full-def* **by** *blast*

7.4 CDCL FW

inductive *cdcl*$_W$*-merge-restart* :: *'st* ⇒ *'st* ⇒ *bool* **where**
fw-r-propagate: *propagate S S'* ⟹ *cdcl*$_W$*-merge-restart S S'* |
fw-r-conflict: *conflict S T* ⟹ *full cdcl*$_W$*-bj T U* ⟹ *cdcl*$_W$*-merge-restart S U* |
fw-r-decide: *decide S S'* ⟹ *cdcl*$_W$*-merge-restart S S'*|
fw-r-rf: *cdcl*$_W$*-rf S S'* ⟹ *cdcl*$_W$*-merge-restart S S'*

lemma *cdcl*$_W$*-merge-restart-cdcl*$_W$:
  assumes *cdcl*$_W$*-merge-restart S T*
  shows *cdcl*$_W$** *S T*
  using *assms*
proof *induction*
  case (*fw-r-conflict S T U*) **note** *confl* = *this*(*1*) **and** *bj* = *this*(*2*)
  have *cdcl*$_W$ *S T* **using** *confl* **by** (*simp add*: *cdcl*$_W$*.intros r-into-rtranclp*)
  moreover
    have *cdcl*$_W$*-bj*** *T U* **using** *bj* **unfolding** *full-def* **by** *auto*
    then have *cdcl*$_W$** *T U* **by** (*metis cdcl*$_W$*-o.bj mono-rtranclp other*)
  ultimately show *?case* **by** *auto*
qed (*simp-all add*: *cdcl*$_W$*-o.intros cdcl*$_W$*.intros r-into-rtranclp*)

lemma *cdcl*$_W$*-merge-restart-conflicting-true-or-no-step*:
  assumes *cdcl*$_W$*-merge-restart S T*
  shows *conflicting T* = *None* ∨ *no-step cdcl*$_W$ *T*
  using *assms*
proof *induction*
  case (*fw-r-conflict S T U*) **note** *confl* = *this*(*1*) **and** *n-s* = *this*(*2*)
  { fix *D V*
    assume *cdcl*$_W$ *U V* **and** *conflicting U* = *Some D*
    then have *False*
      using *n-s* **unfolding** *full-def*
      by (*induction rule*: *cdcl*$_W$*-all-rules-induct*) (*auto dest*!: *cdcl*$_W$*-bj.intros*)
  }
  then show *?case* **by** (*cases conflicting U*) *fastforce+*
qed (*auto simp add*: *cdcl*$_W$*-rf.simps*)

inductive *cdcl*$_W$*-merge* :: *'st* ⇒ *'st* ⇒ *bool* **where**
fw-propagate: *propagate S S'* ⟹ *cdcl*$_W$*-merge S S'* |
fw-conflict: *conflict S T* ⟹ *full cdcl*$_W$*-bj T U* ⟹ *cdcl*$_W$*-merge S U* |
fw-decide: *decide S S'* ⟹ *cdcl*$_W$*-merge S S'*|
fw-forget: *forget S S'* ⟹ *cdcl*$_W$*-merge S S'*

lemma *cdcl*$_W$*-merge-cdcl*$_W$*-merge-restart*:
  cdcl$_W$*-merge S T* ⟹ *cdcl*$_W$*-merge-restart S T*
  by (*meson cdcl*$_W$*-merge.cases cdcl*$_W$*-merge-restart.simps forget*)

lemma *rtranclp-cdcl*$_W$*-merge-tranclp-cdcl*$_W$*-merge-restart*:
  cdcl$_W$*-merge*** *S T* ⟹ *cdcl*$_W$*-merge-restart*** *S T*
  using *rtranclp-mono*[*of cdcl*$_W$*-merge cdcl*$_W$*-merge-restart*] *cdcl*$_W$*-merge-cdcl*$_W$*-merge-restart* **by** *blast*

lemma *cdcl*$_W$*-merge-rtranclp-cdcl*$_W$:
  cdcl$_W$*-merge S T* ⟹ *cdcl*$_W$** *S T*
  using *cdcl*$_W$*-merge-cdcl*$_W$*-merge-restart cdcl*$_W$*-merge-restart-cdcl*$_W$ **by** *blast*

lemma *rtranclp-cdcl*$_W$*-merge-rtranclp-cdcl*$_W$:
  cdcl$_W$*-merge*** *S T* ⟹ *cdcl*$_W$** *S T*

using *rtranclp-mono*[*of* $cdcl_W$*-merge* $cdcl_W{}^{**}$] $cdcl_W$*-merge-rtranclp-*$cdcl_W$ **by** *auto*

lemma $cdcl_W$*-merge-is-*$cdcl_{NOT}$*-merged-bj-learn*:
  assumes
    inv: $cdcl_W$*-all-struct-inv S* **and**
    $cdcl_W$:$cdcl_W$*-merge S T*
  shows $cdcl_{NOT}$*-merged-bj-learn S T*
    ∨ (*no-step* $cdcl_W$*-merge T* ∧ *conflicting T* ≠ *None*)
  using $cdcl_W$ *inv*
proof *induction*
  case (*fw-propagate S T*) **note** *propa* = *this*(*1*)
  then obtain *M N U k L C* **where**
    H: *state S* = (*M*, *N*, *U*, *k*, *None*) **and**
    CL: *C* + {#*L*#} ∈# *clauses S* **and**
    M-C: *M* ⊨*as CNot C* **and**
    undef: *undefined-lit* (*trail S*) *L* **and**
    T: *T* ∼ *cons-trail* (*Propagated L* (*C* + {#*L*#})) *S*
    using *propa* **by** *auto*
  have $propagate_{NOT}$ *S T*
    apply (*rule* $propagate_{NOT}$.$propagate_{NOT}$[*of* - *C L*])
    using *H CL T undef M-C* **by** (*auto simp*: *state-eq*$_{NOT}$*-def state-eq-def clauses-def*
      simp del: *state-simp*)
  then show *?case*
    using $cdcl_{NOT}$*-merged-bj-learn.intros*(*2*) **by** *blast*
next
  case (*fw-decide S T*) **note** *dec* = *this*(*1*) **and** *inv* = *this*(*2*)
  then obtain *L* **where**
    undef-L: *undefined-lit* (*trail S*) *L* **and**
    atm-L: *atm-of L* ∈ *atms-of-msu* (*init-clss S*) **and**
    T: *T* ∼ *cons-trail* (*Marked L* (*Suc* (*backtrack-lvl S*)))
      (*update-backtrack-lvl* (*Suc* (*backtrack-lvl S*)) *S*)
    by *auto*
  have $decide_{NOT}$ *S T*
    apply (*rule* $decide_{NOT}$.$decide_{NOT}$)
       using *undef-L* **apply** *simp*
     using *atm-L inv* **unfolding** $cdcl_W$*-all-struct-inv-def no-strange-atm-def clauses-def* **apply** *auto*[]
    using *T undef-L* **unfolding** *state-eq-def state-eq*$_{NOT}$*-def* **by** (*auto simp*: *clauses-def*)
  then show *?case* **using** $cdcl_{NOT}$*-merged-bj-learn-*$decide_{NOT}$ **by** *blast*
next
  case (*fw-forget S T*) **note** *rf* =*this*(*1*) **and** *inv* = *this*(*2*)
  then obtain *M N C U k* **where**
     S: *state S* = (*M*, *N*, {#*C*#} + *U*, *k*, *None*) **and**
     ¬ *M* ⊨*asm clauses S* **and**
     C ∉ *set* (*get-all-mark-of-propagated* (*trail S*)) **and**
     C-init: *C* ∉# *init-clss S* **and**
     C-le: *C* ∈# *learned-clss S* **and**
     T: *T* ∼ *remove-cls C S*
    by *auto*
  have *init-clss S* ⊨*pm C*
    using *inv C-le* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-learned-clause-def*
    by (*meson mem-set-mset-iff true-clss-clss-in-imp-true-clss-cls*)
  then have *S-C*: *clauses S* − *replicate-mset* (*count* (*clauses S*) *C*) *C* ⊨*pm C*
    using *C-init C-le* **unfolding** *clauses-def* **by** (*simp add*: *Un-Diff*)
  moreover have *H*: *init-clss S* + (*learned-clss S* − *replicate-mset* (*count* (*learned-clss S*) *C*) *C*)
    = *init-clss S* + *learned-clss S* − *replicate-mset* (*count* (*learned-clss S*) *C*) *C*

using *C-le C-init* **by** (*metis clauses-def clauses-remove-cls diff-zero gr0I init-clss-remove-cls learned-clss-remove-cls plus-multiset.rep-eq replicate-mset-0 semiring-normalization-rules*(*5*))
have $forget_{NOT}$ *S T*
apply (*rule* $forget_{NOT}.forget_{NOT}$)
using *S-C* **apply** *blast*
using *S* **apply** *simp*
using ‹*C* ∈# *learned-clss S*› **apply** (*simp add*: *clauses-def*)
using *T C-le C-init* **by** (*auto simp*: *state-eq-def Un-Diff* $state\text{-}eq_{NOT}$*-def clauses-def ac-simps H simp del*: *state-simp*)
then show *?case* **using** $cdcl_{NOT}$*-merged-bj-learn-forget*$_{NOT}$ **by** *blast*
next
case (*fw-conflict S T U*) **note** *confl* = *this*(*1*) **and** *bj* = *this*(*2*) **and** *inv* = *this*(*3*)
obtain C_S **where**
confl-T: *conflicting T* = *Some* C_S **and**
C_S: C_S ∈# *clauses S* **and**
tr-S-C_S: *trail S* ⊨*as CNot* C_S
using *confl* **by** *auto*
have $cdcl_W$*-all-struct-inv T*
using $cdcl_W$*.simps* $cdcl_W$*-all-struct-inv-inv confl inv* **by** *blast*
then have $cdcl_W$*-M-level-inv T*
unfolding $cdcl_W$*-all-struct-inv-def* **by** *auto*
then consider
(*no-bt*) *skip-or-resolve*** *T U*
| (*bt*) *T′* **where** *skip-or-resolve*** *T T′* **and** *backtrack T′ U*
using *bj rtranclp-*$cdcl_W$*-bj-skip-or-resolve-backtrack* **unfolding** *full-def* **by** *meson*
then show *?case*
proof *cases*
case *no-bt*
then have *conflicting U* ≠ *None*
using *confl* **by** (*induction rule*: *rtranclp-induct*) *auto*
moreover then have *no-step* $cdcl_W$*-merge U*
by (*auto simp*: $cdcl_W$*-merge.simps*)
ultimately show *?thesis* **by** *blast*
next
case *bt* **note** *s-or-r* = *this*(*1*) **and** *bt* = *this*(*2*)
have $cdcl_W{}^{**}$ *T T′*
using *s-or-r mono-rtranclp*[*of skip-or-resolve* $cdcl_W$] *rtranclp-skip-or-resolve-rtranclp-*$cdcl_W$
by *blast*
then have $cdcl_W$*-M-level-inv T′*
using *rtranclp-*$cdcl_W$*-consistent-inv* ‹$cdcl_W$*-M-level-inv T*› **by** *blast*
then obtain *M1 M2 i D L K* **where**
confl-T′: *conflicting T′* = *Some* (*D* + {#*L*#}) **and**
M1-M2:(*Marked K* (*i*+*1*) # *M1*, *M2*) ∈ *set* (*get-all-marked-decomposition* (*trail T′*)) **and**
get-level (*trail T′*) *L* = *backtrack-lvl T′* **and**
get-level (*trail T′*) *L* = *get-maximum-level* (*trail T′*) (*D*+{#*L*#}) **and**
get-maximum-level (*trail T′*) *D* = *i* **and**
undef-L: *undefined-lit M1 L* **and**
U: *U* ∼ *cons-trail* (*Propagated L* (*D*+{#*L*#}))
(*reduce-trail-to M1*
(*add-learned-cls* (*D* + {#*L*#})
(*update-backtrack-lvl i*
(*update-conflicting None T′*))))
using *bt* **by** (*auto elim*: *backtrack-levE*)

have [*simp*]: *clauses S* = *clauses T*
  using *confl* **by** *auto*
have [*simp*]: *clauses T* = *clauses T′*
  using *s-or-r*
  proof (*induction*)
    case *base*
    then show *?case* **by** *simp*
  next
    case (*step U V*) **note** *st* = *this*(*1*) **and** *s-o-r* = *this*(*2*) **and** *IH* = *this*(*3*)
    have *clauses U* = *clauses V*
      using *s-o-r* **by** *auto*
    then show *?case* **using** *IH* **by** *auto*
  qed
have *inv-T*: $cdcl_W$*-all-struct-inv T*
  by (*meson* $cdcl_W$*-cp.simps confl inv r-into-rtranclp rtranclp-*$cdcl_W$*-all-struct-inv-inv*
    rtranclp-$cdcl_W$*-cp-rtranclp-*$cdcl_W$)
have $cdcl_W{}^{**}$ *T T′*
  using *rtranclp-skip-or-resolve-rtranclp-*$cdcl_W$ *s-or-r* **by** *blast*
have *inv-T′*: $cdcl_W$*-all-struct-inv T′*
  using ‹$cdcl_W{}^{**}$ *T T′*› *inv-T rtranclp-*$cdcl_W$*-all-struct-inv-inv* **by** *blast*
have *inv-U*: $cdcl_W$*-all-struct-inv U*
  using $cdcl_W$*-merge-restart-*$cdcl_W$ *confl fw-r-conflict inv local.bj*
  rtranclp-$cdcl_W$*-all-struct-inv-inv* **by** *blast*

have [*simp*]: *init-clss S* = *init-clss T′*
  using ‹$cdcl_W{}^{**}$ *T T′*› $cdcl_W$*-init-clss confl* $cdcl_W$*-all-struct-inv-def conflict inv*
  by (*metis* ‹$cdcl_W$*-M-level-inv T*› *rtranclp-*$cdcl_W$*-init-clss*)
then have *atm-L*: *atm-of L* ∈ *atms-of-msu* (*clauses S*)
  using *inv-T′ confl-T′* **unfolding** $cdcl_W$*-all-struct-inv-def no-strange-atm-def clauses-def*
  by *auto*
obtain *M* **where** *tr-T*: *trail T* = *M* @ *trail T′*
  using *s-or-r* **by** (*induction rule*: *rtranclp-induct*) *auto*
obtain *M′* **where**
  tr-T′: *trail T′* = *M′* @ *Marked K* (*i+1*) # *tl* (*trail U*) **and**
  tr-U: *trail U* = *Propagated L* (*D* + {#*L*#}) # *tl* (*trail U*)
  using *U M1-M2 undef-L inv-T′* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-def*
  by *fastforce*
def *M′′* ≡ *M* @ *M′*
  have *tr-T*: *trail S* = *M′′* @ *Marked K* (*i+1*) # *tl* (*trail U*)
  using *tr-T tr-T′ confl* **unfolding** *M′′-def* **by** *auto*
have *init-clss T′* + *learned-clss S* ⊨*pm D* + {#*L*#}
  using *inv-T′ confl-T′* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-learned-clause-def clauses-def*
  by *simp*
have *reduce-trail-to* (*convert-trail-from-NOT* (*convert-trail-from-W M1*)) *S* =
  reduce-trail-to M1 S
  by (*rule reduce-trail-to-length*) *simp*
moreover have *trail* (*reduce-trail-to M1 S*) = *M1*
  apply (*rule reduce-trail-to-skip-beginning*[*of - M* @ *-* @ *M2* @ [*Marked K* (*Suc i*)]])
  using *confl M1-M2* ‹*trail T* = *M* @ *trail T′*›
    apply (*auto dest*!: *get-all-marked-decomposition-exists-prepend*
      elim!: *conflictE*)
    by (*rule sym*) *auto*
ultimately have [*simp*]: *trail* (*reduce-trail-to*$_{NOT}$ (*convert-trail-from-W M1*) *S*) = *M1*
  using *M1-M2 confl* **by** (*auto simp add*: *reduce-trail-to*$_{NOT}$*-reduce-trail-convert*)
have *every-mark-is-a-conflict U*

using *inv-U* **unfolding** *cdcl$_W$-all-struct-inv-def cdcl$_W$-conflicting-def* **by** *simp*
then have *tl* (*trail U*) $\models$*as CNot D*
by (*metis add-diff-cancel-left′ append-self-conv2 tr-U union-commute*)
have *backjump-l S U*
apply (*rule backjump-l*[*of - - - - - L*])
using *tr-T* **apply** *simp*
using *inv* **unfolding** *cdcl$_W$-all-struct-inv-def cdcl$_W$-M-level-inv-def*
apply (*simp add*: *comp-def*)
using *U M1-M2 confl undef-L M1-M2 inv-T′ inv* **unfolding** *cdcl$_W$-all-struct-inv-def cdcl$_W$-M-level-inv-def* **apply** (*auto simp*: *state-eq$_{NOT}$-def trail-reduce-trail-to$_{NOT}$-add-learned-cls*)[]
using C_S **apply** *simp*
using *tr-S-C_S* **apply** *simp*

using *U undef-L M1-M2 inv-T′ inv* **unfolding** *cdcl$_W$-all-struct-inv-def cdcl$_W$-M-level-inv-def* **apply** *auto*[]
using *undef-L atm-L* **apply** (*simp add*: *trail-reduce-trail-to$_{NOT}$-add-learned-cls*)
using ‹*init-clss T′* + *learned-clss S* $\models$*pm D* + {#*L*#}› **unfolding** *clauses-def* **apply** *simp*
apply (*metis* ‹*tl* (*trail U*) $\models$*as CNot D*› *convert-trail-from-W-true-annots*)
using *inv-T′ inv-U U confl-T′ undef-L M1-M2* **unfolding** *cdcl$_W$-all-struct-inv-def distinct-cdcl$_W$-state-def* **by** (*simp add*: *cdcl$_W$-M-level-inv-decomp backjump-l-cond-def*)
then show *?thesis* **using** *cdcl$_{NOT}$-merged-bj-learn-backjump-l* **by** *fast*
qed
qed

abbreviation *cdcl$_{NOT}$-restart* **where**
cdcl$_{NOT}$-restart $\equiv$ *restart-ops.cdcl$_{NOT}$-raw-restart cdcl$_{NOT}$ restart*

lemma *cdcl$_W$-merge-restart-is-cdcl$_{NOT}$-merged-bj-learn-restart-no-step*:
assumes
inv: *cdcl$_W$-all-struct-inv S* **and**
cdcl$_W$:*cdcl$_W$-merge-restart S T*
shows *cdcl$_{NOT}$-restart*** *S T* $\vee$ (*no-step cdcl$_W$-merge T* $\wedge$ *conflicting T* $\neq$ *None*)
proof −
consider
(*fw*) *cdcl$_W$-merge S T*
| (*fw-r*) *restart S T*
using *cdcl$_W$* **by** (*meson cdcl$_W$-merge-restart.simps cdcl$_W$-rf.cases fw-conflict fw-decide fw-forget fw-propagate*)
then show *?thesis*
proof *cases*
case *fw*
then have *IH*: *cdcl$_{NOT}$-merged-bj-learn S T* $\vee$ (*no-step cdcl$_W$-merge T* $\wedge$ *conflicting T* $\neq$ *None*)
using *inv cdcl$_W$-merge-is-cdcl$_{NOT}$-merged-bj-learn* **by** *blast*
have *invS*: *inv$_{NOT}$ S*
using *inv* **unfolding** *cdcl$_W$-all-struct-inv-def cdcl$_W$-M-level-inv-def* **by** *auto*
have *ff2*: *cdcl$_{NOT}$*$^{++}$ *S T* $\longrightarrow$ *cdcl$_{NOT}$*** *S T*
by (*meson tranclp-into-rtranclp*)
have *ff3*: *no-dup* (*convert-trail-from-W* (*trail S*))
using *invS* **by** (*simp add*: *comp-def*)
have *cdcl$_{NOT}$* $\leq$ *cdcl$_{NOT}$-restart*
by (*auto simp*: *restart-ops.cdcl$_{NOT}$-raw-restart.simps*)
then show *?thesis*
using *ff3 ff2 IH cdcl$_{NOT}$-merged-bj-learn-is-tranclp-cdcl$_{NOT}$ rtranclp-mono*[*of cdcl$_{NOT}$ cdcl$_{NOT}$-restart*] *invS predicate2D* **by** *blast*

next
case *fw-r*
then show *?thesis* **by** (*blast intro*: *restart-ops.cdcl*$_{NOT}$*-raw-restart.intros*)
qed
qed

abbreviation μ_{FW} :: *'st* ⇒ *nat* **where**
μ_{FW} *S* ≡ (*if no-step cdcl*$_W$*-merge S then 0 else 1+*μ_{CDCL}*'-merged* (*set-mset* (*init-clss S*)) *S*)

lemma *cdcl*$_W$*-merge-*μ_{FW}*-decreasing*:
assumes
inv: *cdcl*$_W$*-all-struct-inv S* **and**
fw: *cdcl*$_W$*-merge S T*
shows μ_{FW} *T* < μ_{FW} *S*
proof −
let *?A* = *init-clss S*
have *atm-clauses*: *atms-of-msu* (*clauses S*) ⊆ *atms-of-msu ?A*
using *inv* **unfolding** *cdcl*$_W$*-all-struct-inv-def no-strange-atm-def clauses-def* **by** *auto*
have *atm-trail*: *atm-of ' lits-of* (*trail S*) ⊆ *atms-of-msu ?A*
using *inv* **unfolding** *cdcl*$_W$*-all-struct-inv-def no-strange-atm-def clauses-def* **by** *auto*
have *n-d*: *no-dup* (*trail S*)
using *inv* **unfolding** *cdcl*$_W$*-all-struct-inv-def* **by** (*auto simp*: *cdcl*$_W$*-M-level-inv-decomp*)
have [*simp*]: ¬ *no-step cdcl*$_W$*-merge S*
using *fw* **by** *auto*
have [*simp*]: *init-clss S* = *init-clss T*
using *cdcl*$_W$*-merge-restart-cdcl*$_W$[*of S T*] *inv rtranclp-cdcl*$_W$*-init-clss*
unfolding *cdcl*$_W$*-all-struct-inv-def*
by (*meson cdcl*$_W$*-merge.simps cdcl*$_W$*-merge-restart.simps cdcl*$_W$*-rf.simps fw*)
consider
(*merged*) *cdcl*$_{NOT}$*-merged-bj-learn S T*
| (*n-s*) *no-step cdcl*$_W$*-merge T*
using *cdcl*$_W$*-merge-is-cdcl*$_{NOT}$*-merged-bj-learn inv fw* **by** *blast*
then show *?thesis*
proof *cases*
case *merged*
then show *?thesis*
using *cdcl*$_{NOT}$*-decreasing-measure'*[*OF - - atm-clauses*] *atm-trail n-d*
by (*auto split*: *split-if simp*: *comp-def*)
next
case *n-s*
then show *?thesis* **by** *simp*
qed
qed

lemma *wf-cdcl*$_W$*-merge*: *wf* {(*T*, *S*). *cdcl*$_W$*-all-struct-inv S* ∧ *cdcl*$_W$*-merge S T*}
apply (*rule wfP-if-measure*[*of - -* μ_{FW}])
using *cdcl*$_W$*-merge-*μ_{FW}*-decreasing* **by** *blast*

lemma *cdcl*$_W$*-all-struct-inv-tranclp-cdcl*$_W$*-merge-tranclp-cdcl*$_W$*-merge-cdcl*$_W$*-all-struct-inv*:
assumes
inv: *cdcl*$_W$*-all-struct-inv b*
cdcl$_W$*-merge*$^{++}$ *b a*
shows (λ*S T*. *cdcl*$_W$*-all-struct-inv S* ∧ *cdcl*$_W$*-merge S T*)$^{++}$ *b a*
using *assms*(*2*)
proof *induction*

case *base*
then show *?case* **using** *inv* **by** *auto*
next
case (*step c d*) **note** *st* =*this*(*1*) **and** *fw* = *this*(*2*) **and** *IH* = *this*(*3*)
have *cdcl*$_W$*-all-struct-inv c*
using *tranclp-into-rtranclp*[*OF st*] *cdcl*$_W$*-merge-rtranclp-cdcl*$_W$
assms(*1*) *rtranclp-cdcl*$_W$*-all-struct-inv-inv rtranclp-mono*[*of cdcl*$_W$*-merge cdcl*$_W$**] **by** *fastforce*
then have ($\lambda S\ T.$ *cdcl*$_W$*-all-struct-inv S* $\wedge$ *cdcl*$_W$*-merge S T*)$^{++}$ *c d*
using *fw* **by** *auto*
then show *?case* **using** *IH* **by** *auto*
qed

lemma *wf-tranclp-cdcl*$_W$*-merge*: *wf* {(*T*, *S*). *cdcl*$_W$*-all-struct-inv S* $\wedge$ *cdcl*$_W$*-merge*$^{++}$ *S T*}
using *wf-trancl*[*OF wf-cdcl*$_W$*-merge*]
apply (*rule wf-subset*)
by (*auto simp*: *trancl-set-tranclp*
cdcl$_W$*-all-struct-inv-tranclp-cdcl*$_W$*-merge-tranclp-cdcl*$_W$*-merge-cdcl*$_W$*-all-struct-inv*)

lemma *backtrack-is-full1-cdcl*$_W$*-bj*:
assumes *bt*: *backtrack S T* **and** *inv*: *cdcl*$_W$*-M-level-inv S*
shows *full1 cdcl*$_W$*-bj S T*
proof −
have *no-step cdcl*$_W$*-bj T*
using *bt inv backtrack-no-cdcl*$_W$*-bj* **by** *blast*
moreover have *cdcl*$_W$*-bj*$^{++}$ *S T*
using *bt* **by** *auto*
ultimately show *?thesis* **unfolding** *full1-def* **by** *blast*
qed

lemma *rtrancl-cdcl*$_W$*-conflicting-true-cdcl*$_W$*-merge-restart*:
assumes *cdcl*$_W$** *S V* **and** *inv*: *cdcl*$_W$*-M-level-inv S* **and** *conflicting S* = *None*
shows (*cdcl*$_W$*-merge-restart*** *S V* $\wedge$ *conflicting V* = *None*)
$\vee$ ($\exists$ *T U*. *cdcl*$_W$*-merge-restart*** *S T* $\wedge$ *conflicting V* $\neq$ *None* $\wedge$ *conflict T U* $\wedge$ *cdcl*$_W$*-bj*** *U V*)
using *assms*
proof *induction*
case *base*
then show *?case* **by** *simp*
next
case (*step U V*) **note** *st* = *this*(*1*) **and** *cdcl*$_W$ = *this*(*2*) **and** *IH* = *this*(*3*)[*OF this*(*4*−)] **and**
confl[*simp*] = *this*(*5*) **and** *inv* = *this*(*4*)
from *cdcl*$_W$
show *?case*
proof (*cases*)
case *propagate*
moreover then have *conflicting U* = *None*
by *auto*
moreover have *conflicting V* = *None*
using *propagate* **by** *auto*
ultimately show *?thesis* **using** *IH cdcl*$_W$*-merge-restart.fw-r-propagate*[*of U V*] **by** *auto*
next
case *conflict*
moreover then have *conflicting U* = *None*
by *auto*
moreover have *conflicting V* $\neq$ *None*
using *conflict* **by** *auto*

```
  ultimately show ?thesis using IH by auto
next
  case other
  then show ?thesis
    proof cases
      case decide
      moreover then have conflicting U = None
        by auto
      ultimately show ?thesis using IH cdcl_W-merge-restart.fw-r-decide[of U V] by auto
    next
      case bj
      moreover {
        assume skip-or-resolve U V
        have f1: cdcl_W-bj++ U V
          by (simp add: local.bj tranclp.r-into-trancl)
        obtain T T' :: 'st where
          f2: cdcl_W-merge-restart** S U
            ∨ cdcl_W-merge-restart** S T ∧ conflicting U ≠ None
              ∧ conflict T T' ∧ cdcl_W-bj** T' U
          using IH confl by blast
        then have ?thesis
          proof -
            have conflicting V ≠ None ∧ conflicting U ≠ None
              using ‹skip-or-resolve U V› by auto
            then show ?thesis
              by (metis (no-types) IH f1 rtranclp-trans tranclp-into-rtranclp)
          qed
      }
      moreover {
        assume backtrack U V
        then have conflicting U ≠ None by auto
        then obtain T T' where
          cdcl_W-merge-restart** S T and
          conflicting U ≠ None and
          conflict T T' and
          cdcl_W-bj** T' U
          using IH confl by meson
        have invU: cdcl_W-M-level-inv U
          using inv rtranclp-cdcl_W-consistent-inv step.hyps(1) by blast
        then have conflicting V = None
          using ‹backtrack U V› inv by (auto elim: backtrack-levE
            simp: cdcl_W-M-level-inv-decomp)
        have full cdcl_W-bj T' V
          apply (rule rtranclp-fullI[of cdcl_W-bj T' U V])
            using ‹cdcl_W-bj** T' U› apply fast
          using ‹backtrack U V› backtrack-is-full1-cdcl_W-bj invU unfolding full1-def full-def
          by blast
        then have ?thesis
          using cdcl_W-merge-restart.fw-r-conflict[of T T' V] ‹conflict T T'›
          ‹cdcl_W-merge-restart** S T› ‹conflicting V = None› by auto
      }
      ultimately show ?thesis by (auto simp: cdcl_W-bj.simps)
  qed
next
  case rf
```

moreover then have *conflicting U = None* **and** *conflicting V = None*
by (*auto simp*: *cdcl*$_W$*-rf.simps*)
ultimately show *?thesis* **using** *IH cdcl*$_W$*-merge-restart.fw-r-rf*[*of U V*] **by** *auto*
qed
qed

lemma *no-step-cdcl*$_W$*-no-step-cdcl*$_W$*-merge-restart*: *no-step cdcl*$_W$ *S* $\Longrightarrow$ *no-step cdcl*$_W$*-merge-restart S*
by (*auto simp*: *cdcl*$_W$*.simps cdcl*$_W$*-merge-restart.simps cdcl*$_W$*-o.simps cdcl*$_W$*-bj.simps*)

lemma *no-step-cdcl*$_W$*-merge-restart-no-step-cdcl*$_W$:
assumes
conflicting S = None **and**
cdcl$_W$*-M-level-inv S* **and**
no-step cdcl$_W$*-merge-restart S*
shows *no-step cdcl*$_W$ *S*
proof −
{ **fix** *S'*
assume *conflict S S'*
then have *cdcl*$_W$ *S S'* **using** *cdcl*$_W$*.conflict* **by** *auto*
then have *cdcl*$_W$*-M-level-inv S'*
using *assms(2) cdcl*$_W$*-consistent-inv* **by** *blast*
then obtain *S''* **where** *full cdcl*$_W$*-bj S' S''*
using *cdcl*$_W$*-bj-exists-normal-form*[*of S'*] **by** *auto*
then have *False*
using ‹*conflict S S'*› *assms(3) fw-r-conflict* **by** *blast*
}
then show *?thesis*
using *assms* **unfolding** *cdcl*$_W$*.simps cdcl*$_W$*-merge-restart.simps cdcl*$_W$*-o.simps cdcl*$_W$*-bj.simps*
by *fastforce*
qed

lemma *rtranclp-cdcl*$_W$*-merge-restart-no-step-cdcl*$_W$*-bj*:
assumes
cdcl$_W$*-merge-restart*** *S T* **and**
conflicting S = None
shows *no-step cdcl*$_W$*-bj T*
using *assms*
apply (*induction rule*: *rtranclp-induct*)
apply (*fastforce simp*: *cdcl*$_W$*-bj.simps cdcl*$_W$*-rf.simps cdcl*$_W$*-merge-restart.simps full-def*)
apply (*fastforce simp*: *cdcl*$_W$*-bj.simps cdcl*$_W$*-rf.simps cdcl*$_W$*-merge-restart.simps full-def*)

done

If *conflicting* $S \neq$ *None*, we cannot say anything.

Remark that this theorem does not say anything about well-foundedness: even if you know that one relation is well-founded, it only states that the normal forms are shared.

lemma *conflicting-true-full-cdcl*$_W$*-iff-full-cdcl*$_W$*-merge*:
assumes *confl*: *conflicting S = None* **and** *lev*: *cdcl*$_W$*-M-level-inv S*
shows *full cdcl*$_W$ *S V* $\longleftrightarrow$ *full cdcl*$_W$*-merge-restart S V*
proof
assume *full*: *full cdcl*$_W$*-merge-restart S V*
then have *st*: *cdcl*$_W$** *S V*
using *rtranclp-mono*[*of cdcl*$_W$*-merge-restart cdcl*$_W$**] *cdcl*$_W$*-merge-restart-cdcl*$_W$
unfolding *full-def* **by** *auto*

have *n-s*: *no-step* $cdcl_W$*-merge-restart V*
using *full* **unfolding** *full-def* **by** *auto*
have *n-s-bj*: *no-step* $cdcl_W$*-bj V*
using *rtranclp-*$cdcl_W$*-merge-restart-no-step-*$cdcl_W$*-bj confl full* **unfolding** *full-def* **by** *auto*
have $\bigwedge S'$. *conflict V S'* $\Longrightarrow$ $cdcl_W$*-M-level-inv S'*
using $cdcl_W$*.conflict* $cdcl_W$*-consistent-inv lev rtranclp-*$cdcl_W$*-consistent-inv st* **by** *blast*
then have $\bigwedge S'$. *conflict V S'* $\Longrightarrow$ *False*
using *n-s n-s-bj* $cdcl_W$*-bj-exists-normal-form* $cdcl_W$*-merge-restart.simps* **by** *meson*
then have *n-s-*$cdcl_W$: *no-step* $cdcl_W$ *V*
using *n-s n-s-bj* **by** (*auto simp*: $cdcl_W$*.simps* $cdcl_W$*-o.simps* $cdcl_W$*-merge-restart.simps*)
then show *full* $cdcl_W$ *S V* **using** *st* **unfolding** *full-def* **by** *auto*
next
assume *full*: *full* $cdcl_W$ *S V*
have *no-step* $cdcl_W$*-merge-restart V*
using *full no-step-*$cdcl_W$*-no-step-*$cdcl_W$*-merge-restart* **unfolding** *full-def* **by** *blast*
moreover
consider
(*fw*) $cdcl_W$*-merge-restart*** *S V* **and** *conflicting V = None*
| (*bj*) *T U* **where**
$cdcl_W$*-merge-restart*** *S T* **and**
conflicting V $\neq$ *None* **and**
conflict T U **and**
$cdcl_W$*-bj*** *U V*
using *full rtrancl-*$cdcl_W$*-conflicting-true-*$cdcl_W$*-merge-restart confl lev* **unfolding** *full-def*
by *meson*
then have $cdcl_W$*-merge-restart*** *S V*
proof *cases*
case *fw*
then show *?thesis* **by** *fast*
next
case (*bj T U*)
have *no-step* $cdcl_W$*-bj V*
using *full* **unfolding** *full-def* **by** (*meson* $cdcl_W$*-o.bj other*)
then have *full* $cdcl_W$*-bj U V*
using ‹ $cdcl_W$*-bj*** *U V*› **unfolding** *full-def* **by** *auto*
then have $cdcl_W$*-merge-restart T V*
using ‹*conflict T U*› $cdcl_W$*-merge-restart.fw-r-conflict* **by** *blast*
then show *?thesis* **using** ‹$cdcl_W$*-merge-restart*** *S T*› **by** *auto*
qed
ultimately show *full* $cdcl_W$*-merge-restart S V* **unfolding** *full-def* **by** *fast*
qed

lemma *init-state-true-full-*$cdcl_W$*-iff-full-*$cdcl_W$*-merge*:
shows *full* $cdcl_W$ (*init-state N*) *V* $\longleftrightarrow$ *full* $cdcl_W$*-merge-restart* (*init-state N*) *V*
by (*rule conflicting-true-full-*$cdcl_W$*-iff-full-*$cdcl_W$*-merge*) *auto*

7.5 FW with strategy

7.5.1 The intermediate step

inductive $cdcl_W$*-s'* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **where**
conflict': *full1* $cdcl_W$*-cp S S'* $\Longrightarrow$ $cdcl_W$*-s' S S'* |
decide': *decide S S'* $\Longrightarrow$ *no-step* $cdcl_W$*-cp S* $\Longrightarrow$ *full* $cdcl_W$*-cp S' S''* $\Longrightarrow$ $cdcl_W$*-s' S S''* |
bj': *full1* $cdcl_W$*-bj S S'* $\Longrightarrow$ *no-step* $cdcl_W$*-cp S* $\Longrightarrow$ *full* $cdcl_W$*-cp S' S''* $\Longrightarrow$ $cdcl_W$*-s' S S''*

```
inductive-cases cdcl_W-s'E: cdcl_W-s' S T

lemma rtranclp-cdcl_W-bj-full1-cdclp-cdcl_W-stgy:
  cdcl_W-bj** S S' ⟹ full cdcl_W-cp S' S'' ⟹ cdcl_W-stgy** S S''
proof (induction rule: converse-rtranclp-induct)
  case base
  then show ?case by (metis cdcl_W-stgy.conflict' full-unfold rtranclp.simps)
next
  case (step T U) note st =this(2) and bj = this(1) and IH = this(3)[OF this(4)]
  have no-step cdcl_W-cp T
    using bj by (auto simp add: cdcl_W-bj.simps)
  consider
      (U) U = S'
    | (U') U' where cdcl_W-bj U U' and cdcl_W-bj** U' S'
    using st by (metis converse-rtranclpE)
  then show ?case
    proof cases
      case U
      then show ?thesis
        using ‹no-step cdcl_W-cp T› cdcl_W-o.bj local.bj other' step.prems by (meson r-into-rtranclp)
    next
      case U' note U' = this(1)
      have no-step cdcl_W-cp U
        using U' by (fastforce simp: cdcl_W-cp.simps cdcl_W-bj.simps)
      then have full cdcl_W-cp U U
        by (simp add: full-unfold)
      then have cdcl_W-stgy T U
        using ‹no-step cdcl_W-cp T› cdcl_W-stgy.simps local.bj cdcl_W-o.bj by meson
      then show ?thesis using IH by auto
    qed
qed

lemma cdcl_W-s'-is-rtranclp-cdcl_W-stgy:
  cdcl_W-s' S T ⟹ cdcl_W-stgy** S T
  apply (induction rule: cdcl_W-s'.induct)
    apply (auto intro: cdcl_W-stgy.intros)[]
   apply (meson decide other' r-into-rtranclp)
  by (metis full1-def rtranclp-cdcl_W-bj-full1-cdclp-cdcl_W-stgy tranclp-into-rtranclp)

lemma cdcl_W-cp-cdcl_W-bj-bissimulation:
  assumes
    full cdcl_W-cp T U and
    cdcl_W-bj** T T' and
    cdcl_W-all-struct-inv T and
    no-step cdcl_W-bj T'
  shows full cdcl_W-cp T' U
    ∨ (∃ U' U''. full cdcl_W-cp T' U'' ∧ full1 cdcl_W-bj U U' ∧ full cdcl_W-cp U' U'' ∧ cdcl_W-s'** U U'')
  using assms(2,1,3,4)
proof (induction rule: rtranclp-induct)
  case base
  then show ?case by blast
next
  case (step T' T'') note st = this(1) and bj = this(2) and IH = this(3)[OF this(4,5)] and
    full = this(4) and inv = this(5)
  have cdcl_W** T T''
```

by (*metis* (*no-types*, *lifting*) *cdcl*$_W$*-o.bj local.bj mono-rtranclp*[*of cdcl*$_W$*-bj cdcl*$_W$ T T''] *other st rtranclp.rtrancl-into-rtrancl*)
then have *inv-*T'': *cdcl*$_W$*-all-struct-inv* T''
using *inv rtranclp-cdcl*$_W$*-all-struct-inv-inv* **by** *blast*
have *cdcl*$_W$*-bj*$^{++}$ T T''
using *local.bj st* **by** *auto*
have *full1 cdcl*$_W$*-bj* T T''
by (*metis* ‹*cdcl*$_W$*-bj*$^{++}$ T T''› *full1-def step.prems*(*3*))
then have $T = U$
proof −
obtain Z **where** *cdcl*$_W$*-bj* T Z
by (*meson tranclpD* ‹*cdcl*$_W$*-bj*$^{++}$ T T''›)
{ assume *cdcl*$_W$*-cp*$^{++}$ T U
then obtain Z' **where** *cdcl*$_W$*-cp* T Z'
by (*meson tranclpD*)
then have *False*
using ‹*cdcl*$_W$*-bj* T Z› **by** (*fastforce simp*: *cdcl*$_W$*-bj.simps cdcl*$_W$*-cp.simps*)
}
then show *?thesis*
using *full* **unfolding** *full-def rtranclp-unfold* **by** *blast*
qed
obtain U'' **where** *full cdcl*$_W$*-cp* T'' U''
using *cdcl*$_W$*-cp-normalized-element-all-inv inv-*T'' **by** *blast*
moreover then have *cdcl*$_W$*-stgy*** U U''
by (*metis* ‹$T = U$› ‹*cdcl*$_W$*-bj*$^{++}$ T T''› *rtranclp-cdcl*$_W$*-bj-full1-cdclp-cdcl*$_W$*-stgy rtranclp-unfold*)
moreover have *cdcl*$_W$*-s*$'^{**}$ U U''
proof −
obtain *ss* :: $'st \Rightarrow 'st$ **where**
f1: $\forall x2. (\exists v3.$ *cdcl*$_W$*-cp* $x2\ v3) =$ *cdcl*$_W$*-cp* $x2$ (*ss* $x2$)
by *moura*
have ¬ *cdcl*$_W$*-cp* U (*ss* U)
by (*meson full full-def*)
then show *?thesis*
using *f1* **by** (*metis* (*no-types*) ‹$T = U$› ‹*full1 cdcl*$_W$*-bj* T T''› *bj*$'$ *calculation*(*1*) *r-into-rtranclp*)
qed
ultimately show *?case*
using ‹*full1 cdcl*$_W$*-bj* T T''› ‹*full cdcl*$_W$*-cp* T'' U''› **unfolding** ‹$T = U$› **by** *blast*
qed

lemma *cdcl*$_W$*-cp-cdcl*$_W$*-bj-bissimulation*$'$:
assumes
full cdcl$_W$*-cp* T U **and**
cdcl$_W$*-bj*** T T' **and**
cdcl$_W$*-all-struct-inv* T **and**
no-step cdcl$_W$*-bj* T'
shows *full cdcl*$_W$*-cp* T' U
$\vee$ ($\exists$ U'. *full1 cdcl*$_W$*-bj* U U' $\wedge$ ($\forall$ U''. *full cdcl*$_W$*-cp* U' U'' $\longrightarrow$ *full cdcl*$_W$*-cp* T' U''
$\wedge$ *cdcl*$_W$*-s*$'^{**}$ U U''))
using *assms*(*2*,*1*,*3*,*4*)
proof (*induction rule*: *rtranclp-induct*)
case *base*
then show *?case* **by** *blast*
next
case (*step* T' T'') **note** $st =$ *this*(*1*) **and** $bj =$ *this*(*2*) **and** $IH =$ *this*(*3*)[*OF this*(*4*,*5*)] **and**

$full = this(4)$ **and** $inv = this(5)$
have $cdcl_W^{**}\ T\ T''$
by (*metis* (*no-types*, *lifting*) $cdcl_W$-*o.bj local.bj mono-rtranclp*[*of* $cdcl_W$-*bj* $cdcl_W$ $T\ T''$] *other st rtranclp.rtrancl-into-rtrancl*)
then have *inv*-T'': $cdcl_W$-*all-struct-inv* T''
using *inv rtranclp*-$cdcl_W$-*all-struct-inv-inv* **by** *blast*
have $cdcl_W$-$bj^{++}\ T\ T''$
using *local.bj st* **by** *auto*
have *full1* $cdcl_W$-*bj* $T\ T''$
by (*metis* ‹$cdcl_W$-$bj^{++}\ T\ T''$› *full1-def step.prems*(*3*))
then have $T = U$
proof −
obtain Z **where** $cdcl_W$-*bj* $T\ Z$
by (*meson tranclpD* ‹$cdcl_W$-$bj^{++}\ T\ T''$›)
{ **assume** $cdcl_W$-$cp^{++}\ T\ U$
then obtain Z' **where** $cdcl_W$-*cp* $T\ Z'$
by (*meson tranclpD*)
then have *False*
using ‹$cdcl_W$-*bj* $T\ Z$› **by** (*fastforce simp*: $cdcl_W$-*bj.simps* $cdcl_W$-*cp.simps*)
}
then show *?thesis*
using *full* **unfolding** *full-def rtranclp-unfold* **by** *blast*
qed
{ **fix** U''
assume *full* $cdcl_W$-*cp* $T''\ U''$
moreover then have $cdcl_W$-$stgy^{**}\ U\ U''$
by (*metis* ‹$T = U$› ‹$cdcl_W$-$bj^{++}\ T\ T''$› *rtranclp*-$cdcl_W$-*bj-full1-cdclp*-$cdcl_W$-*stgy rtranclp-unfold*)
moreover have $cdcl_W$-$s'^{**}\ U\ U''$
proof −
obtain ss :: $'st \Rightarrow 'st$ **where**
f1: $\forall x2.\ (\exists v3.$ $cdcl_W$-*cp* $x2\ v3) =$ $cdcl_W$-*cp* $x2\ (ss\ x2)$
by *moura*
have $\neg$ $cdcl_W$-*cp* $U\ (ss\ U)$
by (*meson assms*(*1*) *full-def*)
then show *?thesis*
using *f1* **by** (*metis* (*no-types*) ‹$T = U$› ‹*full1* $cdcl_W$-*bj* $T\ T''$› *bj'* *calculation*(*1*) *r-into-rtranclp*)
qed
ultimately have *full1* $cdcl_W$-*bj* $U\ T''$ **and** $cdcl_W$-$s'^{**}\ T''\ U''$
using ‹*full1* $cdcl_W$-*bj* $T\ T''$› ‹*full* $cdcl_W$-*cp* $T''\ U''$› **unfolding** ‹$T = U$›
apply *blast*
by (*metis* ‹*full* $cdcl_W$-*cp* $T''\ U''$› $cdcl_W$-*s'.simps full-unfold rtranclp.simps*)
}
then show *?case*
using ‹*full1* $cdcl_W$-*bj* $T\ T''$› *full bj'* **unfolding** ‹$T = U$› *full-def* **by** (*metis r-into-rtranclp*)
qed

lemma $cdcl_W$-*stgy*-$cdcl_W$-*s'-connected*:
assumes $cdcl_W$-*stgy* $S\ U$ **and** $cdcl_W$-*all-struct-inv* S
shows $cdcl_W$-*s'* $S\ U$
$\vee$ ($\exists U'$. *full1* $cdcl_W$-*bj* $U\ U' \wedge (\forall U''$. *full* $cdcl_W$-*cp* $U'\ U'' \longrightarrow$ $cdcl_W$-*s'* $S\ U''$))
using *assms*
proof (*induction rule*: $cdcl_W$-*stgy.induct*)
case (*conflict'* T)
then have $cdcl_W$-*s'* $S\ T$

using *cdcl*$_W$*-s'.conflict'* **by** *blast*
then show *?case*
by *blast*
next
case (*other' T U*) **note** *o* = *this*(*1*) **and** *n-s* = *this*(*2*) **and** *full* = *this*(*3*) **and** *inv* = *this*(*4*)
show *?case*
using *o*
proof *cases*
case *decide*
then show *?thesis* **using** *cdcl*$_W$*-s'.simps full n-s* **by** *blast*
next
case *bj*
have *inv-T*: *cdcl*$_W$*-all-struct-inv T*
using *cdcl*$_W$*-all-struct-inv-inv o other other'.prems* **by** *blast*
consider
(*cp*) *full cdcl*$_W$*-cp T U* **and** *no-step cdcl*$_W$*-bj T*
| (*fbj*) *T'* **where** *full1 cdcl*$_W$*-bj T T'*
apply (*cases no-step cdcl*$_W$*-bj T*)
using *full* **apply** *blast*
using *cdcl*$_W$*-bj-exists-normal-form*[*of T*] *inv-T* **unfolding** *cdcl*$_W$*-all-struct-inv-def*
by (*metis full-unfold*)
then show *?thesis*
proof *cases*
case *cp*
then show *?thesis*
proof −
obtain *ss* :: *'st* ⇒ *'st* **where**
f1: ∀ *s sa sb*. (¬ *full1 cdcl*$_W$*-bj s sa* ∨ *cdcl*$_W$*-cp s* (*ss s*) ∨ ¬ *full cdcl*$_W$*-cp sa sb*)
∨ *cdcl*$_W$*-s' s sb*
using *bj'* **by** *moura*
have *full1 cdcl*$_W$*-bj S T*
by (*simp add*: *cp*(*2*) *full1-def local.bj tranclp.r-into-trancl*)
then show *?thesis*
using *f1 full n-s* **by** *blast*
qed
next
case (*fbj U'*)
then have *full1 cdcl*$_W$*-bj S U'*
using *bj* **unfolding** *full1-def* **by** *auto*
moreover have *no-step cdcl*$_W$*-cp S*
using *n-s* **by** *blast*
moreover have *T* = *U*
using *full fbj* **unfolding** *full1-def full-def rtranclp-unfold*
by (*force dest*!: *tranclpD simp*:*cdcl*$_W$*-bj.simps*)
ultimately show *?thesis* **using** *cdcl*$_W$*-s'.bj'*[*of S U'*] **using** *fbj* **by** *blast*
qed
qed
qed

lemma *cdcl*$_W$*-stgy-cdcl*$_W$*-s'-connected'*:
assumes *cdcl*$_W$*-stgy S U* **and** *cdcl*$_W$*-all-struct-inv S*
shows *cdcl*$_W$*-s' S U*
∨ (∃ *U' U''*. *cdcl*$_W$*-s' S U''* ∧ *full1 cdcl*$_W$*-bj U U'* ∧ *full cdcl*$_W$*-cp U' U''*)
using *assms*
proof (*induction rule*: *cdcl*$_W$*-stgy.induct*)

case (*conflict'* T)
then have *cdcl*$_W$*-s'* S T
using *cdcl*$_W$*-s'.conflict'* **by** *blast*
then show *?case*
by *blast*
next
case (*other'* T U) **note** o = *this*(*1*) **and** *n-s* = *this*(*2*) **and** *full* = *this*(*3*) **and** *inv* = *this*(*4*)
show *?case*
using o
proof *cases*
case *decide*
then show *?thesis* **using** *cdcl*$_W$*-s'.simps full n-s* **by** *blast*
next
case *bj*
have *cdcl*$_W$*-all-struct-inv* T
using *cdcl*$_W$*-all-struct-inv-inv o other other'.prems* **by** *blast*
then obtain T' **where** T': *full cdcl*$_W$*-bj* T T'
using *cdcl*$_W$*-bj-exists-normal-form* **unfolding** *full-def cdcl*$_W$*-all-struct-inv-def* **by** *metis*
then have *full cdcl*$_W$*-bj* S T'
proof −
have *f1*: *cdcl*$_W$*-bj*** T T' $\wedge$ *no-step cdcl*$_W$*-bj* T'
by (*metis* (*no-types*) T' *full-def*)
then have *cdcl*$_W$*-bj*** S T'
by (*meson converse-rtranclp-into-rtranclp local.bj*)
then show *?thesis*
using *f1* **by** (*simp add*: *full-def*)
qed
have *cdcl*$_W$*-bj*** T T'
using T' **unfolding** *full-def* **by** *simp*
have *cdcl*$_W$*-all-struct-inv* T
using *cdcl*$_W$*-all-struct-inv-inv o other other'.prems* **by** *blast*
then consider
($T'U$) *full cdcl*$_W$*-cp* T' U
| (U) U' U'' **where**
full cdcl$_W$*-cp* T' U'' **and**
full1 cdcl$_W$*-bj* U U' **and**
full cdcl$_W$*-cp* U' U'' **and**
cdcl$_W$*-s'*** U U''
using *cdcl*$_W$*-cp-cdcl*$_W$*-bj-bissimulation*[*OF full* ‹*cdcl*$_W$*-bj*** T T'›] T' **unfolding** *full-def*
by *blast*
then show *?thesis* **by** (*metis* T' *cdcl*$_W$*-s'.simps full-fullI local.bj n-s*)
qed
qed

lemma *cdcl*$_W$*-stgy-cdcl*$_W$*-s'-no-step*:
assumes *cdcl*$_W$*-stgy* S U **and** *cdcl*$_W$*-all-struct-inv* S **and** *no-step cdcl*$_W$*-bj* U
shows *cdcl*$_W$*-s'* S U
using *cdcl*$_W$*-stgy-cdcl*$_W$*-s'-connected*[*OF assms*(*1*,*2*)] *assms*(*3*)
by (*metis* (*no-types*, *lifting*) *full1-def tranclpD*)

lemma *rtranclp-cdcl*$_W$*-stgy-connected-to-rtranclp-cdcl*$_W$*-s'*:
assumes *cdcl*$_W$*-stgy*** S U **and** *inv*: *cdcl*$_W$*-M-level-inv* S
shows *cdcl*$_W$*-s'*** S U $\vee$ ($\exists$ T. *cdcl*$_W$*-s'*** S T $\wedge$ *cdcl*$_W$*-bj*$^{++}$ T U $\wedge$ *conflicting* $U \neq$ *None*)
using *assms*(*1*)
proof *induction*

case *base*
then show *?case* **by** *simp*
next
case (*step T V*) **note** $st = this(1)$ **and** $o = this(2)$ **and** $IH = this(3)$
from *o* **show** *?case*
proof *cases*
case *conflict'*
then have *f2*: $cdcl_W$-*s' T V*
using $cdcl_W$-*s'.conflict'* **by** *blast*
obtain *ss* :: *'st* **where**
f3: $S = T \vee cdcl_W\text{-}stgy^{**}\ S\ ss \wedge cdcl_W\text{-}stgy\ ss\ T$
by (*metis* (*full-types*) *rtranclp.simps st*)
obtain *ssa* :: *'st* **where**
$cdcl_W$-*cp T ssa*
using *conflict'* **by** (*metis* (*no-types*) *full1-def tranclpD*)
then have $S = T$
using *f3* **by** (*metis* (*no-types*) $cdcl_W$-*stgy.simps full-def full1-def*)
then show *?thesis*
using *f2* **by** *blast*
next
case (*other' U*) **note** $o = this(1)$ **and** $n\text{-}s = this(2)$ **and** $full = this(3)$
then show *?thesis*
using *o*
proof (*cases rule*: $cdcl_W$-*o-rule-cases*)
case *decide*
then have $cdcl_W\text{-}s'^{**}\ S\ T$
using *IH* **by** *auto*
then show *?thesis*
by (*meson decide decide' full n-s rtranclp.rtrancl-into-rtrancl*)
next
case *backtrack*
consider
(*s'*) $cdcl_W\text{-}s'^{**}\ S\ T$
| (*bj*) *S'* **where** $cdcl_W\text{-}s'^{**}\ S\ S'$ **and** $cdcl_W\text{-}bj^{++}\ S'\ T$ **and** *conflicting* $T \neq None$
using *IH* **by** *blast*
then show *?thesis*
proof *cases*
case *s'*
moreover
have $cdcl_W$-*M-level-inv T*
using *inv local.step*(*1*) *rtranclp*-$cdcl_W$-*stgy-consistent-inv* **by** *auto*
then have *full1* $cdcl_W$-*bj T U*
using *backtrack-is-full1*-$cdcl_W$-*bj backtrack* **by** *blast*
then have $cdcl_W$-*s' T V*
using *full bj' n-s* **by** *blast*
ultimately show *?thesis* **by** *auto*
next
case (*bj S'*) **note** $S\text{-}S' = this(1)$ **and** $bj\text{-}T = this(2)$
have *no-step* $cdcl_W$-*cp S'*
using *bj-T* **by** (*fastforce simp*: $cdcl_W$-*cp.simps* $cdcl_W$-*bj.simps dest*!: *tranclpD*)
moreover
have $cdcl_W$-*M-level-inv T*
using *inv local.step*(*1*) *rtranclp*-$cdcl_W$-*stgy-consistent-inv* **by** *auto*
then have *full1* $cdcl_W$-*bj T U*
using *backtrack-is-full1*-$cdcl_W$-*bj backtrack* **by** *blast*

then have *full1 cdcl*$_W$*-bj* S' U
using *bj-T* **unfolding** *full1-def* **by** *fastforce*
ultimately have *cdcl*$_W$*-s'* S' V **using** *full* **by** (*simp add*: *bj'*)
then show *?thesis* **using** *S-S'* **by** *auto*
qed
next
case *skip*
then have [*simp*]: $U = V$
using *full converse-rtranclpE* **unfolding** *full-def* **by** *fastforce*

consider
(*s'*) *cdcl*$_W$*-s'*** S T
| (*bj*) S' **where** *cdcl*$_W$*-s'*** S S' **and** *cdcl*$_W$*-bj*$^{++}$ S' T **and** *conflicting* $T \neq$ *None*
using *IH* **by** *blast*
then show *?thesis*
proof *cases*
case *s'*
have *cdcl*$_W$*-bj*$^{++}$ T V
using *skip* **by** *force*
moreover have *conflicting* $V \neq$ *None*
using *skip* **by** *auto*
ultimately show *?thesis* **using** *s'* **by** *auto*
next
case (*bj* S') **note** *S-S'* = *this*(*1*) **and** *bj-T* = *this*(*2*)
have *cdcl*$_W$*-bj*$^{++}$ S' V
using *skip bj-T* **by** (*metis* ‹$U = V$› *cdcl*$_W$*-bj.skip tranclp.simps*)

moreover have *conflicting* $V \neq$ *None*
using *skip* **by** *auto*
ultimately show *?thesis* **using** *S-S'* **by** *auto*
qed
next
case *resolve*
then have [*simp*]: $U = V$
using *full converse-rtranclpE* **unfolding** *full-def* **by** *fastforce*
consider
(*s'*) *cdcl*$_W$*-s'*** S T
| (*bj*) S' **where** *cdcl*$_W$*-s'*** S S' **and** *cdcl*$_W$*-bj*$^{++}$ S' T **and** *conflicting* $T \neq$ *None*
using *IH* **by** *blast*
then show *?thesis*
proof *cases*
case *s'*
have *cdcl*$_W$*-bj*$^{++}$ T V
using *resolve* **by** *force*
moreover have *conflicting* $V \neq$ *None*
using *resolve* **by** *auto*
ultimately show *?thesis* **using** *s'* **by** *auto*
next
case (*bj* S') **note** *S-S'* = *this*(*1*) **and** *bj-T* = *this*(*2*)
have *cdcl*$_W$*-bj*$^{++}$ S' V
using *resolve bj-T* **by** (*metis* ‹$U = V$› *cdcl*$_W$*-bj.resolve tranclp.simps*)
moreover have *conflicting* $V \neq$ *None*
using *resolve* **by** *auto*
ultimately show *?thesis* **using** *S-S'* **by** *auto*
qed

qed
qed
qed

lemma *n-step-cdcl$_W$-stgy-iff-no-step-cdcl$_W$-cl-cdcl$_W$-o*:
assumes *inv*: *cdcl$_W$-all-struct-inv S*
shows *no-step cdcl$_W$-s' S* $\longleftrightarrow$ *no-step cdcl$_W$-cp S* $\wedge$ *no-step cdcl$_W$-o S* (**is** *?S' S* $\longleftrightarrow$ *?C S* $\wedge$ *?O S*)
proof
assume *?C S* $\wedge$ *?O S*
then show *?S' S*
by (*auto simp*: *cdcl$_W$-s'.simps full1-def tranclp-unfold-begin*)
next
assume *n-s*: *?S' S*
have *?C S*
proof (*rule ccontr*)
assume $\neg$ *?thesis*
then obtain *S'* **where** *cdcl$_W$-cp S S'*
by *auto*
then obtain *T* **where** *full1 cdcl$_W$-cp S T*
using *cdcl$_W$-cp-normalized-element-all-inv inv* **by** (*metis* (*no-types*, *lifting*) *full-unfold*)
then show *False* **using** *n-s cdcl$_W$-s'.conflict'* **by** *blast*
qed
moreover have *?O S*
proof (*rule ccontr*)
assume $\neg$ *?thesis*
then obtain *S'* **where** *cdcl$_W$-o S S'*
by *auto*
then obtain *T* **where** *full1 cdcl$_W$-cp S' T*
using *cdcl$_W$-cp-normalized-element-all-inv inv*
by (*meson cdcl$_W$-all-struct-inv-def n-s*
cdcl$_W$-stgy-cdcl$_W$-s'-connected' cdcl$_W$-then-exists-cdcl$_W$-stgy-step)
then show *False* **using** *n-s* **by** (*meson* ‹*cdcl$_W$-o S S'*› *cdcl$_W$-all-struct-inv-def*
cdcl$_W$-stgy-cdcl$_W$-s'-connected' cdcl$_W$-then-exists-cdcl$_W$-stgy-step inv)
qed
ultimately show *?C S* $\wedge$ *?O S* **by** *auto*
qed

lemma *cdcl$_W$-s'-tranclp-cdcl$_W$*:
cdcl$_W$-s' S S' $\Longrightarrow$ *cdcl$_W$$^{++}$ S S'*
proof (*induct rule*: *cdcl$_W$-s'.induct*)
case *conflict'*
then show *?case*
by (*simp add*: *full1-def tranclp-cdcl$_W$-cp-tranclp-cdcl$_W$*)
next
case *decide'*
then show *?case*
using *cdcl$_W$-stgy.simps cdcl$_W$-stgy-tranclp-cdcl$_W$* **by** (*meson cdcl$_W$-o.simps*)
next
case (*bj' Sa S'a S''*) **note** *a2* = *this*(*1*) **and** *a1* = *this*(*2*) **and** *n-s* = *this*(*3*)
obtain *ss* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ (*'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool*) $\Rightarrow$ *'st* **where**
$\forall$ *x0 x1 x2*. ($\exists$ *v3*. *x2 x1 v3* $\wedge$ *x2** v3 x0*) = (*x2 x1* (*ss x0 x1 x2*) $\wedge$ *x2*** (*ss x0 x1 x2*) *x0*)
by *moura*
then have *f3*: $\forall$ *p s sa*. $\neg$ *p^{++} s sa* $\vee$ *p s* (*ss sa s p*) $\wedge$ *p*** (*ss sa s p*) *sa*
by (*metis* (*full-types*) *tranclpD*)
have *cdcl$_W$-bj^{++} Sa S'a* $\wedge$ *no-step cdcl$_W$-bj S'a*

using *a2* **by** (*simp add*: *full1-def*)
then have $cdcl_W$*-bj Sa* (*ss S′a Sa* $cdcl_W$*-bj*) ∧ $cdcl_W$*-bj*** (*ss S′a Sa* $cdcl_W$*-bj*) *S′a*
using *f3* **by** *auto*
then show $cdcl_W{}^{++}$ *Sa S″*
using *a1 n-s* **by** (*meson bj other rtranclp-*$cdcl_W$*-bj-full1-cdclp-*$cdcl_W$*-stgy*
rtranclp-$cdcl_W$*-stgy-rtranclp-*$cdcl_W$ *rtranclp-into-tranclp2*)
qed

lemma *tranclp-*$cdcl_W$*-s′-tranclp-*$cdcl_W$:
$cdcl_W$*-s′*$^{++}$ *S S′* ⟹ $cdcl_W{}^{++}$ *S S′*
apply (*induct rule*: *tranclp.induct*)
using $cdcl_W$*-s′-tranclp-*$cdcl_W$ **apply** *blast*
by (*meson* $cdcl_W$*-s′-tranclp-*$cdcl_W$ *tranclp-trans*)

lemma *rtranclp-*$cdcl_W$*-s′-rtranclp-*$cdcl_W$:
$cdcl_W$*-s′*** *S S′* ⟹ $cdcl_W{}^{**}$ *S S′*
using *rtranclp-unfold*[*of* $cdcl_W$*-s′ S S′*] *tranclp-*$cdcl_W$*-s′-tranclp-*$cdcl_W$[*of S S′*] **by** *auto*

lemma *full-*$cdcl_W$*-stgy-iff-full-*$cdcl_W$*-s′*:
assumes *inv*: $cdcl_W$*-all-struct-inv S*
shows *full* $cdcl_W$*-stgy S T* ⟷ *full* $cdcl_W$*-s′ S T* (**is** *?S* ⟷ *?S′*)
proof
assume *?S′*
then have $cdcl_W{}^{**}$ *S T*
using *rtranclp-*$cdcl_W$*-s′-rtranclp-*$cdcl_W$[*of S T*] **unfolding** *full-def* **by** *blast*
then have *inv′*: $cdcl_W$*-all-struct-inv T*
using *rtranclp-*$cdcl_W$*-all-struct-inv-inv inv* **by** *blast*
have $cdcl_W$*-stgy*** *S T*
using ‹*?S′*› **unfolding** *full-def*
using $cdcl_W$*-s′-is-rtranclp-*$cdcl_W$*-stgy rtranclp-mono*[*of* $cdcl_W$*-s′* $cdcl_W$*-stgy***] **by** *auto*
then show *?S*
using ‹*?S′*› *inv′* $cdcl_W$*-stgy-*$cdcl_W$*-s′-connected′* **unfolding** *full-def* **by** *blast*
next
assume *?S*
then have *inv-T*:$cdcl_W$*-all-struct-inv T*
by (*metis assms full-def rtranclp-*$cdcl_W$*-all-struct-inv-inv rtranclp-*$cdcl_W$*-stgy-rtranclp-*$cdcl_W$)

consider
(*s′*) $cdcl_W$*-s′*** *S T*
| (*st*) *S′* **where** $cdcl_W$*-s′*** *S S′* **and** $cdcl_W$*-bj*$^{++}$ *S′ T* **and** *conflicting T* ≠ *None*
using *rtranclp-*$cdcl_W$*-stgy-connected-to-rtranclp-*$cdcl_W$*-s′*[*of S T*] *inv* ‹*?S*›
unfolding *full-def* $cdcl_W$*-all-struct-inv-def*
by *blast*
then show *?S′*
proof *cases*
case *s′*
then show *?thesis*
by (*metis* ‹*full* $cdcl_W$*-stgy S T*› *inv-T* $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-s′.simps*
$cdcl_W$*-stgy.conflict′* $cdcl_W$*-then-exists-*$cdcl_W$*-stgy-step full-def*
n-step-$cdcl_W$*-stgy-iff-no-step-*$cdcl_W$*-cl-*$cdcl_W$*-o*)
next
case (*st S′*)
have *full* $cdcl_W$*-cp T T*
using *option-full-*$cdcl_W$*-cp st*(*3*) **by** *blast*
moreover

        have *n-s*: *no-step* $cdcl_W$*-bj* *T*
          by (*metis* ‹*full* $cdcl_W$*-stgy* *S* *T*› *bj* *inv-T* $cdcl_W$*-all-struct-inv-def*
            $cdcl_W$*-then-exists-*$cdcl_W$*-stgy-step* *full-def*)
        then have *full1* $cdcl_W$*-bj* S' *T*
          using *st*(*2*) **unfolding** *full1-def* **by** *blast*
      moreover have *no-step* $cdcl_W$*-cp* S'
        using *st*(*2*) **by** (*fastforce* *dest*!: *tranclpD* *simp*: $cdcl_W$*-cp.simps* $cdcl_W$*-bj.simps*)
      ultimately have $cdcl_W$*-s'* S' *T*
        using $cdcl_W$*-s'.bj'*[*of* S' *T* *T*] **by** *blast*
      then have $cdcl_W$*-s'*** *S* *T*
        using *st*(*1*) **by** *auto*
      moreover have *no-step* $cdcl_W$*-s'* *T*
        using *inv-T* **by** (*metis* ‹*full* $cdcl_W$*-cp* *T* *T*› ‹*full* $cdcl_W$*-stgy* *S* *T*› $cdcl_W$*-all-struct-inv-def*
          $cdcl_W$*-then-exists-*$cdcl_W$*-stgy-step* *full-def* *n-step-*$cdcl_W$*-stgy-iff-no-step-*$cdcl_W$*-cl-*$cdcl_W$*-o*)
      ultimately show *?thesis*
        unfolding *full-def* **by** *blast*
    qed
qed

lemma *conflict-step-*$cdcl_W$*-stgy-step*:
  assumes
    conflict *S* *T*
    $cdcl_W$*-all-struct-inv* *S*
  shows $\exists$ *T*. $cdcl_W$*-stgy* *S* *T*
proof −
  obtain *U* **where** *full* $cdcl_W$*-cp* *S* *U*
    using $cdcl_W$*-cp-normalized-element-all-inv* *assms* **by** *blast*
  then have *full1* $cdcl_W$*-cp* *S* *U*
    by (*metis* $cdcl_W$*-cp.conflict'* *assms*(*1*) *full-unfold*)
  then show *?thesis* **using** $cdcl_W$*-stgy.conflict'* **by** *blast*
qed

lemma *decide-step-*$cdcl_W$*-stgy-step*:
  assumes
    decide *S* *T*
    $cdcl_W$*-all-struct-inv* *S*
  shows $\exists$ *T*. $cdcl_W$*-stgy* *S* *T*
proof −
  obtain *U* **where** *full* $cdcl_W$*-cp* *T* *U*
    using $cdcl_W$*-cp-normalized-element-all-inv* **by** (*meson* *assms*(*1*) *assms*(*2*) $cdcl_W$*-all-struct-inv-inv*
      $cdcl_W$*-cp-normalized-element-all-inv* *decide* *other*)
  then show *?thesis*
    by (*metis* *assms* $cdcl_W$*-cp-normalized-element-all-inv* $cdcl_W$*-stgy.conflict'* *decide* *full-unfold*
      other')
qed

lemma *rtranclp-*$cdcl_W$*-cp-conflicting-Some*:
  $cdcl_W$*-cp*** *S* *T* $\Longrightarrow$ *conflicting* *S* = *Some* *D* $\Longrightarrow$ *S* = *T*
  using *rtranclpD* *tranclpD* **by** *fastforce*

inductive $cdcl_W$*-merge-cp* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **where**
conflict'[*intro*]: *conflict* *S* *T* $\Longrightarrow$ *full* $cdcl_W$*-bj* *T* *U* $\Longrightarrow$ $cdcl_W$*-merge-cp* *S* *U* |
propagate'[*intro*]: *propagate*$^{++}$ *S* S' $\Longrightarrow$ $cdcl_W$*-merge-cp* *S* S'

lemma $cdcl_W$*-merge-restart-cases*[*consumes* *1*, *case-names* *conflict* *propagate*]:

assumes
cdcl$_W$*-merge-cp S U* **and**
$\bigwedge T$. *conflict S T* $\Longrightarrow$ *full cdcl*$_W$*-bj T U* $\Longrightarrow$ *P* **and**
propagate^{++} *S U* $\Longrightarrow$ *P*
shows *P*
using *assms* **unfolding** *cdcl*$_W$*-merge-cp.simps* **by** *auto*

lemma *cdcl*$_W$*-merge-cp-tranclp-cdcl*$_W$*-merge*:
cdcl$_W$*-merge-cp S T* $\Longrightarrow$ *cdcl*$_W$*-merge*$^{++}$ *S T*
apply (*induction rule*: *cdcl*$_W$*-merge-cp.induct*)
using *cdcl*$_W$*-merge.simps* **apply** *auto*[*1*]
using *tranclp-mono*[*of propagate cdcl*$_W$*-merge*] *fw-propagate* **by** *blast*

lemma *rtranclp-cdcl*$_W$*-merge-cp-rtranclp-cdcl*$_W$:
cdcl$_W$*-merge-cp*** *S T* $\Longrightarrow$ *cdcl*$_W$** *S T*
apply (*induction rule*: *rtranclp-induct*)
apply *simp*
unfolding *cdcl*$_W$*-merge-cp.simps* **by** (*meson cdcl*$_W$*-merge-restart-cdcl*$_W$ *fw-r-conflict*
rtranclp-propagate-is-rtranclp-cdcl$_W$ *rtranclp-trans tranclp-into-rtranclp*)

lemma *full1-cdcl*$_W$*-bj-no-step-cdcl*$_W$*-bj*:
full1 cdcl$_W$*-bj S T* $\Longrightarrow$ *no-step cdcl*$_W$*-cp S*
by (*metis rtranclp-unfold cdcl*$_W$*-cp-conflicting-not-empty option.exhaust full1-def*
rtranclp-cdcl$_W$*-merge-restart-no-step-cdcl*$_W$*-bj tranclpD*)

inductive *cdcl*$_W$*-s'-without-decide* **where**
conflict'-without-decide[*intro*]: *full1 cdcl*$_W$*-cp S S'* $\Longrightarrow$ *cdcl*$_W$*-s'-without-decide S S'* |
bj'-without-decide[*intro*]: *full1 cdcl*$_W$*-bj S S'* $\Longrightarrow$ *no-step cdcl*$_W$*-cp S* $\Longrightarrow$ *full cdcl*$_W$*-cp S' S''*
$\Longrightarrow$ *cdcl*$_W$*-s'-without-decide S S''*

lemma *rtranclp-cdcl*$_W$*-s'-without-decide-rtranclp-cdcl*$_W$:
cdcl$_W$*-s'-without-decide*** *S T* $\Longrightarrow$ *cdcl*$_W$** *S T*
apply (*induction rule*: *rtranclp-induct*)
apply *simp*
by (*meson cdcl*$_W$*-s'.simps cdcl*$_W$*-s'-tranclp-cdcl*$_W$ *cdcl*$_W$*-s'-without-decide.simps*
rtranclp-tranclp-tranclp tranclp-into-rtranclp)

lemma *rtranclp-cdcl*$_W$*-s'-without-decide-rtranclp-cdcl*$_W$*-s'*:
cdcl$_W$*-s'-without-decide*** *S T* $\Longrightarrow$ *cdcl*$_W$*-s'*** *S T*
proof (*induction rule*: *rtranclp-induct*)
case *base*
then show *?case* **by** *simp*
next
case (*step y z*) **note** *a2* = *this*(*2*) **and** *a1* = *this*(*3*)
have *cdcl*$_W$*-s' y z*
using *a2* **by** (*metis* (*no-types*) *bj' cdcl*$_W$*-s'.conflict' cdcl*$_W$*-s'-without-decide.cases*)
then show *cdcl*$_W$*-s'*** *S z*
using *a1* **by** (*meson r-into-rtranclp rtranclp-trans*)
qed

lemma *rtranclp-cdcl*$_W$*-merge-cp-is-rtranclp-cdcl*$_W$*-s'-without-decide*:
assumes
cdcl$_W$*-merge-cp*** *S V*
conflicting S = *None*
shows

```
    (cdcl_W-s'-without-decide** S V)
    ∨ (∃ T. cdcl_W-s'-without-decide** S T ∧ propagate++ T V)
    ∨ (∃ T U. cdcl_W-s'-without-decide** S T ∧ full1 cdcl_W-bj T U ∧ propagate** U V)
  using assms
proof (induction rule: rtranclp-induct)
  case base
  then show ?case by simp
next
  case (step U V) note st = this(1) and cp = this(2) and IH = this(3)[OF this(4)]
  from cp show ?case
    proof (cases rule: cdcl_W-merge-restart-cases)
      case propagate
      then show ?thesis using IH by (meson rtranclp-tranclp-tranclp tranclp-into-rtranclp)
    next
      case (conflict U') note confl = this(1) and bj = this(2)
      have full1-U-U': full1 cdcl_W-cp U U'
        by (simp add: conflict-is-full1-cdcl_W-cp local.conflict(1))
      consider
          (s') cdcl_W-s'-without-decide** S U
        | (propa) T' where cdcl_W-s'-without-decide** S T' and propagate++ T' U
        | (bj-prop) T' T'' where
            cdcl_W-s'-without-decide** S T'  and
            full1 cdcl_W-bj T' T'' and
            propagate** T'' U
        using IH by blast
      then show ?thesis
        proof cases
          case s'
          have cdcl_W-s'-without-decide U U'
           using full1-U-U' conflict'-without-decide by blast
          then have cdcl_W-s'-without-decide** S U'
            using ‹cdcl_W-s'-without-decide** S U› by auto
          moreover have U' = V ∨ full1 cdcl_W-bj U' V
            using bj by (meson full-unfold)
          ultimately show ?thesis by blast
        next
          case propa note s' = this(1) and T'-U = this(2)
          have full1 cdcl_W-cp T' U'
            using rtranclp-mono[of propagate cdcl_W-cp] T'-U cdcl_W-cp.propagate' full1-U-U'
            rtranclp-full1I[of cdcl_W-cp T'] by (metis (full-types) predicate2D predicate2I
              tranclp-into-rtranclp)
          have cdcl_W-s'-without-decide** S U'
            using ‹full1 cdcl_W-cp T' U'› conflict'-without-decide s' by force
          have full1 cdcl_W-bj U' V ∨ V = U'
            by (metis (lifting) full-unfold local.bj)
          then show ?thesis
            using ‹cdcl_W-s'-without-decide** S U'› by blast
        next
          case bj-prop note s' = this(1) and bj-T' = this(2) and T''-U = this(3)
          have no-step cdcl_W-cp T'
            using bj-T' full1-cdcl_W-bj-no-step-cdcl_W-bj by blast
          moreover have full1 cdcl_W-cp T'' U'
            using rtranclp-mono[of propagate cdcl_W-cp] T''-U cdcl_W-cp.propagate' full1-U-U'
            rtranclp-full1I[of cdcl_W-cp T''] by blast
          ultimately have cdcl_W-s'-without-decide T' U'
```

using *bj′-without-decide*[*of* T' T'' U'] *bj-*T' **by** (*simp add*: *full-unfold*)
then have $cdcl_W$*-s′-without-decide*** S U'
using s' *rtranclp.intros*(*2*)[*of* - S T' U'] **by** *blast*
then show *?thesis*
by (*metis full-unfold local.bj rtranclp.rtrancl-refl*)
qed
qed
qed

lemma *rtranclp-*$cdcl_W$*-s′-without-decide-is-rtranclp-*$cdcl_W$*-merge-cp*:
assumes
$cdcl_W$*-s′-without-decide*** S V **and**
confl: *conflicting* S = *None*
shows
($cdcl_W$*-merge-cp*** S V ∧ *conflicting* V = *None*)
∨ ($cdcl_W$*-merge-cp*** S V ∧ *conflicting* V ≠ *None* ∧ *no-step* $cdcl_W$*-cp* V ∧ *no-step* $cdcl_W$*-bj* V)
∨ (∃ T. $cdcl_W$*-merge-cp*** S T ∧ *conflict* T V)
using *assms*(*1*)
proof (*induction*)
case *base*
then show *?case* **using** *confl* **by** *auto*
next
case (*step* U V) **note** *st* = *this*(*1*) **and** *s* = *this*(*2*) **and** *IH* = *this*(*3*)
from *s* **show** *?case*
proof (*cases rule*: $cdcl_W$*-s′-without-decide.cases*)
case *conflict′-without-decide*
then have *rt*: $cdcl_W$*-cp*$^{++}$ U V **unfolding** *full1-def* **by** *fast*
then have *conflicting* U = *None*
using *tranclp-*$cdcl_W$*-cp-propagate-with-conflict-or-not*[*of* U V]
conflict **by** (*auto dest*!: *tranclpD simp*: *rtranclp-unfold*)
then have $cdcl_W$*-merge-cp*** S U **using** *IH* **by** *auto*
consider
(*propa*) *propagate*$^{++}$ U V
| (*confl′*) *conflict* U V
| (*propa-confl′*) U' **where** *propagate*$^{++}$ U U' *conflict* U' V
using *tranclp-*$cdcl_W$*-cp-propagate-with-conflict-or-not*[*OF rt*] **unfolding** *rtranclp-unfold*
by *fastforce*
then show *?thesis*
proof *cases*
case *propa*
then have $cdcl_W$*-merge-cp* U V
by *auto*
moreover have *conflicting* V = *None*
using *propa* **unfolding** *tranclp-unfold-end* **by** *auto*
ultimately show *?thesis* **using** ‹$cdcl_W$*-merge-cp*** S U› **by** *force*
next
case *confl′*
then show *?thesis* **using** ‹$cdcl_W$*-merge-cp*** S U› **by** *auto*
next
case *propa-confl′* **note** *propa* = *this*(*1*) **and** *confl′* = *this*(*2*)
then have $cdcl_W$*-merge-cp* U U' **by** *auto*
then have $cdcl_W$*-merge-cp*** S U' **using** ‹$cdcl_W$*-merge-cp*** S U› **by** *auto*
then show *?thesis* **using** ‹$cdcl_W$*-merge-cp*** S U› *confl′* **by** *auto*
qed

next
case (bj'-*without-decide* U') **note** *full-bj* = *this*(1) **and** *cp* = *this*(3)
then have *conflicting* $U \neq$ *None*
using *full-bj* **unfolding** *full1-def* **by** (*fastforce dest*!: *tranclpD simp*: $cdcl_W$-*bj.simps*)
with *IH* **obtain** *T* **where**
S-T: $cdcl_W$-*merge-cp*** *S T* **and** *T-U*: *conflict T U*
using *full-bj* **unfolding** *full1-def* **by** (*blast dest*: *tranclpD*)
then have $cdcl_W$-*merge-cp T* U'
using $cdcl_W$-*merge-cp.conflict'*[*of T U* U'] *full-bj* **by** (*simp add*: *full-unfold*)
then have *S*-U': $cdcl_W$-*merge-cp*** *S* U' **using** *S-T* **by** *auto*
consider
(*n-s*) $U' = V$
| (*propa*) *propagate*$^{++}$ U' *V*
| (*confl'*) *conflict* U' *V*
| (*propa-confl'*) U'' **where** *propagate*$^{++}$ U' U'' *conflict* U'' *V*
using *tranclp*-$cdcl_W$-*cp-propagate-with-conflict-or-not cp*
unfolding *rtranclp-unfold full-def* **by** *metis*
then show *?thesis*
proof *cases*
case *propa*
then have $cdcl_W$-*merge-cp* U' *V* **by** *auto*
moreover have *conflicting V* = *None*
using *propa* **unfolding** *tranclp-unfold-end* **by** *auto*
ultimately show *?thesis* **using** *S*-U' **by** *force*
next
case *confl'*
then show *?thesis* **using** *S*-U' **by** *auto*
next
case *propa-confl'* **note** *propa* = *this*(1) **and** *confl* = *this*(2)
have $cdcl_W$-*merge-cp* U' U'' **using** *propa* **by** *auto*
then show *?thesis* **using** *S*-U' *confl* **by** (*meson rtranclp.rtrancl-into-rtrancl*)
next
case *n-s*
then show *?thesis*
using *S*-U' **apply** (*cases conflicting V* = *None*)
using *full-bj* **apply** *simp*
by (*metis cp full-def full-unfold full-bj*)
qed
qed
qed

lemma *no-step*-$cdcl_W$-s'-*no-ste*-$cdcl_W$-*merge-cp*:
assumes
$cdcl_W$-*all-struct-inv S*
conflicting S = *None*
no-step $cdcl_W$-s' *S*
shows *no-step* $cdcl_W$-*merge-cp S*
using *assms* **apply** (*auto simp*: $cdcl_W$-s'.*simps* $cdcl_W$-*merge-cp.simps*)
using *conflict-is-full1*-$cdcl_W$-*cp* **apply** *blast*
using $cdcl_W$-*cp-normalized-element-all-inv* $cdcl_W$-*cp.propagate'* **by** (*metis* $cdcl_W$-*cp.propagate'*
full-unfold tranclpD)

The *no-step decide S* is needed, since $cdcl_W$-*merge-cp* is $cdcl_W$-s' without *decide*.

lemma *conflicting-true-no-step*-$cdcl_W$-*merge-cp-no-step*-s'-*without-decide*:
assumes

confl: *conflicting S = None* **and**
inv: *cdcl*$_W$*-M-level-inv S* **and**
n-s: *no-step cdcl*$_W$*-merge-cp S*
shows *no-step cdcl*$_W$*-s'-without-decide S*
proof (*rule ccontr*)
assume ¬ *no-step cdcl*$_W$*-s'-without-decide S*
then obtain *T* **where**
cdcl$_W$: *cdcl*$_W$*-s'-without-decide S T*
by *auto*
then have *inv-T*: *cdcl*$_W$*-M-level-inv T*
using *rtranclp-cdcl*$_W$*-s'-without-decide-rtranclp-cdcl*$_W$[*of S T*]
rtranclp-cdcl$_W$*-consistent-inv inv* **by** *blast*
from *cdcl*$_W$ **show** *False*
proof *cases*
case *conflict'-without-decide*
have *no-step propagate S*
using *n-s* **by** *blast*
then have *conflict S T*
using *local.conflict' tranclp-cdcl*$_W$*-cp-propagate-with-conflict-or-not*[*of S T*]
unfolding *full1-def* **by** (*metis full1-def local.conflict'-without-decide rtranclp-unfold tranclp-unfold-begin*)
moreover
then obtain *T'* **where** *full cdcl*$_W$*-bj T T'*
using *cdcl*$_W$*-bj-exists-normal-form inv-T* **by** *blast*
ultimately show *False* **using** *cdcl*$_W$*-merge-cp.conflict' n-s* **by** *meson*
next
case (*bj'-without-decide S'*)
then show *?thesis*
using *confl* **unfolding** *full1-def* **by** (*fastforce simp*: *cdcl*$_W$*-bj.simps dest*: *tranclpD*)
qed
qed

lemma *conflicting-true-no-step-s'-without-decide-no-step-cdcl*$_W$*-merge-cp*:
assumes
inv: *cdcl*$_W$*-all-struct-inv S* **and**
n-s: *no-step cdcl*$_W$*-s'-without-decide S*
shows *no-step cdcl*$_W$*-merge-cp S*
proof (*rule ccontr*)
assume ¬ *?thesis*
then obtain *T* **where** *cdcl*$_W$*-merge-cp S T*
by *auto*
then show *False*
proof *cases*
case (*conflict' S'*)
then show *False* **using** *n-s conflict'-without-decide conflict-is-full1-cdcl*$_W$*-cp* **by** *blast*
next
case *propagate'*
moreover
have *cdcl*$_W$*-all-struct-inv T*
using *inv* **by** (*meson local.propagate' rtranclp-cdcl*$_W$*-all-struct-inv-inv rtranclp-propagate-is-rtranclp-cdcl*$_W$ *tranclp-into-rtranclp*)
then obtain *U* **where** *full cdcl*$_W$*-cp T U*
using *cdcl*$_W$*-cp-normalized-element-all-inv* **by** *auto*
ultimately have *full1 cdcl*$_W$*-cp S U*
using *tranclp-full-full1I*[*of cdcl*$_W$*-cp S T U*] *cdcl*$_W$*-cp.propagate'*

tranclp-mono[*of propagate* $cdcl_W$*-cp*] **by** *blast*
then show *False* **using** *conflict′-without-decide n-s* **by** *blast*
qed
qed

lemma *no-step-*$cdcl_W$*-merge-cp-no-step-*$cdcl_W$*-cp*:
no-step $cdcl_W$*-merge-cp S* $\Longrightarrow$ $cdcl_W$*-M-level-inv S* $\Longrightarrow$ *no-step* $cdcl_W$*-cp S*
using $cdcl_W$*-bj-exists-normal-form* $cdcl_W$*-consistent-inv*[*OF* $cdcl_W$*.conflict*, *of S*]
by (*metis* $cdcl_W$*-cp.cases* $cdcl_W$*-merge-cp.simps tranclp.intros*(*1*))

lemma *conflicting-not-true-rtranclp-*$cdcl_W$*-merge-cp-no-step-*$cdcl_W$*-bj*:
assumes
conflicting S = *None* **and**
$cdcl_W$*-merge-cp*** *S T*
shows *no-step* $cdcl_W$*-bj T*
using *assms*(*2*,*1*) **by** (*induction*)
(*fastforce simp*: $cdcl_W$*-merge-cp.simps full-def tranclp-unfold-end* $cdcl_W$*-bj.simps*)+

lemma *conflicting-true-full-*$cdcl_W$*-merge-cp-iff-full-*$cdcl_W$*-s′-without-decode*:
assumes
confl: *conflicting S* = *None* **and**
inv: $cdcl_W$*-all-struct-inv S*
shows
full $cdcl_W$*-merge-cp S V* $\longleftrightarrow$ *full* $cdcl_W$*-s′-without-decide S V* (**is** *?fw* $\longleftrightarrow$ *?s′*)
proof
assume *?fw*
then have *st*: $cdcl_W$*-merge-cp*** *S V* **and** *n-s*: *no-step* $cdcl_W$*-merge-cp V*
unfolding *full-def* **by** *blast*+
have *inv-V*: $cdcl_W$*-all-struct-inv V*
using *rtranclp-*$cdcl_W$*-merge-cp-rtranclp-*$cdcl_W$[*of S V*] ‹*?fw*› **unfolding** *full-def*
by (*simp add*: *inv rtranclp-*$cdcl_W$*-all-struct-inv-inv*)
consider
(*s′*) $cdcl_W$*-s′-without-decide*** *S V*
| (*propa*) *T* **where** $cdcl_W$*-s′-without-decide*** *S T* **and** *propagate*$^{++}$ *T V*
| (*bj*) *T U* **where** $cdcl_W$*-s′-without-decide*** *S T* **and** *full1* $cdcl_W$*-bj T U* **and** *propagate*** *U V*
using *rtranclp-*$cdcl_W$*-merge-cp-is-rtranclp-*$cdcl_W$*-s′-without-decide confl st n-s* **by** *metis*
then have $cdcl_W$*-s′-without-decide*** *S V*
proof *cases*
case *s′*
then show *?thesis* **.**
next
case *propa* **note** *s′* = *this*(*1*) **and** *propa* = *this*(*2*)
have *no-step* $cdcl_W$*-cp V*
using *no-step-*$cdcl_W$*-merge-cp-no-step-*$cdcl_W$*-cp n-s inv-V*
unfolding $cdcl_W$*-all-struct-inv-def* **by** *blast*
then have *full1* $cdcl_W$*-cp T V*
using *propa tranclp-mono*[*of propagate* $cdcl_W$*-cp*] $cdcl_W$*-cp.propagate′* **unfolding** *full1-def*
by *blast*
then have $cdcl_W$*-s′-without-decide T V*
using *conflict′-without-decide* **by** *blast*
then show *?thesis* **using** *s′* **by** *auto*
next
case *bj* **note** *s′* = *this*(*1*) **and** *bj* = *this*(*2*) **and** *propa* = *this*(*3*)
have *no-step* $cdcl_W$*-cp V*
using *no-step-*$cdcl_W$*-merge-cp-no-step-*$cdcl_W$*-cp n-s inv-V*

unfolding $cdcl_W$*-all-struct-inv-def* **by** *blast*
then have *full* $cdcl_W$*-cp U V*
using *propa rtranclp-mono*[*of propagate* $cdcl_W$*-cp*] $cdcl_W$*-cp.propagate'* **unfolding** *full-def* **by** *blast*
moreover have *no-step* $cdcl_W$*-cp T*
using *bj* **unfolding** *full1-def* **by** (*fastforce dest*!: *tranclpD simp*:$cdcl_W$*-bj.simps*)
ultimately have $cdcl_W$*-s'-without-decide T V*
using *bj'-without-decide*[*of T U V*] *bj* **by** *blast*
then show *?thesis* **using** *s'* **by** *auto*
qed
moreover have *no-step* $cdcl_W$*-s'-without-decide V*
proof (*cases conflicting V = None*)
case *False*
{ **fix** *ss* :: *'st*
have *ff1*: $\forall$ *s sa.* $\neg$ $cdcl_W$*-s' s sa* $\vee$ *full1* $cdcl_W$*-cp s sa*
$\vee$ ($\exists$ *sb. decide s sb* $\wedge$ *no-step* $cdcl_W$*-cp s* $\wedge$ *full* $cdcl_W$*-cp sb sa*)
$\vee$ ($\exists$ *sb. full1* $cdcl_W$*-bj s sb* $\wedge$ *no-step* $cdcl_W$*-cp s* $\wedge$ *full* $cdcl_W$*-cp sb sa*)
by (*metis* $cdcl_W$*-s'.cases*)
have *ff2*: ($\forall$ *p s sa.* $\neg$ *full1 p* (*s*::*'st*) *sa* $\vee$ p^{++} *s sa* $\wedge$ *no-step p sa*)
$\wedge$ ($\forall$ *p s sa.* ($\neg$ p^{++} (*s*::*'st*) *sa* $\vee$ ($\exists$ *s. p sa s*)) $\vee$ *full1 p s sa*)
by (*meson full1-def*)
obtain *ssa* :: (*'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool*) $\Rightarrow$ *'st* $\Rightarrow$ *'st* $\Rightarrow$ *'st* **where**
ff3: $\forall$ *p s sa.* $\neg$ p^{++} *s sa* $\vee$ *p s* (*ssa p s sa*) $\wedge$ p^{**} (*ssa p s sa*) *sa*
by (*metis* (*no-types*) *tranclpD*)
then have *a3*: $\neg$ $cdcl_W$*-cp*$^{++}$ *V ss*
using *False* **by** (*metis option-full-*$cdcl_W$*-cp full-def*)
have $\bigwedge$*s.* $\neg$ $cdcl_W$*-bj*$^{++}$ *V s*
using *ff3 False* **by** (*metis confl st*
conflicting-not-true-rtranclp-$cdcl_W$*-merge-cp-no-step-*$cdcl_W$*-bj*)
then have $\neg$ $cdcl_W$*-s'-without-decide V ss*
using *ff1 a3 ff2* **by** (*metis* $cdcl_W$*-s'-without-decide.cases*)
}
then show *?thesis*
by *fastforce*
next
case *True*
then show *?thesis*
using *conflicting-true-no-step-*$cdcl_W$*-merge-cp-no-step-s'-without-decide n-s inv-V*
unfolding $cdcl_W$*-all-struct-inv-def* **by** *blast*
qed
ultimately show *?s'* **unfolding** *full-def* **by** *blast*
next
assume *s'*: *?s'*
then have *st*: $cdcl_W$*-s'-without-decide*** *S V* **and** *n-s*: *no-step* $cdcl_W$*-s'-without-decide V*
unfolding *full-def* **by** *auto*
then have $cdcl_W$** *S V*
using *rtranclp-*$cdcl_W$*-s'-without-decide-rtranclp-*$cdcl_W$ *st* **by** *blast*
then have *inv-V*: $cdcl_W$*-all-struct-inv V* **using** *inv rtranclp-*$cdcl_W$*-all-struct-inv-inv* **by** *blast*
then have *n-s-cp-V*: *no-step* $cdcl_W$*-cp V*
using $cdcl_W$*-cp-normalized-element-all-inv*[*of V*] *full-fullI*[*of* $cdcl_W$*-cp V*] *n-s*
conflict'-without-decide conflicting-true-no-step-s'-without-decide-no-step-$cdcl_W$*-merge-cp*
no-step-$cdcl_W$*-merge-cp-no-step-*$cdcl_W$*-cp*
unfolding $cdcl_W$*-all-struct-inv-def* **by** *presburger*
have *n-s-bj*: *no-step* $cdcl_W$*-bj V*
proof (*rule ccontr*)

assume ¬ *?thesis*
then obtain *W* **where** *W*: *cdcl*$_W$*-bj V W* **by** *blast*
have *cdcl*$_W$*-all-struct-inv W*
using *W cdcl*$_W$*.simps cdcl*$_W$*-all-struct-inv-inv inv-V* **by** *blast*
then obtain *W'* **where** *full1 cdcl*$_W$*-bj V W'*
using *cdcl*$_W$*-bj-exists-normal-form*[*of W*] *full-fullI*[*of cdcl*$_W$*-bj V W*] *W*
unfolding *cdcl*$_W$*-all-struct-inv-def*
by *blast*
moreover
then have *cdcl*$_W$$^{++}$ *V W'*
using *tranclp-mono*[*of cdcl*$_W$*-bj cdcl*$_W$] *cdcl*$_W$*.other cdcl*$_W$*-o.bj* **unfolding** *full1-def* **by** *blast*
then have *cdcl*$_W$*-all-struct-inv W'*
by (*meson inv-V rtranclp-cdcl*$_W$*-all-struct-inv-inv tranclp-into-rtranclp*)
then obtain *X* **where** *full cdcl*$_W$*-cp W' X*
using *cdcl*$_W$*-cp-normalized-element-all-inv* **by** *blast*
ultimately show *False*
using *bj'-without-decide n-s-cp-V n-s* **by** *blast*
qed
from *s'* **consider**
(*cp-true*) *cdcl*$_W$*-merge-cp*** *S V* **and** *conflicting V* = *None*
| (*cp-false*) *cdcl*$_W$*-merge-cp*** *S V* **and** *conflicting V* ≠ *None* **and** *no-step cdcl*$_W$*-cp V* **and**
no-step cdcl$_W$*-bj V*
| (*cp-confl*) *T* **where** *cdcl*$_W$*-merge-cp*** *S T conflict T V*
using *rtranclp-cdcl*$_W$*-s'-without-decide-is-rtranclp-cdcl*$_W$*-merge-cp*[*of S V*] *confl*
unfolding *full-def* **by** *meson*
then have *cdcl*$_W$*-merge-cp*** *S V*
proof *cases*
case *cp-confl* **note** *S-T* = *this*(*1*) **and** *conf-V* = *this*(*2*)
have *full cdcl*$_W$*-bj V V*
using *conf-V n-s-bj* **unfolding** *full-def* **by** *fast*
then have *cdcl*$_W$*-merge-cp T V*
using *cdcl*$_W$*-merge-cp.conflict' conf-V* **by** *auto*
then show *?thesis* **using** *S-T* **by** *auto*
qed *fast+*
moreover
then have *cdcl*$_W$** *S V* **using** *rtranclp-cdcl*$_W$*-merge-cp-rtranclp-cdcl*$_W$ **by** *blast*
then have *cdcl*$_W$*-all-struct-inv V*
using *inv rtranclp-cdcl*$_W$*-all-struct-inv-inv* **by** *blast*
then have *no-step cdcl*$_W$*-merge-cp V*
using *conflicting-true-no-step-s'-without-decide-no-step-cdcl*$_W$*-merge-cp s'*
unfolding *full-def* **by** *blast*
ultimately show *?fw* **unfolding** *full-def* **by** *auto*
qed

lemma *conflicting-true-full1-cdcl*$_W$*-merge-cp-iff-full1-cdcl*$_W$*-s'-without-decode*:
assumes
confl: *conflicting S* = *None* **and**
inv: *cdcl*$_W$*-all-struct-inv S*
shows
full1 cdcl$_W$*-merge-cp S V* ⟷ *full1 cdcl*$_W$*-s'-without-decide S V*
proof −
have *full cdcl*$_W$*-merge-cp S V* = *full cdcl*$_W$*-s'-without-decide S V*
using *confl conflicting-true-full-cdcl*$_W$*-merge-cp-iff-full-cdcl*$_W$*-s'-without-decode inv*
by *blast*
then show *?thesis* **unfolding** *full-unfold full1-def*

by (*metis* (*mono-tags*) *tranclp-unfold-begin*)
qed

lemma *conflicting-true-full1-cdcl*$_W$*-merge-cp-imp-full1-cdcl*$_W$*-s'-without-decode*:
assumes
fw: *full1 cdcl*$_W$*-merge-cp S V* **and**
inv: *cdcl*$_W$*-all-struct-inv S*
shows
full1 cdcl$_W$*-s'-without-decide S V*
proof −
have *conflicting S = None*
using *fw* **unfolding** *full1-def* **by** (*auto dest*!: *tranclpD simp*: *cdcl*$_W$*-merge-cp.simps*)
then show *?thesis*
using *conflicting-true-full1-cdcl*$_W$*-merge-cp-iff-full1-cdcl*$_W$*-s'-without-decode fw inv* **by** *blast*
qed

inductive *cdcl*$_W$*-merge-stgy* **where**
fw-s-cp[*intro*]: *full1 cdcl*$_W$*-merge-cp S T* $\Longrightarrow$ *cdcl*$_W$*-merge-stgy S T* |
fw-s-decide[*intro*]: *decide S T* $\Longrightarrow$ *no-step cdcl*$_W$*-merge-cp S* $\Longrightarrow$ *full cdcl*$_W$*-merge-cp T U*
$\Longrightarrow$ *cdcl*$_W$*-merge-stgy S U*

lemma *cdcl*$_W$*-merge-stgy-tranclp-cdcl*$_W$*-merge*:
assumes *fw*: *cdcl*$_W$*-merge-stgy S T*
shows *cdcl*$_W$*-merge*$^{++}$ *S T*
proof −
{ **fix** *S T*
assume *full1 cdcl*$_W$*-merge-cp S T*
then have *cdcl*$_W$*-merge*$^{++}$ *S T*
using *tranclp-mono*[*of cdcl*$_W$*-merge-cp cdcl*$_W$*-merge*$^{++}$] *cdcl*$_W$*-merge-cp-tranclp-cdcl*$_W$*-merge*
unfolding *full1-def*
by *auto*
} **note** *full1-cdcl*$_W$*-merge-cp-cdcl*$_W$*-merge = this*
show *?thesis*
using *fw*
apply (*induction rule*: *cdcl*$_W$*-merge-stgy.induct*)
using *full1-cdcl*$_W$*-merge-cp-cdcl*$_W$*-merge* **apply** *simp*
unfolding *full-unfold* **by** (*auto dest*!: *full1-cdcl*$_W$*-merge-cp-cdcl*$_W$*-merge fw-decide*)
qed

lemma *rtranclp-cdcl*$_W$*-merge-stgy-rtranclp-cdcl*$_W$*-merge*:
assumes *fw*: *cdcl*$_W$*-merge-stgy*** *S T*
shows *cdcl*$_W$*-merge*** *S T*
using *fw cdcl*$_W$*-merge-stgy-tranclp-cdcl*$_W$*-merge rtranclp-mono*[*of cdcl*$_W$*-merge-stgy cdcl*$_W$*-merge*$^{++}$]
unfolding *tranclp-rtranclp-rtranclp* **by** *blast*

lemma *cdcl*$_W$*-merge-stgy-rtranclp-cdcl*$_W$:
cdcl$_W$*-merge-stgy S T* $\Longrightarrow$ *cdcl*$_W$** *S T*
apply (*induction rule*: *cdcl*$_W$*-merge-stgy.induct*)
using *rtranclp-cdcl*$_W$*-merge-cp-rtranclp-cdcl*$_W$ **unfolding** *full1-def*
apply (*simp add*: *tranclp-into-rtranclp*)
using *rtranclp-cdcl*$_W$*-merge-cp-rtranclp-cdcl*$_W$ *cdcl*$_W$*-o.decide cdcl*$_W$*.other* **unfolding** *full-def*
by (*meson r-into-rtranclp rtranclp-trans*)

lemma *rtranclp-cdcl*$_W$*-merge-stgy-rtranclp-cdcl*$_W$:
cdcl$_W$*-merge-stgy*** *S T* $\Longrightarrow$ *cdcl*$_W$** *S T*

using *rtranclp-mono*[*of* $cdcl_W$*-merge-stgy* $cdcl_W^{**}$] $cdcl_W$*-merge-stgy-rtranclp-*$cdcl_W$ **by** *auto*

lemma $cdcl_W$*-merge-stgy-cases*[*consumes 1*, *case-names fw-s-cp fw-s-decide*]:
assumes
$cdcl_W$*-merge-stgy S U*
full1 $cdcl_W$*-merge-cp S U* $\Longrightarrow$ *P*
$\bigwedge T$. *decide S T* $\Longrightarrow$ *no-step* $cdcl_W$*-merge-cp S* $\Longrightarrow$ *full* $cdcl_W$*-merge-cp T U* $\Longrightarrow$ *P*
shows *P*
using *assms* **by** (*auto simp*: $cdcl_W$*-merge-stgy.simps*)

inductive $cdcl_W$*-s'-w* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **where**
conflict': *full1* $cdcl_W$*-s'-without-decide S S'* $\Longrightarrow$ $cdcl_W$*-s'-w S S'* |
decide': *decide S S'* $\Longrightarrow$ *no-step* $cdcl_W$*-s'-without-decide S* $\Longrightarrow$ *full* $cdcl_W$*-s'-without-decide S' S''*
$\Longrightarrow$ $cdcl_W$*-s'-w S S''*

lemma $cdcl_W$*-s'-w-rtranclp-*$cdcl_W$:
$cdcl_W$*-s'-w S T* $\Longrightarrow$ $cdcl_W^{**}$ *S T*
apply (*induction rule*: $cdcl_W$*-s'-w.induct*)
using *rtranclp-*$cdcl_W$*-s'-without-decide-rtranclp-*$cdcl_W$ **unfolding** *full1-def*
apply (*simp add*: *tranclp-into-rtranclp*)
using *rtranclp-*$cdcl_W$*-s'-without-decide-rtranclp-*$cdcl_W$ **unfolding** *full-def*
by (*meson decide other rtranclp-into-tranclp2 tranclp-into-rtranclp*)

lemma *rtranclp-*$cdcl_W$*-s'-w-rtranclp-*$cdcl_W$:
$cdcl_W$*-s'-w*** *S T* $\Longrightarrow$ $cdcl_W^{**}$ *S T*
using *rtranclp-mono*[*of* $cdcl_W$*-s'-w* $cdcl_W^{**}$] $cdcl_W$*-s'-w-rtranclp-*$cdcl_W$ **by** *auto*

lemma *no-step-*$cdcl_W$*-cp-no-step-*$cdcl_W$*-s'-without-decide*:
assumes *no-step* $cdcl_W$*-cp S* **and** *conflicting S = None* **and** *inv*: $cdcl_W$*-M-level-inv S*
shows *no-step* $cdcl_W$*-s'-without-decide S*
by (*metis assms* $cdcl_W$*-cp.conflict'* $cdcl_W$*-cp.propagate'* $cdcl_W$*-merge-restart-cases tranclpD*
conflicting-true-no-step-$cdcl_W$*-merge-cp-no-step-s'-without-decide*)

lemma *no-step-*$cdcl_W$*-cp-no-step-*$cdcl_W$*-merge-restart*:
assumes *no-step* $cdcl_W$*-cp S* **and** *conflicting S = None*
shows *no-step* $cdcl_W$*-merge-cp S*
by (*metis assms*(*1*) $cdcl_W$*-cp.conflict'* $cdcl_W$*-cp.propagate'* $cdcl_W$*-merge-restart-cases tranclpD*)
lemma *after-*$cdcl_W$*-s'-without-decide-no-step-*$cdcl_W$*-cp*:
assumes $cdcl_W$*-s'-without-decide S T*
shows *no-step* $cdcl_W$*-cp T*
using *assms* **by** (*induction rule*: $cdcl_W$*-s'-without-decide.induct*) (*auto simp*: *full1-def full-def*)

lemma *no-step-*$cdcl_W$*-s'-without-decide-no-step-*$cdcl_W$*-cp*:
$cdcl_W$*-all-struct-inv S* $\Longrightarrow$ *no-step* $cdcl_W$*-s'-without-decide S* $\Longrightarrow$ *no-step* $cdcl_W$*-cp S*
by (*simp add*: *conflicting-true-no-step-s'-without-decide-no-step-*$cdcl_W$*-merge-cp*
no-step-$cdcl_W$*-merge-cp-no-step-*$cdcl_W$*-cp* $cdcl_W$*-all-struct-inv-def*)

lemma *after-*$cdcl_W$*-s'-w-no-step-*$cdcl_W$*-cp*:
assumes $cdcl_W$*-s'-w S T* **and** $cdcl_W$*-all-struct-inv S*
shows *no-step* $cdcl_W$*-cp T*
using *assms*
proof (*induction rule*: $cdcl_W$*-s'-w.induct*)
case *conflict'*
then show *?case*
by (*auto simp*: *full1-def tranclp-unfold-end after-*$cdcl_W$*-s'-without-decide-no-step-*$cdcl_W$*-cp*)

next
  case (*decide′ S T U*)
  moreover
    then have $cdcl_W{}^{**}$ *S U*
      using *rtranclp-*$cdcl_W$*-s′-without-decide-rtranclp-*$cdcl_W$[*of T U*] $cdcl_W$*.other*[*of S T*]
      $cdcl_W$*-o.decide* **unfolding** *full-def* **by** *auto*
    then have $cdcl_W$*-all-struct-inv U*
      using *decide′.prems rtranclp-*$cdcl_W$*-all-struct-inv-inv* **by** *blast*
  ultimately show *?case*
    using *no-step-*$cdcl_W$*-s′-without-decide-no-step-*$cdcl_W$*-cp* **unfolding** *full-def* **by** *blast*
qed

lemma *rtranclp-*$cdcl_W$*-s′-w-no-step-*$cdcl_W$*-cp-or-eq*:
  assumes $cdcl_W$*-s′-w*** *S T* **and** $cdcl_W$*-all-struct-inv S*
  shows $S = T \lor$ *no-step* $cdcl_W$*-cp T*
  using *assms*
proof (*induction rule*: *rtranclp-induct*)
  case *base*
  then show *?case* **by** *simp*
next
  case (*step T U*)
  moreover have $cdcl_W$*-all-struct-inv T*
    using *rtranclp-*$cdcl_W$*-s′-w-rtranclp-*$cdcl_W$[*of S U*] *assms*(*2*) *rtranclp-*$cdcl_W$*-all-struct-inv-inv*
    rtranclp-$cdcl_W$*-s′-w-rtranclp-*$cdcl_W$ *step.hyps*(*1*) **by** *blast*
  ultimately show *?case* **using** *after-*$cdcl_W$*-s′-w-no-step-*$cdcl_W$*-cp* **by** *fast*
qed

lemma *rtranclp-*$cdcl_W$*-merge-stgy′-no-step-*$cdcl_W$*-cp-or-eq*:
  assumes $cdcl_W$*-merge-stgy*** *S T* **and** *inv*: $cdcl_W$*-all-struct-inv S*
  shows $S = T \lor$ *no-step* $cdcl_W$*-cp T*
  using *assms*
proof (*induction rule*: *rtranclp-induct*)
  case *base*
  then show *?case* **by** *simp*
next
  case (*step T U*)
  moreover have $cdcl_W$*-all-struct-inv T*
    using *rtranclp-*$cdcl_W$*-merge-stgy-rtranclp-*$cdcl_W$[*of S U*] *assms*(*2*) *rtranclp-*$cdcl_W$*-all-struct-inv-inv*
    rtranclp-$cdcl_W$*-s′-w-rtranclp-*$cdcl_W$ *step.hyps*(*1*)
    by (*meson rtranclp-*$cdcl_W$*-merge-stgy-rtranclp-*$cdcl_W$)
  ultimately show *?case*
    using *after-*$cdcl_W$*-s′-w-no-step-*$cdcl_W$*-cp inv* **unfolding** $cdcl_W$*-all-struct-inv-def*
    by (*metis* $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-merge-stgy.simps full1-def full-def*
      no-step-$cdcl_W$*-merge-cp-no-step-*$cdcl_W$*-cp rtranclp-*$cdcl_W$*-all-struct-inv-inv*
      rtranclp-$cdcl_W$*-merge-stgy-rtranclp-*$cdcl_W$ *tranclp.intros*(*1*) *tranclp-into-rtranclp*)
qed

lemma *no-step-*$cdcl_W$*-s′-without-decide-no-step-*$cdcl_W$*-bj*:
  assumes *no-step* $cdcl_W$*-s′-without-decide S* **and** *inv*: $cdcl_W$*-all-struct-inv S*
  shows *no-step* $cdcl_W$*-bj S*
proof (*rule ccontr*)
  assume ¬ *?thesis*
  then obtain *T* **where** *S-T*: $cdcl_W$*-bj S T*
    by *auto*
  have $cdcl_W$*-all-struct-inv T*

using *S-T* $cdcl_W$*-all-struct-inv-inv inv other* **by** *blast*
then obtain T' **where** *full1* $cdcl_W$*-bj S* T'
using $cdcl_W$*-bj-exists-normal-form*[*of T*] *full-fullI S-T* **unfolding** $cdcl_W$*-all-struct-inv-def*
by *metis*
moreover
then have $cdcl_W^{**}$ *S* T'
using *rtranclp-mono*[*of* $cdcl_W$*-bj* $cdcl_W$] $cdcl_W$*.other* $cdcl_W$*-o.bj tranclp-into-rtranclp*[*of* $cdcl_W$*-bj*]
unfolding *full1-def* **by** (*metis* (*full-types*) *predicate2D predicate2I*)
then have $cdcl_W$*-all-struct-inv* T'
using *inv rtranclp-*$cdcl_W$*-all-struct-inv-inv* **by** *blast*
then obtain *U* **where** *full* $cdcl_W$*-cp* T' *U*
using $cdcl_W$*-cp-normalized-element-all-inv* **by** *blast*
moreover have *no-step* $cdcl_W$*-cp S*
using *S-T* **by** (*auto simp*: $cdcl_W$*-bj.simps*)
ultimately show *False*
using *assms* $cdcl_W$*-s'-without-decide.intros*(*2*)[*of S* T' *U*] **by** *fast*
qed

lemma $cdcl_W$*-s'-w-no-step-*$cdcl_W$*-bj*:
assumes $cdcl_W$*-s'-w S T* **and** $cdcl_W$*-all-struct-inv S*
shows *no-step* $cdcl_W$*-bj T*
using *assms* **apply** *induction*
using *rtranclp-*$cdcl_W$*-s'-without-decide-rtranclp-*$cdcl_W$ *rtranclp-*$cdcl_W$*-all-struct-inv-inv*
no-step-$cdcl_W$*-s'-without-decide-no-step-*$cdcl_W$*-bj* **unfolding** *full1-def*
apply (*meson tranclp-into-rtranclp*)
using *rtranclp-*$cdcl_W$*-s'-without-decide-rtranclp-*$cdcl_W$ *rtranclp-*$cdcl_W$*-all-struct-inv-inv*
no-step-$cdcl_W$*-s'-without-decide-no-step-*$cdcl_W$*-bj* **unfolding** *full-def*
by (*meson* $cdcl_W$*-merge-restart-*$cdcl_W$ *fw-r-decide*)

lemma *rtranclp-*$cdcl_W$*-s'-w-no-step-*$cdcl_W$*-bj-or-eq*:
assumes $cdcl_W$*-s'-w*** *S T* **and** $cdcl_W$*-all-struct-inv S*
shows $S = T \lor$ *no-step* $cdcl_W$*-bj T*
using *assms* **apply** *induction*
apply *simp*
using *rtranclp-*$cdcl_W$*-s'-w-rtranclp-*$cdcl_W$ *rtranclp-*$cdcl_W$*-all-struct-inv-inv*
$cdcl_W$*-s'-w-no-step-*$cdcl_W$*-bj* **by** *meson*

lemma *rtranclp-*$cdcl_W$*-s'-no-step-*$cdcl_W$*-s'-without-decide-decomp-into-*$cdcl_W$*-merge*:
assumes
$cdcl_W$*-s'*** *R V* **and**
conflicting $R = None$ **and**
inv: $cdcl_W$*-all-struct-inv R*
shows ($cdcl_W$*-merge-stgy*** *R V* $\land$ *conflicting* $V = None$)
$\lor$ ($cdcl_W$*-merge-stgy*** *R V* $\land$ *conflicting* $V \neq None$ $\land$ *no-step* $cdcl_W$*-bj V*)
$\lor$ ($\exists$ *S T U.* $cdcl_W$*-merge-stgy*** *R S* $\land$ *no-step* $cdcl_W$*-merge-cp S* $\land$ *decide S T*
$\land$ $cdcl_W$*-merge-cp*** *T U* $\land$ *conflict U V*)
$\lor$ ($\exists$ *S T.* $cdcl_W$*-merge-stgy*** *R S* $\land$ *no-step* $cdcl_W$*-merge-cp S* $\land$ *decide S T*
$\land$ $cdcl_W$*-merge-cp*** *T V*
$\land$ *conflicting* $V = None$)
$\lor$ ($cdcl_W$*-merge-cp*** *R V* $\land$ *conflicting* $V = None$)
$\lor$ ($\exists$ *U.* $cdcl_W$*-merge-cp*** *R U* $\land$ *conflict U V*)
using *assms*(*1*,*2*)
proof *induction*
case *base*
then show *?case* **by** *simp*

```
next
  case (step V W) note st = this(1) and s' = this(2) and IH = this(3)[OF this(4)] and
    n-s-R = this(4)
  from s'
  show ?case
    proof cases
      case conflict'
      consider
          (s') cdcl_W-merge-stgy** R V
        | (dec-confl) S T U where cdcl_W-merge-stgy** R S and no-step cdcl_W-merge-cp S and
            decide S T and cdcl_W-merge-cp** T U and conflict U V
        | (dec) S T where cdcl_W-merge-stgy** R S and no-step cdcl_W-merge-cp S and decide S T
            and cdcl_W-merge-cp** T V and conflicting V = None
        | (cp) cdcl_W-merge-cp** R V
        | (cp-confl) U where cdcl_W-merge-cp** R U and conflict U V
        using IH by meson
      then show ?thesis
        proof cases
        next
          case s'
          then have R = V
            by (metis full1-def inv local.conflict' tranclp-unfold-begin
              rtranclp-cdcl_W-merge-stgy'-no-step-cdcl_W-cp-or-eq)
          consider
              (V-W) V = W
            | (propa) propagate++ V W and conflicting W = None
            | (propa-confl) V' where propagate** V V' and conflict V' W
            using tranclp-cdcl_W-cp-propagate-with-conflict-or-not[of V W] conflict'
            unfolding full-unfold full1-def by meson
          then show ?thesis
            proof cases
              case V-W
              then show ?thesis using ‹R = V› n-s-R by simp
            next
              case propa
              then show ?thesis using ‹R = V› by auto
            next
              case propa-confl
              moreover
                then have cdcl_W-merge-cp** V V'
                  by (metis rtranclp-unfold cdcl_W-merge-cp.propagate' r-into-rtranclp)
              ultimately show ?thesis using s' ‹R = V› by blast
            qed
        next
          case dec-confl note - = this(5)
          then have False using conflict' unfolding full1-def by (auto dest!: tranclpD)
          then show ?thesis by fast
        next
          case dec note T-V = this(4)
          consider
              (propa) propagate++ V W and conflicting W = None
            | (propa-confl) V' where propagate** V V' and conflict V' W
            using tranclp-cdcl_W-cp-propagate-with-conflict-or-not[of V W] conflict'
            unfolding full1-def by meson
          then show ?thesis
```

proof *cases*
case *propa*
then show *?thesis*
by ($meson$ T-V $cdcl_W$-$merge$-$cp.propagate'$ dec $rtranclp.rtrancl$-$into$-$rtrancl$)
next
case *propa-confl*
then have $cdcl_W$-$merge$-cp^{**} T V'
using T-V **by** ($metis$ $rtranclp$-$unfold$ $cdcl_W$-$merge$-$cp.propagate'$ $rtranclp.simps$)
then show *?thesis* **using** *dec propa-confl*(*2*) **by** *metis*
qed
next
case *cp*
consider
(*propa*) $propagate^{++}$ V W **and** *conflicting* W = *None*
| (*propa-confl*) V' **where** $propagate^{**}$ V V' **and** *conflict* V' W
using $tranclp$-$cdcl_W$-cp-$propagate$-$with$-$conflict$-or-not[of V W] $conflict'$
unfolding *full1-def* **by** *meson*
then show *?thesis*
proof *cases*
case *propa*
then show *?thesis* **by** ($meson$ $cdcl_W$-$merge$-$cp.propagate'$ cp $rtranclp.rtrancl$-$into$-$rtrancl$)
next
case *propa-confl*
then show *?thesis*
using *propa-confl*(*2*) **by** ($metis$ $rtranclp$-$unfold$ $cdcl_W$-$merge$-$cp.propagate'$
cp rtranclp.rtrancl-into-rtrancl)
qed
next
case *cp-confl*
then show *?thesis* **using** $conflict'$ **unfolding** *full1-def* **by** (*fastforce dest*!: *tranclpD*)
qed
next
case ($decide'$ V')
then have *conf-V*: *conflicting* V = *None*
by *auto*
consider
(s') $cdcl_W$-$merge$-$stgy^{**}$ R V
| (*dec-confl*) S T U **where** $cdcl_W$-$merge$-$stgy^{**}$ R S **and** *no-step* $cdcl_W$-$merge$-cp S **and**
decide S T **and** $cdcl_W$-$merge$-cp^{**} T U **and** *conflict* U V
| (*dec*) S T **where** $cdcl_W$-$merge$-$stgy^{**}$ R S **and** *no-step* $cdcl_W$-$merge$-cp S **and** *decide* S T
and $cdcl_W$-$merge$-cp^{**} T V **and** *conflicting* V = *None*
| (*cp*) $cdcl_W$-$merge$-cp^{**} R V
| (*cp-confl*) U **where** $cdcl_W$-$merge$-cp^{**} R U **and** *conflict* U V
using *IH* **by** *meson*
then show *?thesis*
proof *cases*
case s'
have *confl*-V': *conflicting* V' = *None* **using** $decide'$(*1*) **by** *auto*
have *full*: *full1* $cdcl_W$-cp V' W $\vee$ (V' = W $\wedge$ *no-step* $cdcl_W$-cp W)
using $decide'$(*3*) **unfolding** *full-unfold* **by** *blast*
consider
(V'-W) V' = W
| (*propa*) $propagate^{++}$ V' W **and** *conflicting* W = *None*
| (*propa-confl*) V'' **where** $propagate^{**}$ V' V'' **and** *conflict* V'' W
using $tranclp$-$cdcl_W$-cp-$propagate$-$with$-$conflict$-or-not[of V W] $decide'$

by (*metis* ‹*full1* $cdcl_W$*-cp* V' W ∨ $V' = W$ ∧ *no-step* $cdcl_W$*-cp* W› *full1-def*
tranclp-$cdcl_W$*-cp-propagate-with-conflict-or-not*)
then show *?thesis*
proof *cases*
case V'*-*W
then show *?thesis*
using *confl-*V' *local.decide*$'$(*1*,*2*) s' *conf-*V
no-step-$cdcl_W$*-cp-no-step-*$cdcl_W$*-merge-restart*[*of* V] **by** *blast*
next
case *propa*
then show *?thesis* **using** *local.decide*$'$(*1*,*2*) s' **by** (*metis* $cdcl_W$*-merge-cp.simps* *conf-*V
no-step-$cdcl_W$*-cp-no-step-*$cdcl_W$*-merge-restart* *r-into-rtranclp*)
next
case *propa-confl*
then have $cdcl_W$*-merge-cp*** V' V''
by (*metis* *rtranclp-unfold* $cdcl_W$*-merge-cp.propagate*$'$ *r-into-rtranclp*)
then show *?thesis*
using *local.decide*$'$(*1*,*2*) *propa-confl*(*2*) s' *conf-*V
no-step-$cdcl_W$*-cp-no-step-*$cdcl_W$*-merge-restart*
by *metis*
qed
next
case (*dec*) **note** $s' = this(1)$ **and** $dec = this(2)$ **and** $cp = this(3)$ **and** *ns-cp-*T = *this*(*4*)
have *full* $cdcl_W$*-merge-cp* T V
unfolding *full-def* **by** (*simp add*: *conf-*V *local.decide*$'$(*2*)
no-step-$cdcl_W$*-cp-no-step-*$cdcl_W$*-merge-restart* *ns-cp-*T)
moreover have *no-step* $cdcl_W$*-merge-cp* V
by (*simp add*: *conf-*V *local.decide*$'$(*2*) *no-step-*$cdcl_W$*-cp-no-step-*$cdcl_W$*-merge-restart*)
moreover have *no-step* $cdcl_W$*-merge-cp* S
by (*metis* *dec*)
ultimately have $cdcl_W$*-merge-stgy* S V
using *cp* **by** *blast*
then have $cdcl_W$*-merge-stgy*** R V **using** s' **by** *auto*
consider
(V'*-*W) $V' = W$
| (*propa*) *propagate*$^{++}$ V' W **and** *conflicting* W = *None*
| (*propa-confl*) V'' **where** *propagate*** V' V'' **and** *conflict* V'' W
using *tranclp-*$cdcl_W$*-cp-propagate-with-conflict-or-not*[*of* V' W] *decide*$'$
unfolding *full-unfold* *full1-def* **by** *meson*
then show *?thesis*
proof *cases*
case V'*-*W
moreover have *conflicting* V' = *None*
using *decide*$'$(*1*) **by** *auto*
ultimately show *?thesis*
using ‹$cdcl_W$*-merge-stgy*** R V› *decide*$'$ ‹*no-step* $cdcl_W$*-merge-cp* V› **by** *blast*
next
case *propa*
moreover then have $cdcl_W$*-merge-cp* V' W
by *auto*
ultimately show *?thesis*
using ‹$cdcl_W$*-merge-stgy*** R V› *decide*$'$ ‹*no-step* $cdcl_W$*-merge-cp* V›
by (*meson* *r-into-rtranclp*)
next
case *propa-confl*

moreover then have $cdcl_W$*-merge-cp*** V' V''
by (*metis* $cdcl_W$*-merge-cp.propagate'* *rtranclp-unfold tranclp-unfold-end*)
ultimately show *?thesis* **using** ⟨$cdcl_W$*-merge-stgy*** R V⟩ *decide'*
⟨*no-step* $cdcl_W$*-merge-cp* V⟩ **by** (*meson r-into-rtranclp*)
qed
next
case *cp*
have *no-step* $cdcl_W$*-merge-cp* V
using *conf-V local.decide'*(*2*) *no-step-*$cdcl_W$*-cp-no-step-*$cdcl_W$*-merge-restart* **by** *blast*
then have *full* $cdcl_W$*-merge-cp* R V
unfolding *full-def* **using** *cp* **by** *fast*
then have $cdcl_W$*-merge-stgy*** R V
unfolding *full-unfold* **by** *auto*
have *full1* $cdcl_W$*-cp* V' W $\vee$ ($V' = W \wedge$ *no-step* $cdcl_W$*-cp* W)
using *decide'*(*3*) **unfolding** *full-unfold* **by** *blast*

consider
(V'*-W*) $V' = W$
| (*propa*) *propagate*$^{++}$ V' W **and** *conflicting* W = *None*
| (*propa-confl*) V'' **where** *propagate*** V' V'' **and** *conflict* V'' W
using *tranclp-*$cdcl_W$*-cp-propagate-with-conflict-or-not*[*of* V' W] *decide'*
unfolding *full-unfold full1-def* **by** *meson*
then show *?thesis*

proof *cases*
case V'*-W*
moreover have *conflicting* V' = *None*
using *decide'*(*1*) **by** *auto*
ultimately show *?thesis*
using ⟨$cdcl_W$*-merge-stgy*** R V⟩ *decide'* ⟨*no-step* $cdcl_W$*-merge-cp* V⟩ **by** *blast*
next
case *propa*
moreover then have $cdcl_W$*-merge-cp* V' W
by *auto*
ultimately show *?thesis* **using** ⟨$cdcl_W$*-merge-stgy*** R V⟩ *decide'*
⟨*no-step* $cdcl_W$*-merge-cp* V⟩ **by** (*meson r-into-rtranclp*)
next
case *propa-confl*
moreover then have $cdcl_W$*-merge-cp*** V' V''
by (*metis* $cdcl_W$*-merge-cp.propagate'* *rtranclp-unfold tranclp-unfold-end*)
ultimately show *?thesis* **using** ⟨$cdcl_W$*-merge-stgy*** R V⟩ *decide'*
⟨*no-step* $cdcl_W$*-merge-cp* V⟩ **by** (*meson r-into-rtranclp*)
qed
next
case (*dec-confl*)
show *?thesis* **using** *conf-V dec-confl*(*5*) **by** *auto*
next
case *cp-confl*
then show *?thesis* **using** *decide'* **apply** − **by** (*intro HOL.disjI2*) *fastforce*
qed
next
case (*bj'* V')
then have ¬*no-step* $cdcl_W$*-bj* V
by (*auto dest*: *tranclpD simp*: *full1-def*)
then consider

(s') *$cdcl_W$-merge-stgy*** *R V* **and** *conflicting V = None*
| (*dec-confl*) *S T U* **where** *$cdcl_W$-merge-stgy*** *R S* **and** *no-step $cdcl_W$-merge-cp S* **and**
decide S T **and** *$cdcl_W$-merge-cp*** *T U* **and** *conflict U V*
| (*dec*) *S T* **where** *$cdcl_W$-merge-stgy*** *R S* **and** *no-step $cdcl_W$-merge-cp S* **and** *decide S T*
and *$cdcl_W$-merge-cp*** *T V* **and** *conflicting V = None*
| (*cp*) *$cdcl_W$-merge-cp*** *R V* **and** *conflicting V = None*
| (*cp-confl*) *U* **where** *$cdcl_W$-merge-cp*** *R U* **and** *conflict U V*
using *IH* **by** *meson*
then show *?thesis*
proof *cases*
case s' **note** - =*this*(*2*)
then have *False*
using *bj'*(*1*) **unfolding** *full1-def* **by** (*force dest*!: *tranclpD simp*: *$cdcl_W$-bj.simps*)
then show *?thesis* **by** *fast*
next
case *dec* **note** - = *this*(*5*)
then have *False*
using *bj'*(*1*) **unfolding** *full1-def* **by** (*force dest*!: *tranclpD simp*: *$cdcl_W$-bj.simps*)
then show *?thesis* **by** *fast*
next
case *dec-confl*
then have *$cdcl_W$-merge-cp U V'*
using *bj' $cdcl_W$-merge-cp.intros*(*1*)[*of U V V'*] **by** (*simp add*: *full-unfold*)
then have *$cdcl_W$-merge-cp*** *T V'*
using *dec-confl*(*4*) **by** *simp*
consider
(*V'-W*) *V' = W*
| (*propa*) *propagate*$^{++}$ *V' W* **and** *conflicting W = None*
| (*propa-confl*) *V''* **where** *propagate*** *V' V''* **and** *conflict V'' W*
using *tranclp-$cdcl_W$-cp-propagate-with-conflict-or-not*[*of V' W*] *bj'*(*3*)
unfolding *full-unfold full1-def* **by** *meson*
then show *?thesis*
proof *cases*
case *V'-W*
then have *no-step $cdcl_W$-cp V'*
using *bj'*(*3*) **unfolding** *full-def* **by** *auto*
then have *no-step $cdcl_W$-merge-cp V'*
by (*metis $cdcl_W$-cp.propagate' $cdcl_W$-merge-cp.cases tranclpD*
no-step-$cdcl_W$-cp-no-conflict-no-propagate(*1*))
then have *full1 $cdcl_W$-merge-cp T V'*
unfolding *full1-def* **using** ‹*$cdcl_W$-merge-cp U V'*› *dec-confl*(*4*) **by** *auto*
then have *full $cdcl_W$-merge-cp T V'*
by (*simp add*: *full-unfold*)
then have *$cdcl_W$-merge-stgy S V'*
using *dec-confl*(*3*) *$cdcl_W$-merge-stgy.fw-s-decide* ‹*no-step $cdcl_W$-merge-cp S*› **by** *blast*
then have *$cdcl_W$-merge-stgy*** *R V'*
using ‹*$cdcl_W$-merge-stgy*** *R S*› **by** *auto*
show *?thesis*
proof *cases*
assume *conflicting W = None*
then show *?thesis* **using** ‹*$cdcl_W$-merge-stgy*** *R V'*› ‹*V' = W*› **by** *auto*
next
assume *conflicting W ≠ None*
then show *?thesis*
using ‹*$cdcl_W$-merge-stgy*** *R V'*› ‹*V' = W*› **by** (*metis* ‹*$cdcl_W$-merge-cp U V'*›

conflicting-not-true-rtranclp-cdcl$_W$-merge-cp-no-step-cdcl$_W$-bj dec-confl(5) r-into-rtranclp conflictE)
qed
next
case *propa*
moreover then have *cdcl$_W$-merge-cp V' W*
by *auto*
ultimately show *?thesis* **using** *decide'* **by** (*meson* ‹*cdcl$_W$-merge-cp** T V'*› *dec-confl(1−3) rtranclp.rtrancl-into-rtrancl*)
next
case *propa-confl*
moreover then have *cdcl$_W$-merge-cp** V' V''*
by (*metis cdcl$_W$-merge-cp.propagate' rtranclp-unfold tranclp-unfold-end*)
ultimately show *?thesis* **by** (*meson* ‹*cdcl$_W$-merge-cp** T V'*› *dec-confl(1−3) rtranclp-trans*)
qed
next
case *cp* **note** - = *this(2)*
then show *?thesis* **using** *bj'(1)* ‹$\neg$ *no-step cdcl$_W$-bj V*›
conflicting-not-true-rtranclp-cdcl$_W$-merge-cp-no-step-cdcl$_W$-bj **by** *auto*
next
case *cp-confl*
then have *cdcl$_W$-merge-cp U V'* **by** (*simp add*: *cdcl$_W$-merge-cp.conflict' full-unfold local.bj'(1)*))
consider
(V'-W) $V' = W$
| (*propa*) *propagate^{++} V' W* **and** *conflicting W = None*
| (*propa-confl*) V'' **where** *propagate** V' V''* **and** *conflict V'' W*
using *tranclp-cdcl$_W$-cp-propagate-with-conflict-or-not[of V' W] bj'*
unfolding *full-unfold full1-def* **by** *meson*
then show *?thesis*

proof *cases*
case *V'-W*
show *?thesis*
proof *cases*
assume *conflicting V' = None*
then show *?thesis*
using *V'-W* ‹*cdcl$_W$-merge-cp U V'*› *cp-confl(1)* **by** *force*
next
assume *confl*: *conflicting $V' \neq$ None*
then have *no-step cdcl$_W$-merge-stgy V'*
by (*fastforce simp*: *cdcl$_W$-merge-stgy.simps full1-def full-def cdcl$_W$-merge-cp.simps dest*!: *tranclpD*)
have *no-step cdcl$_W$-merge-cp V'*
using *confl* **by** (*auto simp*: *full1-def full-def cdcl$_W$-merge-cp.simps dest*!: *tranclpD*)
moreover have *cdcl$_W$-merge-cp U W*
using *V'-W* ‹*cdcl$_W$-merge-cp U V'*› **by** *blast*
ultimately have *full1 cdcl$_W$-merge-cp R V'*
using *cp-confl(1) V'-W* **unfolding** *full1-def* **by** *auto*
then have *cdcl$_W$-merge-stgy R V'*
by *auto*
moreover have *no-step cdcl$_W$-merge-stgy V'*
using *confl* ‹*no-step cdcl$_W$-merge-cp V'*› **by** (*auto simp*: *cdcl$_W$-merge-stgy.simps full1-def dest*!: *tranclpD*)

ultimately have $cdcl_W$*-merge-stgy*** R V' **by** *auto*
show *?thesis* **by** (*metis* V'*-W* ‹$cdcl_W$*-merge-cp* U V'› ‹$cdcl_W$*-merge-stgy*** R V'›
conflicting-not-true-rtranclp-$cdcl_W$*-merge-cp-no-step-*$cdcl_W$*-bj cp-confl*(*1*)
rtranclp.rtrancl-into-rtrancl step.prems)
qed
next
case *propa*
moreover then have $cdcl_W$*-merge-cp* V' W
by *auto*
ultimately show *?thesis* **using** ‹$cdcl_W$*-merge-cp* U V'› *cp-confl*(*1*) **by** *force*
next
case *propa-confl*
moreover then have $cdcl_W$*-merge-cp*** V' V''
by (*metis* $cdcl_W$*-merge-cp.propagate'* *rtranclp-unfold tranclp-unfold-end*)
ultimately show *?thesis*
using ‹$cdcl_W$*-merge-cp* U V'› *cp-confl*(*1*) **by** (*metis rtranclp.rtrancl-into-rtrancl*
rtranclp-trans)
qed
qed
qed
qed

lemma *decide-rtranclp-*$cdcl_W$*-s'-rtranclp-*$cdcl_W$*-s'*:
assumes
dec: *decide* S T **and**
$cdcl_W$*-s'*** T U **and**
n-s-S: *no-step* $cdcl_W$*-cp* S **and**
no-step $cdcl_W$*-cp* U
shows $cdcl_W$*-s'*** S U
using *assms*(*2*,*4*)
proof *induction*
case (*step* U V) **note** *st* =*this*(*1*) **and** *s'* = *this*(*2*) **and** *IH* = *this*(*3*) **and** *n-s* = *this*(*4*)
consider
(TU) $T = U$
| (*s'-st*) T' **where** $cdcl_W$*-s'* T T' **and** $cdcl_W$*-s'*** T' U
using *st*[*unfolded rtranclp-unfold*] **by** (*auto dest*!: *tranclpD*)
then show *?case*
proof *cases*
case TU
then show *?thesis*
proof −
assume *a1*: $T = U$
then have *f2*: $cdcl_W$*-s'* T V
using *s'* **by** *force*
obtain *ss* :: *'st* **where**
$cdcl_W$*-s'*** S T ∨ $cdcl_W$*-cp* T *ss*
using *a1 step.IH* **by** *blast*
then show *?thesis*
using *f2* **by** (*metis* (*full-types*) $cdcl_W$*-s'.decide'* $cdcl_W$*-s'E dec full1-is-full n-s-S*
rtranclp-unfold tranclp-unfold-end)
qed
next
case (*s'-st* T') **note** *s'-T'* = *this*(*1*) **and** *st* = *this*(*2*)
have $cdcl_W$*-s'*** S T'
using *s'-T'*

proof *cases*
case *conflict'*
then have $cdcl_W$*-s' S T'*
using *dec* $cdcl_W$*-s'.decide' n-s-S* **by** (*simp add*: *full-unfold*)
then show *?thesis*
using *st* **by** *auto*
next
case (*decide' T''*)
then have $cdcl_W$*-s' S T*
using *dec* $cdcl_W$*-s'.decide' n-s-S* **by** (*simp add*: *full-unfold*)
then show *?thesis* **using** *decide' s'-T'* **by** *auto*
next
case *bj'*
then have *False*
using *dec* **unfolding** *full1-def* **by** (*fastforce dest*!: *tranclpD simp*: $cdcl_W$*-bj.simps*)
then show *?thesis* **by** *fast*
qed
then show *?thesis* **using** *s' st* **by** *auto*
qed
next
case *base*
then have *full* $cdcl_W$*-cp T T*
by (*simp add*: *full-unfold*)
then show *?case*
using $cdcl_W$*-s'.simps dec n-s-S* **by** *auto*
qed

lemma *rtranclp-*$cdcl_W$*-merge-stgy-rtranclp-*$cdcl_W$*-s'*:
assumes
$cdcl_W$*-merge-stgy*** *R V* **and**
inv: $cdcl_W$*-all-struct-inv R*
shows $cdcl_W$*-s'*** *R V*
using *assms*(*1*)
proof *induction*
case *base*
then show *?case* **by** *simp*
next
case (*step S T*) **note** *st* = *this*(*1*) **and** *fw* = *this*(*2*) **and** *IH* = *this*(*3*)
have $cdcl_W$*-all-struct-inv S*
using *inv rtranclp-*$cdcl_W$*-all-struct-inv-inv rtranclp-*$cdcl_W$*-merge-stgy-rtranclp-*$cdcl_W$ *st* **by** *blast*
from *fw* **show** *?case*
proof (*cases rule*: $cdcl_W$*-merge-stgy-cases*)
case *fw-s-cp*
then show *?thesis*
proof −
assume *a1*: *full1* $cdcl_W$*-merge-cp S T*
obtain *ss* :: (*'st* ⇒ *'st* ⇒ *bool*) ⇒ *'st* ⇒ *'st* **where**
f2: $\bigwedge$*p s sa pa sb sc sd pb se sf*. (¬ *full1 p* (*s*::*'st*) *sa* ∨ p^{++} *s sa*)
∧ (¬ *pa* (*sb*::*'st*) *sc* ∨ ¬ *full1 pa sd sb*) ∧ (¬ pb^{++} *se sf* ∨ *pb sf* (*ss pb sf*)
∨ *full1 pb se sf*)
by (*metis* (*no-types*) *full1-def*)
then have *f3*: $cdcl_W$*-merge-cp*$^{++}$ *S T*
using *a1* **by** *auto*
obtain *ssa* :: (*'st* ⇒ *'st* ⇒ *bool*) ⇒ *'st* ⇒ *'st* ⇒ *'st* **where**
f4: $\bigwedge$*p s sa*. ¬ p^{++} *s sa* ∨ *p s* (*ssa p s sa*)

by (*meson tranclp-unfold-begin*)
then have *f5*: $\bigwedge s.$ ¬ *full1* $cdcl_W$*-merge-cp s S*
using *f3 f2* by (*metis* (*full-types*))
have $\bigwedge s.$ ¬ *full* $cdcl_W$*-merge-cp s S*
using *f4 f3* by (*meson full-def*)
then have $S = R$
using *f5* by (*metis* (*no-types*) $cdcl_W$*-merge-stgy.simps rtranclp-unfold st tranclp-unfold-end*)
then show *?thesis*
using *f2 a1* by (*metis* (*no-types*) ‹$cdcl_W$*-all-struct-inv S*›
conflicting-true-full1-$cdcl_W$*-merge-cp-imp-full1-*$cdcl_W$*-s'-without-decode*
rtranclp-$cdcl_W$*-s'-without-decide-rtranclp-*$cdcl_W$*-s' rtranclp-unfold*)
qed
next
case (*fw-s-decide S'*) note *dec* = *this*(*1*) and *n-S* = *this*(*2*) and *full* = *this*(*3*)
moreover then have *conflicting S'* = *None*
by *auto*
ultimately have *full* $cdcl_W$*-s'-without-decide S' T*
by (*meson* ‹$cdcl_W$*-all-struct-inv S*› $cdcl_W$*-merge-restart-*$cdcl_W$ *fw-r-decide*
rtranclp-$cdcl_W$*-all-struct-inv-inv*
conflicting-true-full-$cdcl_W$*-merge-cp-iff-full-*$cdcl_W$*-s'-without-decode*)
then have *a1*: $cdcl_W$*-s'*** *S' T*
unfolding *full-def* by (*metis* (*full-types*)*rtranclp-*$cdcl_W$*-s'-without-decide-rtranclp-*$cdcl_W$*-s'*)
have $cdcl_W$*-merge-stgy*** *S T*
using *fw* by *blast*
then have $cdcl_W$*-s'*** *S T*
using *decide-rtranclp-*$cdcl_W$*-s'-rtranclp-*$cdcl_W$*-s' a1* by (*metis* ‹$cdcl_W$*-all-struct-inv S*› *dec*
n-S no-step-$cdcl_W$*-merge-cp-no-step-*$cdcl_W$*-cp* $cdcl_W$*-all-struct-inv-def*
rtranclp-$cdcl_W$*-merge-stgy'-no-step-*$cdcl_W$*-cp-or-eq*)
then show *?thesis* using *IH* by *auto*
qed
qed

lemma *rtranclp-*$cdcl_W$*-merge-stgy-distinct-mset-clauses*:
assumes *invR*: $cdcl_W$*-all-struct-inv R* and
st: $cdcl_W$*-merge-stgy*** *R S* and
dist: *distinct-mset* (*clauses R*) and
R: *trail R* = []
shows *distinct-mset* (*clauses S*)
using *rtranclp-*$cdcl_W$*-stgy-distinct-mset-clauses*[*OF invR - dist R*]
invR st rtranclp-mono[*of* $cdcl_W$*-s'* $cdcl_W$*-stgy***] $cdcl_W$*-s'-is-rtranclp-*$cdcl_W$*-stgy*
by (*auto dest*!: $cdcl_W$*-s'-is-rtranclp-*$cdcl_W$*-stgy rtranclp-*$cdcl_W$*-merge-stgy-rtranclp-*$cdcl_W$*-s'*)

lemma *no-step-*$cdcl_W$*-s'-no-step-*$cdcl_W$*-merge-stgy*:
assumes
inv: $cdcl_W$*-all-struct-inv R* and *s'*: *no-step* $cdcl_W$*-s' R*
shows *no-step* $cdcl_W$*-merge-stgy R*
proof –
{ fix *ss* :: *'st*
obtain *ssa* :: *'st* ⇒ *'st* ⇒ *'st* where
ff1: $\bigwedge s\ sa.$ ¬ $cdcl_W$*-merge-stgy s sa* ∨ *full1* $cdcl_W$*-merge-cp s sa* ∨ *decide s* (*ssa s sa*)
using $cdcl_W$*-merge-stgy.cases* by *moura*
obtain *ssb* :: (*'st* ⇒ *'st* ⇒ *bool*) ⇒ *'st* ⇒ *'st* ⇒ *'st* where
ff2: $\bigwedge p\ s\ sa.$ ¬ p^{++} *s sa* ∨ *p s* (*ssb p s sa*)
by (*meson tranclp-unfold-begin*)

obtain *ssc* :: *′st* ⇒ *′st* **where**
ff3: $\bigwedge$*s sa sb*. (¬ $cdcl_W$*-all-struct-inv s* ∨ ¬ $cdcl_W$*-cp s sa* ∨ $cdcl_W$*-s′ s* (*ssc s*))
∧ (¬ $cdcl_W$*-all-struct-inv s* ∨ ¬ $cdcl_W$*-o s sb* ∨ $cdcl_W$*-s′ s* (*ssc s*))
using *n-step-*$cdcl_W$*-stgy-iff-no-step-*$cdcl_W$*-cl-*$cdcl_W$*-o* **by** *moura*
then have *ff4*: $\bigwedge$*s*. ¬ $cdcl_W$*-o R s*
using *s′ inv* **by** *blast*
have *ff5*: $\bigwedge$*s*. ¬ $cdcl_W$*-cp*$^{++}$ *R s*
using *ff3 ff2 s′* **by** (*metis inv*)
have $\bigwedge$*s*. ¬ $cdcl_W$*-bj*$^{++}$ *R s*
using *ff4 ff2* **by** (*metis bj*)
then have $\bigwedge$*s*. ¬ $cdcl_W$*-s′-without-decide R s*
using *ff5* **by** (*simp add*: $cdcl_W$*-s′-without-decide.simps full1-def*)
then have ¬ $cdcl_W$*-s′-without-decide*$^{++}$ *R ss*
using *ff2* **by** *blast*
then have ¬ $cdcl_W$*-merge-stgy R ss*
using *ff4 ff1* **by** (*metis* (*full-types*) *decide full1-def inv*
conflicting-true-full1-$cdcl_W$*-merge-cp-imp-full1-*$cdcl_W$*-s′-without-decode*) **}**
then show *?thesis*
by *fastforce*
qed

lemma *wf-*$cdcl_W$*-merge-cp*:
wf{(*T*, *S*). $cdcl_W$*-all-struct-inv S* ∧ $cdcl_W$*-merge-cp S T*}
using *wf-tranclp-*$cdcl_W$*-merge* **by** (*rule wf-subset*) (*auto simp*: $cdcl_W$*-merge-cp-tranclp-*$cdcl_W$*-merge*)

lemma *wf-*$cdcl_W$*-merge-stgy*:
wf{(*T*, *S*). $cdcl_W$*-all-struct-inv S* ∧ $cdcl_W$*-merge-stgy S T*}
using *wf-tranclp-*$cdcl_W$*-merge* **by** (*rule wf-subset*)
(*auto simp add*: $cdcl_W$*-merge-stgy-tranclp-*$cdcl_W$*-merge*)

lemma $cdcl_W$*-merge-cp-obtain-normal-form*:
assumes *inv*: $cdcl_W$*-all-struct-inv R*
obtains *S* **where** *full* $cdcl_W$*-merge-cp R S*
proof −
obtain *S* **where** *full* (*λS T*. $cdcl_W$*-all-struct-inv S* ∧ $cdcl_W$*-merge-cp S T*) *R S*
using *wf-exists-normal-form-full*[*OF wf-*$cdcl_W$*-merge-cp*] **by** *blast*
then have
st: (*λS T*. $cdcl_W$*-all-struct-inv S* ∧ $cdcl_W$*-merge-cp S T*)** *R S* **and**
n-s: *no-step* (*λS T*. $cdcl_W$*-all-struct-inv S* ∧ $cdcl_W$*-merge-cp S T*) *S*
unfolding *full-def* **by** *blast+*
have $cdcl_W$*-merge-cp*** *R S*
using *st* **by** *induction auto*
moreover
have $cdcl_W$*-all-struct-inv S*
using *st inv*
apply (*induction rule*: *rtranclp-induct*)
apply *simp*
by (*meson r-into-rtranclp rtranclp-*$cdcl_W$*-all-struct-inv-inv*
rtranclp-$cdcl_W$*-merge-cp-rtranclp-*$cdcl_W$)
then have *no-step* $cdcl_W$*-merge-cp S*
using *n-s* **by** *auto*
ultimately show *?thesis*
using *that* **unfolding** *full-def* **by** *blast*
qed

lemma *no-step-cdcl$_W$-merge-stgy-no-step-cdcl$_W$-s'*:
  assumes
    inv: *cdcl$_W$-all-struct-inv R* **and**
    confl: *conflicting R = None* **and**
    n-s: *no-step cdcl$_W$-merge-stgy R*
  shows *no-step cdcl$_W$-s' R*
proof (*rule ccontr*)
  assume ¬ *?thesis*
  then obtain *S* **where** *cdcl$_W$-s' R S* **by** *auto*
  then show *False*
    proof *cases*
      case *conflict'*
      then obtain *S'* **where** *full1 cdcl$_W$-merge-cp R S'*
        by (*metis* (*full-types*) *cdcl$_W$-merge-cp-obtain-normal-form cdcl$_W$-s'-without-decide.simps confl conflicting-true-no-step-cdcl$_W$-merge-cp-no-step-s'-without-decide full-def full-unfold inv cdcl$_W$-all-struct-inv-def*)
      then show *False* **using** *n-s* **by** *blast*
    next
      case (*decide' R'*)
      then have *cdcl$_W$-all-struct-inv R'*
        using *inv cdcl$_W$-all-struct-inv-inv cdcl$_W$.other cdcl$_W$-o.decide* **by** *meson*
      then obtain *R''* **where** *full cdcl$_W$-merge-cp R' R''*
        using *cdcl$_W$-merge-cp-obtain-normal-form* **by** *blast*
      moreover have *no-step cdcl$_W$-merge-cp R*
        by (*simp add*: *confl local.decide'*(*2*) *no-step-cdcl$_W$-cp-no-step-cdcl$_W$-merge-restart*)
      ultimately show *False* **using** *n-s cdcl$_W$-merge-stgy.intros local.decide'*(*1*) **by** *blast*
    next
      case (*bj' R'*)
      then show *False*
        using *confl no-step-cdcl$_W$-cp-no-step-cdcl$_W$-s'-without-decide inv*
        unfolding *cdcl$_W$-all-struct-inv-def* **by** *blast*
    qed
qed

lemma *rtranclp-cdcl$_W$-merge-cp-no-step-cdcl$_W$-bj*:
  assumes *conflicting R = None* **and** *cdcl$_W$-merge-cp** R S*
  shows *no-step cdcl$_W$-bj S*
  using *assms conflicting-not-true-rtranclp-cdcl$_W$-merge-cp-no-step-cdcl$_W$-bj* **by** *blast*

lemma *rtranclp-cdcl$_W$-merge-stgy-no-step-cdcl$_W$-bj*:
  assumes *confl*: *conflicting R = None* **and** *cdcl$_W$-merge-stgy** R S*
  shows *no-step cdcl$_W$-bj S*
  using *assms*(*2*)
proof *induction*
  case *base*
  then show *?case*
    using *confl* **by** (*auto simp*: *cdcl$_W$-bj.simps*)[]
next
  case (*step S T*) **note** *st = this*(*1*) **and** *fw = this*(*2*) **and** *IH = this*(*3*)
  have *confl-S*: *conflicting S = None*
    using *fw* **apply** *cases*
    by (*auto simp*: *full1-def cdcl$_W$-merge-cp.simps dest*!: *tranclpD*)
  from *fw* **show** *?case*
    proof *cases*
      case *fw-s-cp*

then show *?thesis*
using *rtranclp-cdcl*$_W$*-merge-cp-no-step-cdcl*$_W$*-bj confl-S*
by (*simp add*: *full1-def tranclp-into-rtranclp*)
next
case (*fw-s-decide S'*)
moreover then have *conflicting S'* = *None* **by** *auto*
ultimately show *?thesis*
using *conflicting-not-true-rtranclp-cdcl*$_W$*-merge-cp-no-step-cdcl*$_W$*-bj*
unfolding *full-def* **by** *meson*
qed
qed

lemma *full-cdcl*$_W$*-s'-full-cdcl*$_W$*-merge-restart*:
assumes
conflicting R = *None* **and**
inv: *cdcl*$_W$*-all-struct-inv R*
shows *full cdcl*$_W$*-s' R V* $\longleftrightarrow$ *full cdcl*$_W$*-merge-stgy R V* (**is** *?s'* $\longleftrightarrow$ *?fw*)
proof
assume *?s'*
then have *cdcl*$_W$*-s'*** *R V* **unfolding** *full-def* **by** *blast*
have *cdcl*$_W$*-all-struct-inv V*
using ‹*cdcl*$_W$*-s'*** *R V*› *inv rtranclp-cdcl*$_W$*-all-struct-inv-inv rtranclp-cdcl*$_W$*-s'-rtranclp-cdcl*$_W$
by *blast*
then have *n-s*: *no-step cdcl*$_W$*-merge-stgy V*
using *no-step-cdcl*$_W$*-s'-no-step-cdcl*$_W$*-merge-stgy* **by** (*meson* ‹*full cdcl*$_W$*-s' R V*› *full-def*)
have *n-s-bj*: *no-step cdcl*$_W$*-bj V*
by (*metis* ‹*cdcl*$_W$*-all-struct-inv V*› ‹*full cdcl*$_W$*-s' R V*› *bj full-def*
n-step-cdcl$_W$*-stgy-iff-no-step-cdcl*$_W$*-cl-cdcl*$_W$*-o*)
have *n-s-cp*: *no-step cdcl*$_W$*-merge-cp V*
proof −
{ **fix** *ss* :: *'st*
obtain *ssa* :: *'st* ⇒ *'st* **where**
ff1: ∀ *s*. ¬ *cdcl*$_W$*-all-struct-inv s* ∨ *cdcl*$_W$*-s'-without-decide s* (*ssa s*)
∨ *no-step cdcl*$_W$*-merge-cp s*
using *conflicting-true-no-step-s'-without-decide-no-step-cdcl*$_W$*-merge-cp* **by** *moura*
have (∀ *p s sa*. ¬ *full p* (*s*::*'st*) *sa* ∨ *p*** *s sa* ∧ *no-step p sa*) **and**
(∀ *p s sa*. (¬ *p*** (*s*::*'st*) *sa* ∨ (∃ *s*. *p sa s*)) ∨ *full p s sa*)
by (*meson full-def*)+
then have ¬ *cdcl*$_W$*-merge-cp V ss*
using *ff1* **by** (*metis* (*no-types*) ‹*cdcl*$_W$*-all-struct-inv V*› ‹*full cdcl*$_W$*-s' R V*› *cdcl*$_W$*-s'.simps*
cdcl$_W$*-s'-without-decide.cases*) }
then show *?thesis*
by *blast*
qed
consider
(*fw-no-confl*) *cdcl*$_W$*-merge-stgy*** *R V* **and** *conflicting V* = *None*
| (*fw-confl*) *cdcl*$_W$*-merge-stgy*** *R V* **and** *conflicting V* ≠ *None* **and** *no-step cdcl*$_W$*-bj V*
| (*fw-dec-confl*) *S T U* **where** *cdcl*$_W$*-merge-stgy*** *R S* **and** *no-step cdcl*$_W$*-merge-cp S* **and**
decide S T **and** *cdcl*$_W$*-merge-cp*** *T U* **and** *conflict U V*
| (*fw-dec-no-confl*) *S T* **where** *cdcl*$_W$*-merge-stgy*** *R S* **and** *no-step cdcl*$_W$*-merge-cp S* **and**
decide S T **and** *cdcl*$_W$*-merge-cp*** *T V* **and** *conflicting V* = *None*
| (*cp-no-confl*) *cdcl*$_W$*-merge-cp*** *R V* **and** *conflicting V* = *None*
| (*cp-confl*) *U* **where** *cdcl*$_W$*-merge-cp*** *R U* **and** *conflict U V*
using *rtranclp-cdcl*$_W$*-s'-no-step-cdcl*$_W$*-s'-without-decide-decomp-into-cdcl*$_W$*-merge*[*OF*
‹*cdcl*$_W$*-s'*** *R V*› *assms*] **by** *auto*

then show *?fw*
proof *cases*
case *fw-no-confl*
then show *?thesis* **using** *n-s* **unfolding** *full-def* **by** *blast*
next
case *fw-confl*
then show *?thesis* **using** *n-s* **unfolding** *full-def* **by** *blast*
next
case *fw-dec-confl*
have $cdcl_W$*-merge-cp U V*
using *n-s-bj* **by** (*metis* $cdcl_W$*-merge-cp.simps full-unfold fw-dec-confl*(*5*))
then have *full1* $cdcl_W$*-merge-cp T V*
unfolding *full1-def* **by** (*metis fw-dec-confl*(*4*) *n-s-cp tranclp-unfold-end*)
then have $cdcl_W$*-merge-stgy S V* **using** ‹*decide S T*› ‹*no-step* $cdcl_W$*-merge-cp S*› **by** *auto*
then show *?thesis* **using** *n-s* ‹ $cdcl_W$*-merge-stgy*** *R S*› **unfolding** *full-def* **by** *auto*
next
case *fw-dec-no-confl*
then have *full* $cdcl_W$*-merge-cp T V*
using *n-s-cp* **unfolding** *full-def* **by** *blast*
then have $cdcl_W$*-merge-stgy S V* **using** ‹*decide S T*› ‹*no-step* $cdcl_W$*-merge-cp S*› **by** *auto*
then show *?thesis* **using** *n-s* ‹ $cdcl_W$*-merge-stgy*** *R S*› **unfolding** *full-def* **by** *auto*
next
case *cp-no-confl*
then have *full* $cdcl_W$*-merge-cp R V*
by (*simp add*: *full-def n-s-cp*)
then have $R = V \lor$ $cdcl_W$*-merge-stgy*$^{++}$ *R V*
by (*metis* (*no-types*) *full-unfold fw-s-cp rtranclp-unfold tranclp-unfold-end*)
then show *?thesis*
by (*simp add*: *full-def n-s rtranclp-unfold*)
next
case *cp-confl*
have *full* $cdcl_W$*-bj V V*
using *n-s-bj* **unfolding** *full-def* **by** *blast*
then have *full1* $cdcl_W$*-merge-cp R V*
unfolding *full1-def* **by** (*meson* $cdcl_W$*-merge-cp.conflict′ cp-confl*(*1*,*2*) *n-s-cp rtranclp-into-tranclp1*)
then show *?thesis* **using** *n-s* **unfolding** *full-def* **by** *auto*
qed
next
assume *?fw*
then have $cdcl_W{}^{**}$ *R V* **using** *rtranclp-mono*[*of* $cdcl_W$*-merge-stgy* $cdcl_W{}^{**}$] $cdcl_W$*-merge-stgy-rtranclp-*$cdcl_W$ **unfolding** *full-def* **by** *auto*
then have *inv′*: $cdcl_W$*-all-struct-inv V* **using** *inv rtranclp-*$cdcl_W$*-all-struct-inv-inv* **by** *blast*
have $cdcl_W$*-s′*** *R V*
using ‹*?fw*› **by** (*simp add*: *full-def inv rtranclp-*$cdcl_W$*-merge-stgy-rtranclp-*$cdcl_W$*-s′*)
moreover have *no-step* $cdcl_W$*-s′ V*
proof *cases*
assume *conflicting V = None*
then show *?thesis*
by (*metis inv′* ‹*full* $cdcl_W$*-merge-stgy R V*› *full-def no-step-*$cdcl_W$*-merge-stgy-no-step-*$cdcl_W$*-s′*)
next
assume *confl-V*: *conflicting V ≠ None*
then have *no-step* $cdcl_W$*-bj V*
using *rtranclp-*$cdcl_W$*-merge-stgy-no-step-*$cdcl_W$*-bj* **by** (*meson* ‹*full* $cdcl_W$*-merge-stgy R V*›

assms(*1*) *full-def*)
then show *?thesis* **using** *confl-V* **by** (*fastforce simp*: $cdcl_W$*-s′.simps full1-def* $cdcl_W$*-cp.simps dest*!: *tranclpD*)
qed
ultimately show *?s′* **unfolding** *full-def* **by** *blast*
qed

lemma *full-*$cdcl_W$*-stgy-full-*$cdcl_W$*-merge*:
assumes
conflicting R = *None* **and**
inv: $cdcl_W$*-all-struct-inv R*
shows *full* $cdcl_W$*-stgy R V* ⟷ *full* $cdcl_W$*-merge-stgy R V*
by (*simp add*: *assms*(*1*) *full-*$cdcl_W$*-s′-full-*$cdcl_W$*-merge-restart full-*$cdcl_W$*-stgy-iff-full-*$cdcl_W$*-s′ inv*)

lemma *full-*$cdcl_W$*-merge-stgy-final-state-conclusive′*:
fixes *S′* :: *′st*
assumes *full*: *full* $cdcl_W$*-merge-stgy* (*init-state N*) *S′*
and *no-d*: *distinct-mset-mset N*
shows (*conflicting S′* = *Some* {#} ∧ *unsatisfiable* (*set-mset N*))
∨ (*conflicting S′* = *None* ∧ *trail S′* ⊨*asm N* ∧ *satisfiable* (*set-mset N*))
proof −
have $cdcl_W$*-all-struct-inv* (*init-state N*)
using *no-d* **unfolding** $cdcl_W$*-all-struct-inv-def* **by** *auto*
moreover have *conflicting* (*init-state N*) = *None*
by *auto*
ultimately show *?thesis*
by (*simp add*: *full full-*$cdcl_W$*-stgy-final-state-conclusive-from-init-state full-*$cdcl_W$*-stgy-full-*$cdcl_W$*-merge no-d*)
qed

end

7.6 Adding Restarts

locale $cdcl_W$*-restart* =
$cdcl_W$ *trail init-clss learned-clss backtrack-lvl conflicting cons-trail tl-trail add-init-cls add-learned-cls remove-cls update-backtrack-lvl update-conflicting init-state restart-state*
for
trail :: *′st* ⇒ (*′v*, *nat*, *′v clause*) *ann-literals* **and**
init-clss :: *′st* ⇒ *′v clauses* **and**
learned-clss :: *′st* ⇒ *′v clauses* **and**
backtrack-lvl :: *′st* ⇒ *nat* **and**
conflicting :: *′st* ⇒*′v clause option* **and**

cons-trail :: (*′v*, *nat*, *′v clause*) *ann-literal* ⇒ *′st* ⇒ *′st* **and**
tl-trail :: *′st* ⇒ *′st* **and**
add-init-cls :: *′v clause* ⇒ *′st* ⇒ *′st* **and**
add-learned-cls remove-cls :: *′v clause* ⇒ *′st* ⇒ *′st* **and**
update-backtrack-lvl :: *nat* ⇒ *′st* ⇒ *′st* **and**
update-conflicting :: *′v clause option* ⇒ *′st* ⇒ *′st* **and**

init-state :: *′v clauses* ⇒ *′st* **and**
restart-state :: *′st* ⇒ *′st* +

fixes f :: *nat* ⇒ *nat*
assumes f: *unbounded f*
begin

The condition of the differences of cardinality has to be strict. Otherwise, you could be in a strange state, where nothing remains to do, but a restart is done. See the proof of well-foundedness.

inductive *cdcl*$_W$*-merge-with-restart* **where**
restart-step:
(*cdcl*$_W$*-merge-stgy*^^(*card* (*set-mset* (*learned-clss T*)) − *card* (*set-mset* (*learned-clss S*)))) *S T*
⟹ *card* (*set-mset* (*learned-clss T*)) − *card* (*set-mset* (*learned-clss S*)) > *f n*
⟹ *restart T U* ⟹ *cdcl*$_W$*-merge-with-restart* (*S*, *n*) (*U*, *Suc n*) |
restart-full: *full1 cdcl*$_W$*-merge-stgy S T* ⟹ *cdcl*$_W$*-merge-with-restart* (*S*, *n*) (*T*, *Suc n*)

lemma *cdcl*$_W$*-merge-with-restart S T* ⟹ *cdcl*$_W$*-merge-restart*** (*fst S*) (*fst T*)
by (*induction rule*: *cdcl*$_W$*-merge-with-restart.induct*)
(*auto dest*!: *relpowp-imp-rtranclp cdcl*$_W$*-merge-stgy-tranclp-cdcl*$_W$*-merge tranclp-into-rtranclp rtranclp-cdcl*$_W$*-merge-stgy-rtranclp-cdcl*$_W$*-merge rtranclp-cdcl*$_W$*-merge-tranclp-cdcl*$_W$*-merge-restart fw-r-rf cdcl*$_W$*-rf.restart*
simp: *full1-def*)

lemma *cdcl*$_W$*-merge-with-restart-rtranclp-cdcl*$_W$:
cdcl$_W$*-merge-with-restart S T* ⟹ *cdcl*$_W$** (*fst S*) (*fst T*)
by (*induction rule*: *cdcl*$_W$*-merge-with-restart.induct*)
(*auto dest*!: *relpowp-imp-rtranclp rtranclp-cdcl*$_W$*-merge-stgy-rtranclp-cdcl*$_W$ *cdcl*$_W$*.rf cdcl*$_W$*-rf.restart tranclp-into-rtranclp simp*: *full1-def*)

lemma *cdcl*$_W$*-merge-with-restart-increasing-number*:
cdcl$_W$*-merge-with-restart S T* ⟹ *snd T* = *1* + *snd S*
by (*induction rule*: *cdcl*$_W$*-merge-with-restart.induct*) *auto*

lemma *full1 cdcl*$_W$*-merge-stgy S T* ⟹ *cdcl*$_W$*-merge-with-restart* (*S*, *n*) (*T*, *Suc n*)
using *restart-full* **by** *blast*

lemma *cdcl*$_W$*-all-struct-inv-learned-clss-bound*:
assumes *inv*: *cdcl*$_W$*-all-struct-inv S*
shows *set-mset* (*learned-clss S*) ⊆ *simple-clss* (*atms-of-msu* (*init-clss S*))
proof
fix *C*
assume *C*: *C* ∈ *set-mset* (*learned-clss S*)
have *distinct-mset C*
using *C inv* **unfolding** *cdcl*$_W$*-all-struct-inv-def distinct-cdcl*$_W$*-state-def distinct-mset-set-def*
by *auto*
moreover have ¬*tautology C*
using *C inv* **unfolding** *cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-learned-clause-def* **by** *auto*
moreover
have *atms-of C* ⊆ *atms-of-msu* (*learned-clss S*)
using *C* **by** *auto*
then have *atms-of C* ⊆ *atms-of-msu* (*init-clss S*)
using *inv* **unfolding** *cdcl*$_W$*-all-struct-inv-def no-strange-atm-def* **by** *force*
moreover have *finite* (*atms-of-msu* (*init-clss S*))
using *inv* **unfolding** *cdcl*$_W$*-all-struct-inv-def* **by** *auto*
ultimately show *C* ∈ *simple-clss* (*atms-of-msu* (*init-clss S*))
using *distinct-mset-not-tautology-implies-in-simple-clss simple-clss-mono*
by *blast*

qed

lemma *cdcl$_W$-merge-with-restart-init-clss*:
cdcl$_W$-merge-with-restart S T $\Longrightarrow$ *cdcl$_W$-M-level-inv* (*fst S*) $\Longrightarrow$
init-clss (*fst S*) = *init-clss* (*fst T*)
using *cdcl$_W$-merge-with-restart-rtranclp-cdcl$_W$ rtranclp-cdcl$_W$-init-clss* **by** *blast*

lemma
wf {(*T*, *S*). *cdcl$_W$-all-struct-inv* (*fst S*) $\wedge$ *cdcl$_W$-merge-with-restart S T*}
proof (*rule ccontr*)
assume ¬ *?thesis*
then obtain *g* **where**
g: $\bigwedge$*i*. *cdcl$_W$-merge-with-restart* (*g i*) (*g* (*Suc i*)) **and**
inv: $\bigwedge$*i*. *cdcl$_W$-all-struct-inv* (*fst* (*g i*))
unfolding *wf-iff-no-infinite-down-chain* **by** *fast*
{ fix *i*
have *init-clss* (*fst* (*g i*)) = *init-clss* (*fst* (*g 0*))
apply (*induction i*)
apply *simp*
using *g inv* **unfolding** *cdcl$_W$-all-struct-inv-def* **by** (*metis cdcl$_W$-merge-with-restart-init-clss*)
} note *init-g* = *this*
let *?S* = *g 0*
have *finite* (*atms-of-msu* (*init-clss* (*fst ?S*)))
using *inv* **unfolding** *cdcl$_W$-all-struct-inv-def* **by** *auto*
have *snd-g*: $\bigwedge$*i*. *snd* (*g i*) = *i* + *snd* (*g 0*)
apply (*induct-tac i*)
apply *simp*
by (*metis Suc-eq-plus1-left add-Suc cdcl$_W$-merge-with-restart-increasing-number g*)
then have *snd-g-0*: $\bigwedge$*i*. *i* > *0* $\Longrightarrow$ *snd* (*g i*) = *i* + *snd* (*g 0*)
by *blast*
have *unbounded-f-g*: *unbounded* (λ*i*. *f* (*snd* (*g i*)))
using *f* **unfolding** *bounded-def* **by** (*metis add.commute f less-or-eq-imp-le snd-g*
not-bounded-nat-exists-larger not-le le-iff-add)

obtain *k* **where**
f-g-k: *f* (*snd* (*g k*)) > *card* (*simple-clss* (*atms-of-msu* (*init-clss* (*fst ?S*)))) **and**
k > *card* (*simple-clss* (*atms-of-msu* (*init-clss* (*fst ?S*))))
using *not-bounded-nat-exists-larger*[*OF unbounded-f-g*] **by** *blast*

The following does not hold anymore with the non-strict version of cardinality in the definition.

{ fix *i*
assume *no-step cdcl$_W$-merge-stgy* (*fst* (*g i*))
with *g*[*of i*]
have *False*
proof (*induction rule*: *cdcl$_W$-merge-with-restart.induct*)
case (*restart-step T S n*) **note** *H* = *this*(*1*) **and** *c* = *this*(*2*) **and** *n-s* = *this*(*4*)
obtain *S'* **where** *cdcl$_W$-merge-stgy S S'*
using *H c* **by** (*metis gr-implies-not0 relpowp-E2*)
then show *False* **using** *n-s* **by** *auto*
next
case (*restart-full S T*)
then show *False* **unfolding** *full1-def* **by** (*auto dest*: *tranclpD*)
qed
} note *H* = *this*
obtain *m T* **where**

m: $m = card\ (set\text{-}mset\ (learned\text{-}clss\ T)) - card\ (set\text{-}mset\ (learned\text{-}clss\ (fst\ (g\ k))))$ **and**
$m > f\ (snd\ (g\ k))$ **and**
restart T $(fst\ (g\ (k+1)))$ **and**
$cdcl_W$*-merge-stgy*: $(cdcl_W\text{-}merge\text{-}stgy\ \widehat{}\widehat{}\ m)\ (fst\ (g\ k))\ T$
using *g*[*of k*] *H*[*of Suc k*] **by** (*force simp*: $cdcl_W$*-merge-with-restart.simps full1-def*)
have $cdcl_W\text{-}merge\text{-}stgy^{**}\ (fst\ (g\ k))\ T$
using $cdcl_W$*-merge-stgy relpowp-imp-rtranclp* **by** *metis*
then have $cdcl_W$*-all-struct-inv T*
using *inv*[*of k*] *rtranclp-*$cdcl_W$*-all-struct-inv-inv rtranclp-*$cdcl_W$*-merge-stgy-rtranclp-*$cdcl_W$
by *blast*
moreover have $card\ (set\text{-}mset\ (learned\text{-}clss\ T)) - card\ (set\text{-}mset\ (learned\text{-}clss\ (fst\ (g\ k))))$
$> card\ (simple\text{-}clss\ (atms\text{-}of\text{-}msu\ (init\text{-}clss\ (fst\ ?S))))$
unfolding *m*[*symmetric*] **using** ‹$m > f\ (snd\ (g\ k))$› *f-g-k* **by** *linarith*
then have $card\ (set\text{-}mset\ (learned\text{-}clss\ T))$
$> card\ (simple\text{-}clss\ (atms\text{-}of\text{-}msu\ (init\text{-}clss\ (fst\ ?S))))$
by *linarith*
moreover
have $init\text{-}clss\ (fst\ (g\ k)) = init\text{-}clss\ T$
using ‹$cdcl_W\text{-}merge\text{-}stgy^{**}\ (fst\ (g\ k))\ T$› *rtranclp-*$cdcl_W$*-merge-stgy-rtranclp-*$cdcl_W$
rtranclp-$cdcl_W$*-init-clss inv* **unfolding** $cdcl_W$*-all-struct-inv-def* **by** *blast*
then have $init\text{-}clss\ (fst\ ?S) = init\text{-}clss\ T$
using *init-g*[*of k*] **by** *auto*
ultimately show *False*
using $cdcl_W$*-all-struct-inv-learned-clss-bound*
by (*simp add*: ‹*finite* (*atms-of-msu* (*init-clss* (*fst* (*g 0*))))› *simple-clss-finite*
card-mono leD)
qed

lemma $cdcl_W$*-merge-with-restart-distinct-mset-clauses*:
assumes *invR*: $cdcl_W$*-all-struct-inv* (*fst R*) **and**
st: $cdcl_W$*-merge-with-restart R S* **and**
dist: *distinct-mset* (*clauses* (*fst R*)) **and**
R: *trail* (*fst R*) = []
shows *distinct-mset* (*clauses* (*fst S*))
using *assms*(*2*,*1*,*3*,*4*)
proof (*induction*)
case (*restart-full S T*)
then show *?case* **using** *rtranclp-*$cdcl_W$*-merge-stgy-distinct-mset-clauses*[*of S T*] **unfolding** *full1-def*
by (*auto dest*: *tranclp-into-rtranclp*)
next
case (*restart-step T S n U*)
then have *distinct-mset* (*clauses T*)
using *rtranclp-*$cdcl_W$*-merge-stgy-distinct-mset-clauses*[*of S T*] **unfolding** *full1-def*
by (*auto dest*: *relpowp-imp-rtranclp*)
then show *?case* **using** ‹*restart T U*› **by** (*metis clauses-restart distinct-mset-union fstI*
mset-le-exists-conv restart.cases state-eq-clauses)
qed

inductive $cdcl_W$*-with-restart* **where**
restart-step:
$(cdcl_W\text{-}stgy\widehat{}\widehat{}(card\ (set\text{-}mset\ (learned\text{-}clss\ T)) - card\ (set\text{-}mset\ (learned\text{-}clss\ S))))\ S\ T \Longrightarrow$
$card\ (set\text{-}mset\ (learned\text{-}clss\ T)) - card\ (set\text{-}mset\ (learned\text{-}clss\ S)) > f\ n \Longrightarrow$
$restart\ T\ U \Longrightarrow$
$cdcl_W\text{-}with\text{-}restart\ (S,\ n)\ (U,\ Suc\ n)$ |
restart-full: $full1\ cdcl_W\text{-}stgy\ S\ T \Longrightarrow cdcl_W\text{-}with\text{-}restart\ (S,\ n)\ (T,\ Suc\ n)$

lemma *cdcl*$_W$*-with-restart-rtranclp-cdcl*$_W$:
cdcl$_W$*-with-restart S T* $\Longrightarrow$ *cdcl*$_W$** (*fst S*) (*fst T*)
apply (*induction rule*: *cdcl*$_W$*-with-restart.induct*)
by (*auto dest*!: *relpowp-imp-rtranclp tranclp-into-rtranclp fw-r-rf*
cdcl$_W$*-rf.restart rtranclp-cdcl*$_W$*-stgy-rtranclp-cdcl*$_W$ *cdcl*$_W$*-merge-restart-cdcl*$_W$
simp: *full1-def*)

lemma *cdcl*$_W$*-with-restart-increasing-number*:
cdcl$_W$*-with-restart S T* $\Longrightarrow$ *snd T* = *1* + *snd S*
by (*induction rule*: *cdcl*$_W$*-with-restart.induct*) *auto*

lemma *full1 cdcl*$_W$*-stgy S T* $\Longrightarrow$ *cdcl*$_W$*-with-restart* (*S*, *n*) (*T*, *Suc n*)
using *restart-full* **by** *blast*

lemma *cdcl*$_W$*-with-restart-init-clss*:
cdcl$_W$*-with-restart S T* $\Longrightarrow$ *cdcl*$_W$*-M-level-inv* (*fst S*) $\Longrightarrow$ *init-clss* (*fst S*) = *init-clss* (*fst T*)
using *cdcl*$_W$*-with-restart-rtranclp-cdcl*$_W$ *rtranclp-cdcl*$_W$*-init-clss* **by** *blast*

lemma
wf {(*T*, *S*). *cdcl*$_W$*-all-struct-inv* (*fst S*) $\wedge$ *cdcl*$_W$*-with-restart S T*}
proof (*rule ccontr*)
assume $\neg$ *?thesis*
then obtain *g* **where**
g: $\bigwedge$*i*. *cdcl*$_W$*-with-restart* (*g i*) (*g* (*Suc i*)) **and**
inv: $\bigwedge$*i*. *cdcl*$_W$*-all-struct-inv* (*fst* (*g i*))
unfolding *wf-iff-no-infinite-down-chain* **by** *fast*
{ fix *i*
have *init-clss* (*fst* (*g i*)) = *init-clss* (*fst* (*g 0*))
apply (*induction i*)
apply *simp*
using *g inv* **unfolding** *cdcl*$_W$*-all-struct-inv-def* **by** (*metis cdcl*$_W$*-with-restart-init-clss*)
} note *init-g* = *this*
let *?S* = *g 0*
have *finite* (*atms-of-msu* (*init-clss* (*fst ?S*)))
using *inv* **unfolding** *cdcl*$_W$*-all-struct-inv-def* **by** *auto*
have *snd-g*: $\bigwedge$*i*. *snd* (*g i*) = *i* + *snd* (*g 0*)
apply (*induct-tac i*)
apply *simp*
by (*metis Suc-eq-plus1-left add-Suc cdcl*$_W$*-with-restart-increasing-number g*)
then have *snd-g-0*: $\bigwedge$*i*. *i* > *0* $\Longrightarrow$ *snd* (*g i*) = *i* + *snd* (*g 0*)
by *blast*
have *unbounded-f-g*: *unbounded* (λ*i*. *f* (*snd* (*g i*)))
using *f* **unfolding** *bounded-def* **by** (*metis add.commute f less-or-eq-imp-le snd-g*
not-bounded-nat-exists-larger not-le le-iff-add)

obtain *k* **where**
f-g-k: *f* (*snd* (*g k*)) > *card* (*simple-clss* (*atms-of-msu* (*init-clss* (*fst ?S*)))) **and**
k > *card* (*simple-clss* (*atms-of-msu* (*init-clss* (*fst ?S*))))
using *not-bounded-nat-exists-larger*[*OF unbounded-f-g*] **by** *blast*

The following does not hold anymore with the non-strict version of cardinality in the definition.

{ fix *i*
assume *no-step cdcl*$_W$*-stgy* (*fst* (*g i*))
with *g*[*of i*]

have *False*
proof (*induction rule*: $cdcl_W$*-with-restart.induct*)
case (*restart-step T S n*) **note** $H = this(1)$ **and** $c = this(2)$ **and** $n\text{-}s = this(4)$
obtain S' **where** $cdcl_W$*-stgy* S S'
using *H c* **by** (*metis gr-implies-not0 relpowp-E2*)
then show *False* **using** *n-s* **by** *auto*
next
case (*restart-full S T*)
then show *False* **unfolding** *full1-def* **by** (*auto dest*: *tranclpD*)
qed
} **note** $H = this$
obtain *m T* **where**
m: $m = card\ (set\text{-}mset\ (learned\text{-}clss\ T)) - card\ (set\text{-}mset\ (learned\text{-}clss\ (fst\ (g\ k))))$ **and**
$m > f\ (snd\ (g\ k))$ **and**
restart T $(fst\ (g\ (k+1)))$ **and**
$cdcl_W$*-merge-stgy*: $(cdcl_W\text{-}stgy\ \widehat{\ }\widehat{\ }\ m)\ (fst\ (g\ k))\ T$
using $g[of\ k]$ $H[of\ Suc\ k]$ **by** (*force simp*: $cdcl_W$*-with-restart.simps full1-def*)
have $cdcl_W\text{-}stgy^{**}\ (fst\ (g\ k))\ T$
using $cdcl_W$*-merge-stgy relpowp-imp-rtranclp* **by** *metis*
then have $cdcl_W$*-all-struct-inv T*
using $inv[of\ k]$ *rtranclp-*$cdcl_W$*-all-struct-inv-inv rtranclp-*$cdcl_W$*-stgy-rtranclp-*$cdcl_W$ **by** *blast*
moreover have $card\ (set\text{-}mset\ (learned\text{-}clss\ T)) - card\ (set\text{-}mset\ (learned\text{-}clss\ (fst\ (g\ k))))$
$> card\ (simple\text{-}clss\ (atms\text{-}of\text{-}msu\ (init\text{-}clss\ (fst\ ?S))))$
unfolding $m[symmetric]$ **using** ‹$m > f\ (snd\ (g\ k))$› *f-g-k* **by** *linarith*
then have $card\ (set\text{-}mset\ (learned\text{-}clss\ T))$
$> card\ (simple\text{-}clss\ (atms\text{-}of\text{-}msu\ (init\text{-}clss\ (fst\ ?S))))$
by *linarith*
moreover
have $init\text{-}clss\ (fst\ (g\ k)) = init\text{-}clss\ T$
using ‹$cdcl_W\text{-}stgy^{**}\ (fst\ (g\ k))\ T$› *rtranclp-*$cdcl_W$*-stgy-rtranclp-*$cdcl_W$ *rtranclp-*$cdcl_W$*-init-clss*
inv **unfolding** $cdcl_W$*-all-struct-inv-def*
by *blast*
then have $init\text{-}clss\ (fst\ ?S) = init\text{-}clss\ T$
using $init\text{-}g[of\ k]$ **by** *auto*
ultimately show *False*
using $cdcl_W$*-all-struct-inv-learned-clss-bound*
by (*simp add*: ‹$finite\ (atms\text{-}of\text{-}msu\ (init\text{-}clss\ (fst\ (g\ 0))))$› *simple-clss-finite*
card-mono leD)
qed

lemma $cdcl_W$*-with-restart-distinct-mset-clauses*:
assumes *invR*: $cdcl_W$*-all-struct-inv* $(fst\ R)$ **and**
st: $cdcl_W$*-with-restart R S* **and**
dist: *distinct-mset* $(clauses\ (fst\ R))$ **and**
R: $trail\ (fst\ R) = []$
shows *distinct-mset* $(clauses\ (fst\ S))$
using $assms(2,1,3,4)$
proof (*induction*)
case (*restart-full S T*)
then show *?case* **using** *rtranclp-*$cdcl_W$*-stgy-distinct-mset-clauses*$[of\ S\ T]$ **unfolding** *full1-def*
by (*auto dest*: *tranclp-into-rtranclp*)
next
case (*restart-step T S n U*)
then have *distinct-mset* $(clauses\ T)$ **using** *rtranclp-*$cdcl_W$*-stgy-distinct-mset-clauses*$[of\ S\ T]$
unfolding *full1-def* **by** (*auto dest*: *relpowp-imp-rtranclp*)

then show *?case* **using** ‹*restart T U*› **by** (*metis clauses-restart distinct-mset-union fstI mset-le-exists-conv restart.cases state-eq-clauses*)
qed
end

locale *luby-sequence* =
fixes *ur* :: *nat*
assumes *ur* > *0*
begin

lemma *exists-luby-decomp*:
fixes *i* ::*nat*
shows $\exists k::nat.\ (2\ \hat{}\ (k - 1) \leq i \wedge i < 2\ \hat{}\ k - 1) \vee i = 2\ \hat{}\ k - 1$
proof (*induction i*)
case *0*
then show *?case*
by (*rule exI*[*of - 0*], *simp*)
next
case (*Suc n*)
then obtain *k* **where** $2\ \hat{}\ (k - 1) \leq n \wedge n < 2\ \hat{}\ k - 1 \vee n = 2\ \hat{}\ k - 1$
by *blast*
then consider
(*st-interv*) $2\ \hat{}\ (k - 1) \leq n$ **and** $n \leq 2\ \hat{}\ k - 2$
| (*end-interv*) $2\ \hat{}\ (k - 1) \leq n$ **and** $n = 2\ \hat{}\ k - 2$
| (*pow2*) $n = 2\ \hat{}\ k - 1$
by *linarith*
then show *?case*
proof *cases*
case *st-interv*
then show *?thesis* **apply** − **apply** (*rule exI*[*of - k*])
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-diff-Suc Suc-lessI* ‹$2\ \hat{}\ (k - 1) \leq n \wedge n < 2\ \hat{}\ k - 1 \vee n = 2\ \hat{}\ k - 1$› *diff-self-eq-0 dual-order.trans le-SucI le-imp-less-Suc numeral-2-eq-2 one-le-numeral one-le-power zero-less-numeral zero-less-power*)
next
case *end-interv*
then show *?thesis* **apply** − **apply** (*rule exI*[*of - k*]) **by** *auto*
next
case *pow2*
then show *?thesis* **apply** − **apply** (*rule exI*[*of - k+1*]) **by** *auto*
qed
qed

Luby sequences are defined by:

- $2^k - 1$, if $i = (2::'a)^k - (1::'a)$
- *luby-sequence-core* $(i - 2^{k - 1} + 1)$, if $(2::'a)^{k - 1} \leq i$ and $i \leq (2::'a)^k - (1::'a)$

Then the sequence is then scaled by a constant unit run (called *ur* here), strictly positive.

function *luby-sequence-core* :: *nat* ⇒ *nat* **where**
luby-sequence-core i =
(*if* $\exists k.\ i = 2\hat{}k - 1$
then $2\hat{}((SOME\ k.\ i = 2\hat{}k - 1) - 1)$
else luby-sequence-core $(i - 2\hat{}((SOME\ k.\ 2\hat{}(k-1) \leq i \wedge i < 2\hat{}k - 1) - 1) + 1))$

by *auto*
termination
proof (*relation less-than*, *goal-cases*)
  case *1*
  then show *?case* **by** *auto*
next
  case (*2 i*)
  let *?k* = (*SOME k*. $2 \hat{} (k - 1) \leq i \wedge i < 2 \hat{} k - 1$)
  have $2 \hat{} (?k - 1) \leq i \wedge i < 2 \hat{} ?k - 1$
    apply (*rule someI-ex*)
    using *2 exists-luby-decomp* **by** *blast*
  then show *?case*

    proof −
      have $\forall n\ na.\ \neg (1::nat) \leq n \vee 1 \leq n \hat{} na$
        by (*meson one-le-power*)
      then have *f1*: $(1::nat) \leq 2 \hat{} (?k - 1)$
        using *one-le-numeral* **by** *blast*
      have *f2*: $i - 2 \hat{} (?k - 1) + 2 \hat{} (?k - 1) = i$
        using ‹$2 \hat{} (?k - 1) \leq i \wedge i < 2 \hat{} ?k - 1$› *le-add-diff-inverse2* **by** *blast*
      have *f3*: $2 \hat{} ?k - 1 \neq Suc\ 0$
        using *f1* ‹$2 \hat{} (?k - 1) \leq i \wedge i < 2 \hat{} ?k - 1$› **by** *linarith*
      have $2 \hat{} ?k - (1::nat) \neq 0$
        using ‹$2 \hat{} (?k - 1) \leq i \wedge i < 2 \hat{} ?k - 1$› *gr-implies-not0* **by** *blast*
      then have *f4*: $2 \hat{} ?k \neq (1::nat)$
        by *linarith*
      have *f5*: $\forall n\ na.$ *if* $na = 0$ *then* $(n::nat) \hat{} na = 1$ *else* $n \hat{} na = n * n \hat{} (na - 1)$
        by (*simp add*: *power-eq-if*)
      then have $?k \neq 0$
        using *f4* **by** *meson*
      then have $2 \hat{} (?k - 1) \neq Suc\ 0$
        using *f5 f3* **by** *presburger*
      then have $Suc\ 0 < 2 \hat{} (?k - 1)$
        using *f1* **by** *linarith*
      then show *?thesis*
        using *f2 less-than-iff* **by** *presburger*
    qed
qed

declare *luby-sequence-core.simps*[*simp del*]

lemma *two-pover-n-eq-two-power-n′-eq*:
  assumes *H*: $(2::nat) \hat{} (k::nat) - 1 = 2 \hat{} k' - 1$
  shows $k' = k$
proof −
  have $(2::nat) \hat{} (k::nat) = 2 \hat{} k'$
    using *H* **by** (*metis One-nat-def Suc-pred zero-less-numeral zero-less-power*)
  then show *?thesis* **by** *simp*
qed

lemma *luby-sequence-core-two-power-minus-one*:
  luby-sequence-core $(2\hat{}k - 1) = 2\hat{}(k-1)$ (**is** *?L* = *?K*)
proof −
  have *decomp*: $\exists ka.\ 2 \hat{} k - 1 = 2 \hat{} ka - 1$
    by *auto*

have *?L* = $2\hat{}((SOME\ k'.\ (2::nat)\hat{}k - 1 = 2\hat{}k' - 1) - 1)$
  apply (*subst luby-sequence-core.simps*, *subst decomp*)
  by *simp*
moreover have $(SOME\ k'.\ (2::nat)\hat{}k - 1 = 2\hat{}k' - 1) = k$
  apply (*rule some-equality*)
    apply *simp*
    using *two-pover-n-eq-two-power-n'-eq* **by** *blast*
ultimately show *?thesis* **by** *presburger*
qed

lemma *different-luby-decomposition-false*:
  assumes
    H: $2 \hat{} (k - Suc\ 0) \leq i$ **and**
    k': $i < 2 \hat{} k' - Suc\ 0$ **and**
    k-k': $k > k'$
  shows *False*
proof −
  have $2 \hat{} k' - Suc\ 0 < 2 \hat{} (k - Suc\ 0)$
    using *k-k' less-eq-Suc-le* **by** *auto*
  then show *?thesis*
    using *H k'* **by** *linarith*
qed

lemma *luby-sequence-core-not-two-power-minus-one*:
  assumes
    k-i: $2 \hat{} (k - 1) \leq i$ **and**
    i-k: $i < 2\hat{}k - 1$
  shows *luby-sequence-core i* = *luby-sequence-core* $(i - 2 \hat{} (k - 1) + 1)$
proof −
  have *H*: $\neg (\exists ka.\ i = 2 \hat{} ka - 1)$
    proof (*rule ccontr*)
      assume ¬ *?thesis*
      then obtain *k'::nat* **where** *k'*: $i = 2 \hat{} k' - 1$ **by** *blast*
      have $(2::nat) \hat{} k' - 1 < 2 \hat{} k - 1$
        using *i-k* **unfolding** *k'* **.**
      then have $(2::nat) \hat{} k' < 2 \hat{} k$
        by *linarith*
      then have $k' < k$
        by *simp*
      have $2 \hat{} (k - 1) \leq 2 \hat{} k' - (1::nat)$
        using *k-i* **unfolding** *k'* **.**
      then have $(2::nat) \hat{} (k-1) < 2 \hat{} k'$
        by (*metis Suc-diff-1 not-le not-less-eq zero-less-numeral zero-less-power*)
      then have $k-1 < k'$
        by *simp*

      show *False* **using** ‹$k' < k$› ‹$k-1 < k'$› **by** *linarith*
    qed
  have $\bigwedge k\ k'.\ 2 \hat{} (k - Suc\ 0) \leq i \Longrightarrow i < 2 \hat{} k - Suc\ 0 \Longrightarrow 2 \hat{} (k' - Suc\ 0) \leq i \Longrightarrow$
    $i < 2 \hat{} k' - Suc\ 0 \Longrightarrow k = k'$
    by (*meson different-luby-decomposition-false linorder-neqE-nat*)
  then have *k*: $(SOME\ k.\ 2 \hat{} (k - Suc\ 0) \leq i \wedge i < 2 \hat{} k - Suc\ 0) = k$
    using *k-i i-k* **by** *auto*
  show *?thesis*
    apply (*subst luby-sequence-core.simps*[*of i*], *subst H*)

```
    by (simp add: k)
qed

lemma unbounded-luby-sequence-core: unbounded luby-sequence-core
  unfolding bounded-def
proof
  assume ∃ b. ∀ n. luby-sequence-core n ≤ b
  then obtain b where b: ⋀n. luby-sequence-core n ≤ b
    by metis
  have luby-sequence-core (2^(b+1) − 1) = 2^b
    using luby-sequence-core-two-power-minus-one[of b+1] by simp
  moreover have (2::nat)^b > b
    by (induction b) auto
  ultimately show False using b[of 2^(b+1) − 1] by linarith
qed

abbreviation luby-sequence :: nat ⇒ nat where
luby-sequence n ≡  ur ∗ luby-sequence-core n

lemma bounded-luby-sequence: unbounded luby-sequence
  using bounded-const-product[of ur] luby-sequence-axioms
  luby-sequence-def unbounded-luby-sequence-core by blast

lemma luby-sequence-core-0: luby-sequence-core 0 = 1
proof −
  have 0: (0::nat) = 2^0−1
    by auto
  show ?thesis
    by (subst 0, subst luby-sequence-core-two-power-minus-one) simp
qed

lemma luby-sequence-core n ≥ 1
proof (induction n rule: nat-less-induct-case)
  case 0
  then show ?case by (simp add: luby-sequence-core-0)
next
  case (Suc n) note IH = this

  consider
      (interv) k where 2 ^ (k − 1) ≤ Suc n and Suc n < 2 ^ k − 1
    | (pow2)  k where Suc n = 2 ^ k − Suc 0
    using exists-luby-decomp[of Suc n] by auto

  then show ?case
     proof cases
       case pow2
       show ?thesis
         using luby-sequence-core-two-power-minus-one pow2 by auto
     next
       case interv
       have n: Suc n − 2 ^ (k − 1) + 1 < Suc n
         by (metis Suc-1 Suc-eq-plus1 add.commute add-diff-cancel-left′ add-less-mono1 gr0I
           interv(1) interv(2) le-add-diff-inverse2 less-Suc-eq not-le power-0 power-one-right
           power-strict-increasing-iff)
       show ?thesis
```

apply (*subst luby-sequence-core-not-two-power-minus-one*[*OF interv*])
using *IH n* **by** *auto*
qed
qed
end

locale *luby-sequence-restart* =
luby-sequence ur +
$cdcl_W$ *trail init-clss learned-clss backtrack-lvl conflicting cons-trail tl-trail*
add-init-cls
add-learned-cls remove-cls update-backtrack-lvl update-conflicting init-state
restart-state
for
ur :: *nat* **and**
trail :: *'st* ⇒ (*'v*, *nat*, *'v clause*) *ann-literals* **and**
init-clss :: *'st* ⇒ *'v clauses* **and**
learned-clss :: *'st* ⇒ *'v clauses* **and**
backtrack-lvl :: *'st* ⇒ *nat* **and**
conflicting :: *'st* ⇒*'v clause option* **and**
cons-trail :: (*'v*, *nat*, *'v clause*) *ann-literal* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒ *'st* **and**
add-init-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
add-learned-cls remove-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
update-backtrack-lvl :: *nat* ⇒ *'st* ⇒ *'st* **and**
update-conflicting :: *'v clause option* ⇒ *'st* ⇒ *'st* **and**

init-state :: *'v clauses* ⇒ *'st* **and**
restart-state :: *'st* ⇒ *'st*
begin

sublocale $cdcl_W$*-restart* - - - - - - - - - - - - - - *luby-sequence*
apply *unfold-locales*
using *bounded-luby-sequence* **by** *blast*

end

end
theory *CDCL-W-Incremental*
imports *CDCL-W-Termination*
begin

8 Incremental SAT solving

context $cdcl_W$
begin

This invariant holds all the invariant related to the strategy. See the structural invariant in $cdcl_W$*-all-struct-inv*

definition $cdcl_W$*-stgy-invariant* **where**
$cdcl_W$*-stgy-invariant S* ⟷
conflict-is-false-with-level S
∧ *no-clause-is-false S*
∧ *no-smaller-confl S*
∧ *no-clause-is-false S*

lemma *cdcl*$_W$*-stgy-cdcl*$_W$*-stgy-invariant*:
assumes
cdcl$_W$: *cdcl*$_W$*-stgy S T* **and**
inv-s: *cdcl*$_W$*-stgy-invariant S* **and**
inv: *cdcl*$_W$*-all-struct-inv S*
shows
cdcl$_W$*-stgy-invariant T*
unfolding *cdcl*$_W$*-stgy-invariant-def cdcl*$_W$*-all-struct-inv-def* **apply** *standard*
apply (*rule cdcl*$_W$*-stgy-ex-lit-of-max-level*[*of S*])
using *assms* **unfolding** *cdcl*$_W$*-stgy-invariant-def cdcl*$_W$*-all-struct-inv-def* **apply** *auto*[*7*]
apply *standard*
using *cdcl*$_W$ *cdcl*$_W$*-stgy-not-non-negated-init-clss* **apply** *blast*
apply *standard*
apply (*rule cdcl*$_W$*-stgy-no-smaller-confl-inv*)
using *assms* **unfolding** *cdcl*$_W$*-stgy-invariant-def cdcl*$_W$*-all-struct-inv-def* **apply** *auto*[*4*]
using *cdcl*$_W$ *cdcl*$_W$*-stgy-not-non-negated-init-clss* **by** *auto*

lemma *rtranclp-cdcl*$_W$*-stgy-cdcl*$_W$*-stgy-invariant*:
assumes
cdcl$_W$: *cdcl*$_W$*-stgy*** *S T* **and**
inv-s: *cdcl*$_W$*-stgy-invariant S* **and**
inv: *cdcl*$_W$*-all-struct-inv S*
shows
cdcl$_W$*-stgy-invariant T*
using *assms* **apply** (*induction*)
apply *simp*
using *cdcl*$_W$*-stgy-cdcl*$_W$*-stgy-invariant rtranclp-cdcl*$_W$*-all-struct-inv-inv*
rtranclp-cdcl$_W$*-stgy-rtranclp-cdcl*$_W$ **by** *blast*

abbreviation *decr-bt-lvl* **where**
decr-bt-lvl S ≡ *update-backtrack-lvl* (*backtrack-lvl S* − *1*) *S*

When we add a new clause, we reduce the trail until we get to tho first literal included in C. Then we can mark the conflict.

fun *cut-trail-wrt-clause* **where**
cut-trail-wrt-clause C [] *S* = *S* |
cut-trail-wrt-clause C (*Marked L* - # *M*) *S* =
(*if* −*L* ∈# *C then S*
else cut-trail-wrt-clause C M (*decr-bt-lvl* (*tl-trail S*))) |
cut-trail-wrt-clause C (*Propagated L* - # *M*) *S* =
(*if* −*L* ∈# *C then S*
else cut-trail-wrt-clause C M (*tl-trail S*))

definition *add-new-clause-and-update* :: *'v literal multiset* ⇒ *'st* ⇒ *'st* **where**
add-new-clause-and-update C S =
(*if trail S* ⊨*as CNot C*
then update-conflicting (*Some C*) (*add-init-cls C* (*cut-trail-wrt-clause C* (*trail S*) *S*))
else add-init-cls C S)

thm *cut-trail-wrt-clause.induct*
lemma *init-clss-cut-trail-wrt-clause*[*simp*]:
init-clss (*cut-trail-wrt-clause C M S*) = *init-clss S*
by (*induction rule*: *cut-trail-wrt-clause.induct*) *auto*

lemma *learned-clss-cut-trail-wrt-clause*[*simp*]:

learned-clss (*cut-trail-wrt-clause C M S*) = *learned-clss S*
by (*induction rule*: *cut-trail-wrt-clause.induct*) *auto*

lemma *conflicting-clss-cut-trail-wrt-clause*[*simp*]:
conflicting (*cut-trail-wrt-clause C M S*) = *conflicting S*
by (*induction rule*: *cut-trail-wrt-clause.induct*) *auto*

lemma *trail-cut-trail-wrt-clause*:
$\exists$ *M*. *trail S* = *M* @ *trail* (*cut-trail-wrt-clause C* (*trail S*) *S*)
proof (*induction trail S arbitrary*:*S rule*: *ann-literal-list-induct*)
case *nil*
then show *?case* **by** *simp*
next
case (*marked L l M*) **note** *IH* = *this*(*1*)[*of decr-bt-lvl* (*tl-trail S*)] **and** *M* = *this*(*2*)[*symmetric*]
then show *?case* **using** *Cons-eq-appendI* **by** *fastforce+*
next
case (*proped L l M*) **note** *IH* = *this*(*1*)[*of tl-trail S*] **and** *M* = *this*(*2*)[*symmetric*]
then show *?case* **using** *Cons-eq-appendI* **by** *fastforce+*
qed

lemma *n-dup-no-dup-trail-cut-trail-wrt-clause*[*simp*]:
assumes *n-d*: *no-dup* (*trail T*)
shows *no-dup* (*trail* (*cut-trail-wrt-clause C* (*trail T*) *T*))
proof −
obtain *M* **where**
M: *trail T* = *M* @ *trail* (*cut-trail-wrt-clause C* (*trail T*) *T*)
using *trail-cut-trail-wrt-clause*[*of T C*] **by** *auto*
show *?thesis*
using *n-d* **unfolding** *arg-cong*[*OF M*, *of no-dup*] **by** *auto*
qed

lemma *cut-trail-wrt-clause-backtrack-lvl-length-marked*:
assumes
backtrack-lvl T = *length* (*get-all-levels-of-marked* (*trail T*))
shows
backtrack-lvl (*cut-trail-wrt-clause C* (*trail T*) *T*) =
length (*get-all-levels-of-marked* (*trail* (*cut-trail-wrt-clause C* (*trail T*) *T*)))
using *assms*
proof (*induction trail T arbitrary*:*T rule*: *ann-literal-list-induct*)
case *nil*
then show *?case* **by** *simp*
next
case (*marked L l M*) **note** *IH* = *this*(*1*)[*of decr-bt-lvl* (*tl-trail T*)] **and** *M* = *this*(*2*)[*symmetric*]
and *bt* = *this*(*3*)
then show *?case* **by** *auto*
next
case (*proped L l M*) **note** *IH* = *this*(*1*)[*of tl-trail T*] **and** *M* = *this*(*2*)[*symmetric*] **and** *bt* = *this*(*3*)
then show *?case* **by** *auto*
qed

lemma *cut-trail-wrt-clause-get-all-levels-of-marked*:
assumes *get-all-levels-of-marked* (*trail T*) = *rev* [*Suc 0*..<
Suc (*length* (*get-all-levels-of-marked* (*trail T*)))]
shows
get-all-levels-of-marked (*trail* ((*cut-trail-wrt-clause C* (*trail T*) *T*))) = *rev* [*Suc 0*..<

Suc (*length* (*get-all-levels-of-marked* (*trail* ((*cut-trail-wrt-clause* *C* (*trail* *T*) *T*)))))]
using *assms*
proof (*induction trail T arbitrary*: *T rule*: *ann-literal-list-induct*)
case *nil*
then show *?case* **by** *simp*
next
case (*marked L l M*) **note** *IH* = *this*(*1*)[*of decr-bt-lvl* (*tl-trail T*)] **and** *M* = *this*(*2*)[*symmetric*]
and *bt* = *this*(*3*)
then show *?case* **by** (*cases count C L* = *0*) *auto*
next
case (*proped L l M*) **note** *IH* = *this*(*1*)[*of tl-trail T*] **and** *M* = *this*(*2*)[*symmetric*] **and** *bt* = *this*(*3*)
then show *?case* **by** (*cases count C L* = *0*) *auto*
qed

lemma *cut-trail-wrt-clause-CNot-trail*:
assumes *trail T* $\models$*as CNot C*
shows
(*trail* ((*cut-trail-wrt-clause C* (*trail T*) *T*))) $\models$*as CNot C*
using *assms*
proof (*induction trail T arbitrary*: *T rule*: *ann-literal-list-induct*)
case *nil*
then show *?case* **by** *simp*
next
case (*marked L l M*) **note** *IH* = *this*(*1*)[*of decr-bt-lvl* (*tl-trail T*)] **and** *M* = *this*(*2*)[*symmetric*]
and *bt* = *this*(*3*)
show *?case*
proof (*cases count C* ($-L$) = *0*)
case *False*
then show *?thesis*
using *IH M bt* **by** (*auto simp*: *true-annots-true-cls*)
next
case *True*
obtain *mma* :: *'v literal multiset* **where**
f6: (*mma* $\in$ $\{\{\#- l\#\} \mid l.\ l \in\# C\}$ $\longrightarrow$ *M* $\models$*a mma*) $\longrightarrow$ *M* $\models$*as* $\{\{\#- l\#\} \mid l.\ l \in\# C\}$
using *true-annots-def* **by** *moura*
have *mma* $\in$ $\{\{\#- l\#\} \mid l.\ l \in\# C\}$ $\longrightarrow$ *trail T* $\models$*a mma*
using *CNot-def M bt* **by** (*metis* (*no-types*) *true-annots-def*)
then have *M* $\models$*as* $\{\{\#- l\#\} \mid l.\ l \in\# C\}$
using *f6 True M bt* **by** *force*
then show *?thesis*
using *IH true-annots-true-cls M* **by** (*auto simp*: *CNot-def*)
qed
next
case (*proped L l M*) **note** *IH* = *this*(*1*)[*of tl-trail T*] **and** *M* = *this*(*2*)[*symmetric*] **and** *bt* = *this*(*3*)
show *?case*
proof (*cases count C* ($-L$) = *0*)
case *False*
then show *?thesis*
using *IH M bt* **by** (*auto simp*: *true-annots-true-cls*)
next
case *True*
obtain *mma* :: *'v literal multiset* **where**
f6: (*mma* $\in$ $\{\{\#- l\#\} \mid l.\ l \in\# C\}$ $\longrightarrow$ *M* $\models$*a mma*) $\longrightarrow$ *M* $\models$*as* $\{\{\#- l\#\} \mid l.\ l \in\# C\}$
using *true-annots-def* **by** *moura*
have *mma* $\in$ $\{\{\#- l\#\} \mid l.\ l \in\# C\}$ $\longrightarrow$ *trail T* $\models$*a mma*

using *CNot-def M bt* **by** (*metis* (*no-types*) *true-annots-def*)
then have $M \models as \ \{\{\#- \ l\#\} \ |l. \ l \in\# \ C\}$
using *f6 True M bt* **by** *force*
then show *?thesis*
using *IH true-annots-true-cls M* **by** (*auto simp*: *CNot-def*)
qed
qed

lemma *cut-trail-wrt-clause-hd-trail-in-or-empty-trail*:
$((\forall L \in\# C. \ -L \notin \textit{lits-of} \ (\textit{trail} \ T)) \wedge \textit{trail} \ (\textit{cut-trail-wrt-clause} \ C \ (\textit{trail} \ T) \ T) = [])$
$\vee \ (-\textit{lit-of} \ (\textit{hd} \ (\textit{trail} \ (\textit{cut-trail-wrt-clause} \ C \ (\textit{trail} \ T) \ T))) \in\# \ C$
$\wedge \ \textit{length} \ (\textit{trail} \ (\textit{cut-trail-wrt-clause} \ C \ (\textit{trail} \ T) \ T)) \geq 1)$
using *assms*
proof (*induction trail T arbitrary*: *T rule*: *ann-literal-list-induct*)
case *nil*
then show *?case* **by** *simp*
next
case (*marked L l M*) **note** *IH* = *this*(*1*)[*of decr-bt-lvl* (*tl-trail T*)] **and** *M* = *this*(*2*)[*symmetric*]
then show *?case* **by** *simp force*
next
case (*proped L l M*) **note** *IH* = *this*(*1*)[*of tl-trail T*] **and** *M* = *this*(*2*)[*symmetric*]
then show *?case* **by** *simp force*
qed

We can fully run $cdcl_W$*-s* or add a clause. Remark that we use $cdcl_W$*-s* to avoid an explicit *skip*, *resolve*, and *backtrack* normalisation to get rid of the conflict C if possible.

inductive *incremental-*$cdcl_W$ :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* **where**
add-confl:
trail S ⊨*asm init-clss S* ⟹ *distinct-mset C* ⟹ *conflicting S* = *None* ⟹
trail S ⊨*as CNot C* ⟹
full $cdcl_W$*-stgy*
(*update-conflicting* (*Some C*) (*add-init-cls C* (*cut-trail-wrt-clause C* (*trail S*) *S*))) *T* ⟹
incremental-$cdcl_W$ *S T* |
add-no-confl:
trail S ⊨*asm init-clss S* ⟹ *distinct-mset C* ⟹ *conflicting S* = *None* ⟹
¬*trail S* ⊨*as CNot C* ⟹
full $cdcl_W$*-stgy* (*add-init-cls C S*) *T* ⟹
incremental-$cdcl_W$ *S T*

inductive *add-learned-clss* :: *'st* ⇒ *'v clauses* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
add-learned-clss-nil: *add-learned-clss S* {#} *S* |
add-learned-clss-plus:
add-learned-clss S A T ⟹ *add-learned-clss S* ({#*x*#} + *A*) (*add-learned-cls x T*)
declare *add-learned-clss.intros*[*intro*]

lemma *Ex-add-learned-clss*:
∃ *T*. *add-learned-clss S A T*
by (*induction A arbitrary*: *S rule*: *multiset-induct*) (*auto simp*: *union-commute*[*of* - {#-#}])

lemma *add-learned-clss-trail*:
assumes *add-learned-clss S U T* **and** *no-dup* (*trail S*)
shows *trail T* = *trail S*
using *assms* **by** (*induction rule*: *add-learned-clss.induct*) (*simp-all add*: *ac-simps*)

lemma *add-learned-clss-learned-clss*:

assumes *add-learned-clss S U T* **and** *no-dup* (*trail S*)
shows *learned-clss T* = *U* + *learned-clss S*
using *assms* **by** (*induction rule*: *add-learned-clss.induct*)
(*auto simp*: *ac-simps dest*: *add-learned-clss-trail*)

lemma *add-learned-clss-init-clss*:
assumes *add-learned-clss S U T* **and** *no-dup* (*trail S*)
shows *init-clss T* = *init-clss S*
using *assms* **by** (*induction rule*: *add-learned-clss.induct*)
(*auto simp*: *ac-simps dest*: *add-learned-clss-trail*)

lemma *add-learned-clss-conflicting*:
assumes *add-learned-clss S U T* **and** *no-dup* (*trail S*)
shows *conflicting T* = *conflicting S*
using *assms* **by** (*induction rule*: *add-learned-clss.induct*)
(*auto simp*: *ac-simps dest*: *add-learned-clss-trail*)

lemma *add-learned-clss-backtrack-lvl*:
assumes *add-learned-clss S U T* **and** *no-dup* (*trail S*)
shows *backtrack-lvl T* = *backtrack-lvl S*
using *assms* **by** (*induction rule*: *add-learned-clss.induct*)
(*auto simp*: *ac-simps dest*: *add-learned-clss-trail*)

lemma *add-learned-clss-init-state-mempty*[*dest*!]:
add-learned-clss (*init-state N*) {#} *T* ⟹ *T* = *init-state N*
by (*cases rule*: *add-learned-clss.cases*) (*auto simp*: *add-learned-clss.cases*)

For multiset larger that 1 element, there is no way to know in which order the clauses are added. But contrary to a definition *fold-mset*, there is an element.

lemma *add-learned-clss-init-state-single*[*dest*!]:
add-learned-clss (*init-state N*) {#*C*#} *T* ⟹ *T* = *add-learned-cls C* (*init-state N*)
by (*induction* {#*C*#} *T rule*: *add-learned-clss.induct*)
(*auto simp*: *add-learned-clss.cases ac-simps union-is-single split*: *split-if-asm*)

thm *rtranclp-cdcl*$_W$*-stgy-no-smaller-confl-inv cdcl*$_W$*-stgy-final-state-conclusive*
lemma *cdcl*$_W$*-all-struct-inv-add-new-clause-and-update-cdcl*$_W$*-all-struct-inv*:
assumes
inv-T: *cdcl*$_W$*-all-struct-inv T* **and**
tr-T-N[*simp*]: *trail T* ⊨*asm N* **and**
tr-C[*simp*]: *trail T* ⊨*as CNot C* **and**
[*simp*]: *distinct-mset C*
shows *cdcl*$_W$*-all-struct-inv* (*add-new-clause-and-update C T*) (**is** *cdcl*$_W$*-all-struct-inv ?T'*)
proof −
let *?T* = *update-conflicting* (*Some C*) (*add-init-cls C* (*cut-trail-wrt-clause C* (*trail T*) *T*))
obtain *M* **where**
M: *trail T* = *M* @ *trail* (*cut-trail-wrt-clause C* (*trail T*) *T*)
using *trail-cut-trail-wrt-clause*[*of T C*] **by** *blast*
have *H*[*dest*]: ⋀*x*. *x* ∈ *lits-of* (*trail* (*cut-trail-wrt-clause C* (*trail T*) *T*)) ⟹
x ∈ *lits-of* (*trail T*)
using *inv-T arg-cong*[*OF M*, *of lits-of*] **by** *auto*
have *H'*[*dest*]: ⋀*x*. *x* ∈ *set* (*trail* (*cut-trail-wrt-clause C* (*trail T*) *T*)) ⟹ *x* ∈ *set* (*trail T*)
using *inv-T arg-cong*[*OF M*, *of set*] **by** *auto*

have *H-proped*:⋀*x*. *x* ∈ *set* (*get-all-mark-of-propagated* (*trail* (*cut-trail-wrt-clause C* (*trail T*) *T*))) ⟹ *x* ∈ *set* (*get-all-mark-of-propagated* (*trail T*))

using *inv-T arg-cong*[*OF M*, *of get-all-mark-of-propagated*] **by** *auto*

have [*simp*]: *no-strange-atm ?T*
using *inv-T* **unfolding** $cdcl_W$*-all-struct-inv-def no-strange-atm-def add-new-clause-and-update-def*
$cdcl_W$*-M-level-inv-def*
by (*auto dest*!: *H H′*)

have *M-lev*: $cdcl_W$*-M-level-inv T*
using *inv-T* **unfolding** $cdcl_W$*-all-struct-inv-def* **by** *blast*
then have *no-dup* (*M* @ *trail* (*cut-trail-wrt-clause C* (*trail T*) *T*))
unfolding $cdcl_W$*-M-level-inv-def* **unfolding** *M*[*symmetric*] **by** *auto*
then have [*simp*]: *no-dup* (*trail* (*cut-trail-wrt-clause C* (*trail T*) *T*))
by *auto*

have *consistent-interp* (*lits-of* (*M* @ *trail* (*cut-trail-wrt-clause C* (*trail T*) *T*)))
using *M-lev* **unfolding** $cdcl_W$*-M-level-inv-def* **unfolding** *M*[*symmetric*] **by** *auto*
then have [*simp*]: *consistent-interp* (*lits-of* (*trail* (*cut-trail-wrt-clause C* (*trail T*) *T*)))
unfolding *consistent-interp-def* **by** *auto*

have [*simp*]: $cdcl_W$*-M-level-inv ?T*

using *M-lev cut-trail-wrt-clause-get-all-levels-of-marked*[*of T C*]
unfolding $cdcl_W$*-M-level-inv-def* **by** (*auto dest*: *H H′*
simp: *M-lev* $cdcl_W$*-M-level-inv-def cut-trail-wrt-clause-backtrack-lvl-length-marked*)

have [*simp*]: $\bigwedge s.\ s \in\#$ *learned-clss T* $\Longrightarrow \neg$*tautology s*
using *inv-T* **unfolding** $cdcl_W$*-all-struct-inv-def* **by** *auto*

have *distinct-*$cdcl_W$*-state T*
using *inv-T* **unfolding** $cdcl_W$*-all-struct-inv-def* **by** *auto*
then have [*simp*]: *distinct-*$cdcl_W$*-state ?T*
unfolding *distinct-*$cdcl_W$*-state-def* **by** *auto*

have $cdcl_W$*-conflicting T*
using *inv-T* **unfolding** $cdcl_W$*-all-struct-inv-def* **by** *auto*
have *trail ?T* $\models$*as CNot C*
by (*simp add*: *cut-trail-wrt-clause-CNot-trail*)
then have [*simp*]: $cdcl_W$*-conflicting ?T*
unfolding $cdcl_W$*-conflicting-def* **apply** *simp*
by (*metis M* ‹$cdcl_W$*-conflicting T*› *append-assoc* $cdcl_W$*-conflicting-decomp*(*2*))

have
decomp-T: *all-decomposition-implies-m* (*init-clss T*) (*get-all-marked-decomposition* (*trail T*))
using *inv-T* **unfolding** $cdcl_W$*-all-struct-inv-def* **by** *auto*
have *all-decomposition-implies-m* (*init-clss ?T*)
(*get-all-marked-decomposition* (*trail ?T*))
unfolding *all-decomposition-implies-def*
proof *clarify*
fix *a b*
assume (*a*, *b*) $\in$ *set* (*get-all-marked-decomposition* (*trail ?T*))
from *in-get-all-marked-decomposition-in-get-all-marked-decomposition-prepend*[*OF this*, *of M*]
obtain *b′* **where**
(*a*, *b′* @ *b*) $\in$ *set* (*get-all-marked-decomposition* (*trail T*))
using *M* **by** *auto*
then have *unmark a* $\cup$ *set-mset* (*init-clss T*) $\models$*ps unmark* (*b′* @ *b*)

using *decomp-T* **unfolding** *all-decomposition-implies-def* **by** *fastforce*
then have *unmark a* ∪ *set-mset* (*init-clss ?T*)
$\models$*ps unmark* (*b* @ *b'*)
by (*simp add*: *Un-commute*)
then show *unmark a* ∪ *set-mset* (*init-clss ?T*)
$\models$*ps unmark b*
by (*auto simp*: *image-Un*)
qed

have [*simp*]: $cdcl_W$*-learned-clause ?T*
using *inv-T* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-learned-clause-def*
by (*auto dest*!: *H-proped simp*: *clauses-def*)
show *?thesis*
using ‹*all-decomposition-implies-m* (*init-clss ?T*)
(*get-all-marked-decomposition* (*trail ?T*))›
unfolding $cdcl_W$*-all-struct-inv-def* **by** (*auto simp*: *add-new-clause-and-update-def*)
qed

lemma $cdcl_W$*-all-struct-inv-add-new-clause-and-update-*$cdcl_W$*-stgy-inv*:
assumes
inv-s: $cdcl_W$*-stgy-invariant T* **and**
inv: $cdcl_W$*-all-struct-inv T* **and**
tr-T-N[*simp*]: *trail T* $\models$*asm N* **and**
tr-C[*simp*]: *trail T* $\models$*as CNot C* **and**
[*simp*]: *distinct-mset C*
shows $cdcl_W$*-stgy-invariant* (*add-new-clause-and-update C T*) (**is** $cdcl_W$*-stgy-invariant ?T'*)
proof −
have $cdcl_W$*-all-struct-inv ?T'*
using $cdcl_W$*-all-struct-inv-add-new-clause-and-update-*$cdcl_W$*-all-struct-inv assms* **by** *blast*
then have
no-dup-cut-T[*simp*]: *no-dup* (*trail* (*cut-trail-wrt-clause C* (*trail T*) *T*)) **and**
n-d[*simp*]: *no-dup* (*trail T*)
using $cdcl_W$*-M-level-inv-decomp*(*2*) $cdcl_W$*-all-struct-inv-def inv*
n-dup-no-dup-trail-cut-trail-wrt-clause **by** *blast+*
then have *trail* (*add-new-clause-and-update C T*) $\models$*as CNot C*
by (*simp add*: *add-new-clause-and-update-def cut-trail-wrt-clause-CNot-trail*
$cdcl_W$*-M-level-inv-def* $cdcl_W$*-all-struct-inv-def*)
obtain *MT* **where**
MT: *trail T* = *MT* @ *trail* (*cut-trail-wrt-clause C* (*trail T*) *T*)
using *trail-cut-trail-wrt-clause* **by** *blast*
consider
(*false*) $\forall L\in\#C.$ − *L* $\notin$ *lits-of* (*trail T*) **and***trail* (*cut-trail-wrt-clause C* (*trail T*) *T*) = []
| (*not-false*) − *lit-of* (*hd* (*trail* (*cut-trail-wrt-clause C* (*trail T*) *T*))) $\in\#$ *C* **and**
1 ≤ *length* (*trail* (*cut-trail-wrt-clause C* (*trail T*) *T*))
using *cut-trail-wrt-clause-hd-trail-in-or-empty-trail*[*of C T*] **by** *auto*
then show *?thesis*
proof *cases*
case *false* **note** *C* = *this*(*1*) **and** *empty-tr* = *this*(*2*)
then have [*simp*]: *C* = {#}
by (*simp add*: *in-CNot-implies-uminus*(*2*) *multiset-eqI*)
show *?thesis*
using *empty-tr* **unfolding** $cdcl_W$*-stgy-invariant-def no-smaller-confl-def*
$cdcl_W$*-all-struct-inv-def* **by** (*auto simp*: *add-new-clause-and-update-def*)
next
case *not-false* **note** *C* = *this*(*1*) **and** *l* = *this*(*2*)

let *?L* = − *lit-of* (*hd* (*trail* (*cut-trail-wrt-clause C* (*trail T*) *T*)))
have *get-all-levels-of-marked* (*trail* (*add-new-clause-and-update C T*)) =
  rev [*1..<1* + *length* (*get-all-levels-of-marked* (*trail* (*add-new-clause-and-update C T*)))]
  using ‹$cdcl_W$*-all-struct-inv ?T′*› **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-def*
  by *blast*
moreover
  have *backtrack-lvl* (*cut-trail-wrt-clause C* (*trail T*) *T*) =
    length (*get-all-levels-of-marked* (*trail* (*add-new-clause-and-update C T*)))
    using ‹$cdcl_W$*-all-struct-inv ?T′*› **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-def*
    by (*auto simp*:*add-new-clause-and-update-def*)
moreover
  have *no-dup* (*trail* (*cut-trail-wrt-clause C* (*trail T*) *T*))
    using ‹$cdcl_W$*-all-struct-inv ?T′*› **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-def*
    by (*auto simp*:*add-new-clause-and-update-def*)
  then have *atm-of ?L* ∉ *atm-of* ‘ *lits-of* (*tl* (*trail* (*cut-trail-wrt-clause C* (*trail T*) *T*)))
    apply (*cases trail* (*cut-trail-wrt-clause C* (*trail T*) *T*))
    apply (*auto*)
    using *Marked-Propagated-in-iff-in-lits-of defined-lit-map* **by** *blast*

ultimately have *L*: *get-level* (*trail* (*cut-trail-wrt-clause C* (*trail T*) *T*)) (−*?L*)
  = *length* (*get-all-levels-of-marked* (*trail* (*cut-trail-wrt-clause C* (*trail T*) *T*)))
  using *get-level-get-rev-level-get-all-levels-of-marked*[*OF*
    ‹*atm-of ?L* ∉ *atm-of* ‘ *lits-of* (*tl* (*trail* (*cut-trail-wrt-clause C* (*trail T*) *T*)))›,
    of [*hd* (*trail* (*cut-trail-wrt-clause C* (*trail T*) *T*))]]

    apply (*cases trail* (*add-init-cls C* (*cut-trail-wrt-clause C* (*trail T*) *T*));
     cases hd (*trail* (*cut-trail-wrt-clause C* (*trail T*) *T*)))
    using *l* **by** (*auto split*: *split-if-asm*
      simp:*rev-swap*[*symmetric*] *add-new-clause-and-update-def*)

have *L′*: *length* (*get-all-levels-of-marked* (*trail* (*cut-trail-wrt-clause C* (*trail T*) *T*)))
  = *backtrack-lvl* (*cut-trail-wrt-clause C* (*trail T*) *T*)
  using ‹$cdcl_W$*-all-struct-inv ?T′*› **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-def*
  by (*auto simp*:*add-new-clause-and-update-def*)

have [*simp*]: *no-smaller-confl* (*update-conflicting* (*Some C*)
  (*add-init-cls C* (*cut-trail-wrt-clause C* (*trail T*) *T*)))
  unfolding *no-smaller-confl-def*
proof (*clarify*, *goal-cases*)
  case (*1 M K i M′ D*)
  then consider
      (*DC*) *D* = *C*
    | (*D-T*) *D* ∈# *clauses T*
    by (*auto simp*: *clauses-def split*: *split-if-asm*)
  then show *False*
    proof *cases*
      case *D-T*
      have *no-smaller-confl T*
        using *inv-s* **unfolding** $cdcl_W$*-stgy-invariant-def* **by** *auto*
      have (*MT* @ *M′*) @ *Marked K i* # *M* = *trail T*
        using *MT 1*(*1*) **by** *auto*
      thus *False* **using** *D-T* ‹*no-smaller-confl T*› *1*(*3*) **unfolding** *no-smaller-confl-def* **by** *blast*
    next
      case *DC* **note** -[*simp*] = *this*
      then have *atm-of* (−*?L*) ∈ *atm-of* ‘ (*lits-of M*)

using *1*(*3*) *C in-CNot-implies-uminus*(*2*) **by** *blast*
moreover
have *lit-of* (*hd* (*M'* @ *Marked K i* # [])) = −*?L*
using *l 1*(*1*)[*symmetric*] *inv*
by (*cases trail* (*add-init-cls C* (*cut-trail-wrt-clause C* (*trail T*) *T*)))
(*auto dest*!: *arg-cong*[*of* - # - - *hd*] *simp*: *hd-append* *cdcl*$_W$*-all-struct-inv-def*
cdcl$_W$*-M-level-inv-def*)
from *arg-cong*[*OF this*, *of atm-of*]
have *atm-of* (−*?L*) ∈ *atm-of* ' (*lits-of* (*M'* @ *Marked K i* # []))
by (*cases* (*M'* @ *Marked K i* # [])) *auto*
moreover have *no-dup* (*trail* (*cut-trail-wrt-clause C* (*trail T*) *T*))
using ‹*cdcl*$_W$*-all-struct-inv ?T'*› **unfolding** *cdcl*$_W$*-all-struct-inv-def*
cdcl$_W$*-M-level-inv-def* **by** (*auto simp*: *add-new-clause-and-update-def*)
ultimately show *False*
unfolding *1*(*1*)[*symmetric*, *simplified*]
apply *auto*
using *Marked-Propagated-in-iff-in-lits-of defined-lit-map* **apply** *blast*
by (*metis IntI Marked-Propagated-in-iff-in-lits-of defined-lit-map empty-iff*)
qed
qed
show *?thesis* **using** *L L' C*
unfolding *cdcl*$_W$*-stgy-invariant-def*
unfolding *cdcl*$_W$*-all-struct-inv-def* **by** (*auto simp*: *add-new-clause-and-update-def*)
qed
qed

lemma *full-cdcl*$_W$*-stgy-inv-normal-form*:
assumes
full: *full cdcl*$_W$*-stgy S T* **and**
inv-s: *cdcl*$_W$*-stgy-invariant S* **and**
inv: *cdcl*$_W$*-all-struct-inv S*
shows *conflicting T* = *Some* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss S*))
∨ *conflicting T* = *None* ∧ *trail T* ⊨*asm init-clss S* ∧ *satisfiable* (*set-mset* (*init-clss S*))
proof −
have *no-step cdcl*$_W$*-stgy T*
using *full* **unfolding** *full-def* **by** *blast*
moreover have *cdcl*$_W$*-all-struct-inv T* **and** *inv-s*: *cdcl*$_W$*-stgy-invariant T*
apply (*metis cdcl*$_W$*.rtranclp-cdcl*$_W$*-stgy-rtranclp-cdcl*$_W$ *cdcl*$_W$*-axioms full full-def inv*
rtranclp-cdcl$_W$*-all-struct-inv-inv*)
by (*metis full full-def inv inv-s rtranclp-cdcl*$_W$*-stgy-cdcl*$_W$*-stgy-invariant*)
ultimately have *conflicting T* = *Some* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss T*))
∨ *conflicting T* = *None* ∧ *trail T* ⊨*asm init-clss T*
using *cdcl*$_W$*-stgy-final-state-conclusive*[*of T*] *full*
unfolding *cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-stgy-invariant-def full-def* **by** *fast*
moreover have *consistent-interp* (*lits-of* (*trail T*))
using ‹*cdcl*$_W$*-all-struct-inv T*› **unfolding** *cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-M-level-inv-def*
by *auto*
moreover have *init-clss S* = *init-clss T*
using *inv* **unfolding** *cdcl*$_W$*-all-struct-inv-def*
by (*metis rtranclp-cdcl*$_W$*-stgy-no-more-init-clss full full-def*)
ultimately show *?thesis*
by (*metis satisfiable-carac' true-annot-def true-annots-def true-clss-def*)
qed

lemma *incremental-cdcl*$_W$*-inv*:

assumes
inc: *incremental-cdcl*$_W$ *S T* **and**
inv: *cdcl*$_W$*-all-struct-inv S* **and**
s-inv: *cdcl*$_W$*-stgy-invariant S*
shows
cdcl$_W$*-all-struct-inv T* **and**
cdcl$_W$*-stgy-invariant T*
using *inc*
proof (*induction*)
case (*add-confl C T*)
let *?T* = (*update-conflicting* (*Some C*) (*add-init-cls C* (*cut-trail-wrt-clause C* (*trail S*) *S*)))
have *cdcl*$_W$*-all-struct-inv ?T* **and** *inv-s-T*: *cdcl*$_W$*-stgy-invariant ?T*
using *add-confl.hyps*(*1*,*2*,*4*) *add-new-clause-and-update-def*
cdcl$_W$*-all-struct-inv-add-new-clause-and-update-cdcl*$_W$*-all-struct-inv inv* **apply** *auto*[*1*]
using *add-confl.hyps*(*1*,*2*,*4*) *add-new-clause-and-update-def*
cdcl$_W$*-all-struct-inv-add-new-clause-and-update-cdcl*$_W$*-stgy-inv inv s-inv* **by** *auto*
case *1* **show** *?case*
by (*metis add-confl.hyps*(*1*,*2*,*4*,*5*) *add-new-clause-and-update-def*
cdcl$_W$*-all-struct-inv-add-new-clause-and-update-cdcl*$_W$*-all-struct-inv*
rtranclp-cdcl$_W$*-all-struct-inv-inv rtranclp-cdcl*$_W$*-stgy-rtranclp-cdcl*$_W$ *full-def inv*)

case *2* **show** *?case*
by (*metis inv-s-T add-confl.hyps*(*1*,*2*,*4*,*5*) *add-new-clause-and-update-def*
cdcl$_W$*-all-struct-inv-add-new-clause-and-update-cdcl*$_W$*-all-struct-inv full-def inv*
rtranclp-cdcl$_W$*-stgy-cdcl*$_W$*-stgy-invariant*)
next
case (*add-no-confl C T*)
case *1*
have *cdcl*$_W$*-all-struct-inv* (*add-init-cls C S*)
using *inv* ‹*distinct-mset C*› **unfolding** *cdcl*$_W$*-all-struct-inv-def no-strange-atm-def*
cdcl$_W$*-M-level-inv-def distinct-cdcl*$_W$*-state-def cdcl*$_W$*-conflicting-def cdcl*$_W$*-learned-clause-def*
by (*auto simp*: *all-decomposition-implies-insert-single clauses-def*)
then show *?case*
using *add-no-confl*(*5*) **unfolding** *full-def* **by** (*auto intro*: *rtranclp-cdcl*$_W$*-stgy-cdcl*$_W$*-all-struct-inv*)
case *2* **have** *cdcl*$_W$*-stgy-invariant* (*add-init-cls C S*)
using *s-inv* ‹¬ *trail S* ⊨*as CNot C*› *inv* **unfolding** *cdcl*$_W$*-stgy-invariant-def no-smaller-confl-def*
eq-commute[*of - trail -*] *cdcl*$_W$*-M-level-inv-def cdcl*$_W$*-all-struct-inv-def*
by (*auto simp*: *true-annots-true-cls-def-iff-negation-in-model clauses-def split*: *split-if-asm*)
then show *?case*
by (*metis* ‹*cdcl*$_W$*-all-struct-inv* (*add-init-cls C S*)› *add-no-confl.hyps*(*5*) *full-def*
rtranclp-cdcl$_W$*-stgy-cdcl*$_W$*-stgy-invariant*)
qed

lemma *rtranclp-incremental-cdcl*$_W$*-inv*:
assumes
inc: *incremental-cdcl*$_W$** *S T* **and**
inv: *cdcl*$_W$*-all-struct-inv S* **and**
s-inv: *cdcl*$_W$*-stgy-invariant S*
shows
cdcl$_W$*-all-struct-inv T* **and**
cdcl$_W$*-stgy-invariant T*
using *inc* **apply** *induction*
using *inv* **apply** *simp*
using *s-inv* **apply** *simp*
using *incremental-cdcl*$_W$*-inv* **by** *blast*+

lemma *incremental-conclusive-state*:
  assumes
    inc: *incremental-cdcl*$_W$ *S T* **and**
    inv: *cdcl*$_W$*-all-struct-inv S* **and**
    s-inv: *cdcl*$_W$*-stgy-invariant S*
  shows *conflicting T* = *Some* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss T*))
    ∨ *conflicting T* = *None* ∧ *trail T* ⊨*asm init-clss T* ∧ *satisfiable* (*set-mset* (*init-clss T*))
  using *inc* **apply** *induction*

  apply (*metis Nitpick.rtranclp-unfold add-confl full-cdcl*$_W$*-stgy-inv-normal-form full-def*
    incremental-cdcl$_W$*-inv*(*1*) *incremental-cdcl*$_W$*-inv*(*2*) *inv s-inv*)
  by (*metis* (*full-types*) *rtranclp-unfold add-no-confl full-cdcl*$_W$*-stgy-inv-normal-form*
    full-def incremental-cdcl$_W$*-inv*(*1*) *incremental-cdcl*$_W$*-inv*(*2*) *inv s-inv*)

lemma *tranclp-incremental-correct*:
  assumes
    inc: *incremental-cdcl*$_W$$^{++}$ *S T* **and**
    inv: *cdcl*$_W$*-all-struct-inv S* **and**
    s-inv: *cdcl*$_W$*-stgy-invariant S*
  shows *conflicting T* = *Some* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss T*))
    ∨ *conflicting T* = *None* ∧ *trail T* ⊨*asm init-clss T* ∧ *satisfiable* (*set-mset* (*init-clss T*))
  using *inc* **apply** *induction*
   using *assms incremental-conclusive-state* **apply** *blast*
  by (*meson incremental-conclusive-state inv rtranclp-incremental-cdcl*$_W$*-inv s-inv*
    tranclp-into-rtranclp)

lemma *blocked-induction-with-marked*:
  assumes
    n-d: *no-dup* (*L* # *M*) **and**
    nil: *P* [] **and**
    append: $\bigwedge$*M L M'*. *P M* ⟹ *is-marked L* ⟹ ∀ *m* ∈ *set M'*. ¬*is-marked m* ⟹ *no-dup* (*L* # *M'* @
M) ⟹
      P (*L* # *M'* @ *M*) **and**
    L: *is-marked L*
  shows
    P (*L* # *M*)
  using *n-d L*
proof (*induction card* {*L'* ∈ *set M*. *is-marked L'*} *arbitrary*: *L M*)
  case *0* **note** *n* = *this*(*1*) **and** *n-d* = *this*(*2*) **and** *L* = *this*(*3*)
  then have ∀ *m* ∈ *set M*. ¬*is-marked m* **by** *auto*
  then show *?case* **using** *append*[*of* [] *L M*] *L nil n-d* **by** *auto*
next
  case (*Suc n*) **note** *IH* = *this*(*1*) **and** *n* = *this*(*2*) **and** *n-d* = *this*(*3*) **and** *L* = *this*(*4*)
  have ∃ *L'* ∈ *set M*. *is-marked L'*
    proof (*rule ccontr*)
      assume ¬*?thesis*
      then have *H*: {*L'* ∈ *set M*. *is-marked L'*} = {}
        by *auto*
      show *False* **using** *n* **unfolding** *H* **by** *auto*
    qed
  then obtain *L' M' M''* **where**
    M: *M* = *M'* @ *L'* # *M''* **and**
    L': *is-marked L'* **and**
    nm: ∀ *m*∈*set M'*. ¬*is-marked m*

by (*auto elim*!: *split-list-first-propE*)
have $Suc\ n = card\ \{L' \in set\ M.\ is\text{-}marked\ L'\}$
using *n* **.**
moreover have $\{L' \in set\ M.\ is\text{-}marked\ L'\} = \{L'\} \cup \{L' \in set\ M''.\ is\text{-}marked\ L'\}$
using *nm L' n-d* **unfolding** *M* **by** *auto*
moreover have $L' \notin \{L' \in set\ M''.\ is\text{-}marked\ L'\}$
using *n-d* **unfolding** *M* **by** *auto*
ultimately have $n = card\ \{L'' \in set\ M''.\ is\text{-}marked\ L''\}$
using *n L'* **by** *auto*
then have $P\ (L' \# M'')$ **using** *IH L' n-d M* **by** *auto*
then show *?case* **using** *append*[*of* $L' \# M''\ L\ M'$] *nm L n-d* **unfolding** *M* **by** *blast*
qed

lemma *trail-bloc-induction*:
assumes
n-d: *no-dup M* **and**
nil: *P* [] **and**
append: $\bigwedge M\ L\ M'.\ P\ M \Longrightarrow is\text{-}marked\ L \Longrightarrow \forall m \in set\ M'.\ \neg is\text{-}marked\ m \Longrightarrow no\text{-}dup\ (L \# M'\ @\ M) \Longrightarrow$
$P\ (L \# M'\ @\ M)$ **and**
append-nm: $\bigwedge M'\ M''.\ P\ M' \Longrightarrow M = M''\ @\ M' \Longrightarrow \forall m{\in}set\ M''.\ \neg is\text{-}marked\ m \Longrightarrow P\ M$
shows
P M
proof (*cases* $\{L' \in set\ M.\ is\text{-}marked\ L'\} = \{\}$)
case *True*
then show *?thesis* **using** *append-nm*[*of* [] *M*] *nil* **by** *auto*
next
case *False*
then have $\exists L' \in set\ M.\ is\text{-}marked\ L'$
by *auto*
then obtain $L'\ M'\ M''$ **where**
M: $M = M'\ @\ L' \# M''$ **and**
L': *is-marked L'* **and**
nm: $\forall m{\in}set\ M'.\ \neg is\text{-}marked\ m$
by (*auto elim*!: *split-list-first-propE*)
have $P\ (L' \# M'')$
apply (*rule blocked-induction-with-marked*)
using *n-d* **unfolding** *M* **apply** *simp*
using *nil* **apply** *simp*
using *append* **apply** *simp*
using *L'* **by** *auto*
then show *?thesis*
using *append-nm*[*of* - *M'*] *nm* **unfolding** *M* **by** *simp*
qed

inductive *Tcons* :: ($'v$, *nat*, $'v$ *clause*) *ann-literals* ⇒($'v$, *nat*, $'v$ *clause*) *ann-literals* ⇒ *bool*
for *M* :: ($'v$, *nat*, $'v$ *clause*) *ann-literals* **where**
Tcons M [] |
$Tcons\ M\ M' \Longrightarrow M = M''\ @\ M' \Longrightarrow (\forall m \in set\ M''.\ \neg is\text{-}marked\ m) \Longrightarrow Tcons\ M\ (M''\ @\ M')$ |
$Tcons\ M\ M' \Longrightarrow is\text{-}marked\ L \Longrightarrow M = M'''\ @\ L \# M''\ @\ M' \Longrightarrow (\forall m \in set\ M''.\ \neg is\text{-}marked\ m) \Longrightarrow$
$Tcons\ M\ (L \# M''\ @\ M')$

lemma *Tcons-same-end*: $Tcons\ M\ M' \Longrightarrow \exists M''.\ M = M''\ @\ M'$
by (*induction rule*: *Tcons.induct*) *auto*

end

end

9 2-Watched-Literal

theory *CDCL-Two-Watched-Literals*
imports *CDCL-WNOT*
begin

9.1 Datastructure and Access Functions

Only the 2-watched literals have to be verified here: the backtrack level and the trail that appear in the state are not related to the 2-watched algoritm.

datatype $'v$ *twl-clause* =
TWL-Clause (*watched*: $'v$) (*unwatched*: $'v$)

abbreviation *raw-clause* :: $'v$ *clause twl-clause* ⇒ $'v$ *clause* **where**
raw-clause C ≡ *watched C* + *unwatched C*

datatype ($'a$, $'b$, $'c$, $'d$) *twl-state* =
TWL-State (*trail*: $'a$ *list*) (*init-clss*: $'b$)
(*learned-clss*: $'b$) (*backtrack-lvl*: $'c$)
(*conflicting*: $'d$ *option*)

type-synonym ($'v$, $'lvl$, $'mark$) *twl-state-abs* =
(($'v$, $'lvl$, $'mark$) *ann-literal*, $'v$ *clause twl-clause multiset*, $'lvl$, $'v$ *clause*) *twl-state*

abbreviation *raw-init-clss* **where**
raw-init-clss S ≡ *image-mset raw-clause* (*init-clss S*)

abbreviation *raw-learned-clss* **where**
raw-learned-clss S ≡ *image-mset raw-clause* (*learned-clss S*)

abbreviation *clauses* **where**
clauses S ≡ *init-clss S* + *learned-clss S*

abbreviation *raw-clauses* **where**
raw-clauses S ≡ *image-mset raw-clause* (*clauses S*)

definition
candidates-propagate :: ($'v$, $'lvl$, $'mark$) *twl-state-abs* ⇒ ($'v$ *literal* × $'v$ *clause*) *set*
where
candidates-propagate S =
{(*L*, *raw-clause C*) | *L C*.
C ∈# *clauses S* ∧ *watched C* − *mset-set* (*uminus* ‘ *lits-of* (*trail S*)) = {#*L*#} ∧
undefined-lit (*trail S*) *L*}

definition *candidates-conflict* :: ($'v$, $'lvl$, $'mark$) *twl-state-abs* ⇒ $'v$ *clause set* **where**
candidates-conflict S =
{*raw-clause C* | *C*. *C* ∈# *clauses S* ∧ *watched C* ⊆# *mset-set* (*uminus* ‘ *lits-of* (*trail S*))}

primrec (*nonexhaustive*) *index* :: $'a$ *list* ⇒ $'a$ ⇒ *nat* **where**
index (*a* # *l*) *c* = (*if a* = *c then 0 else 1*+*index l c*)

lemma *index-nth*:
  $a \in set\ l \Longrightarrow l\ !\ (index\ l\ a) = a$
  by (*induction l*) *auto*

9.2 Invariants

We need the following property about updates: if there is a literal L with $-L$ in the trail, and L is not watched, then it stays unwatched; i.e., while updating with *rewatch* it does not get swap with a watched literal L' such that $-L'$ is in the trail.

primrec *watched-decided-most-recently* :: ($'v$, $'lvl$, $'mark$) *ann-literal list* $\Rightarrow$ $'v$ *clause twl-clause* $\Rightarrow$ *bool*
  where
watched-decided-most-recently M (*TWL-Clause* W UW) $\longleftrightarrow$
  $(\forall L' \in\# W.\ \forall L \in\# UW.$
    $-L' \in lits\text{-}of\ M \longrightarrow -L \in lits\text{-}of\ M \longrightarrow L \notin\# W \longrightarrow$
      $index\ (map\ lit\text{-}of\ M)\ (-L') \leq index\ (map\ lit\text{-}of\ M)\ (-L))$

Here are the invariant strictly related to the 2-WL data structure.

primrec *wf-twl-cls* :: ($'v$, $'lvl$, $'mark$) *ann-literal list* $\Rightarrow$ $'v$ *clause twl-clause* $\Rightarrow$ *bool* **where**
  wf-twl-cls M (*TWL-Clause* W UW) $\longleftrightarrow$
   $distinct\text{-}mset\ W \wedge size\ W \leq 2 \wedge (size\ W < 2 \longrightarrow set\text{-}mset\ UW \subseteq set\text{-}mset\ W) \wedge$
   $(\forall L \in\# W.\ -L \in lits\text{-}of\ M \longrightarrow (\forall L' \in\# UW.\ L' \notin\# W \longrightarrow -L' \in lits\text{-}of\ M)) \wedge$
   watched-decided-most-recently M (*TWL-Clause* W UW)

lemma $-L \in lits\text{-}of\ M \Longrightarrow \{i.\ map\ lit\text{-}of\ M!i = -L\} \neq \{\}$
  unfolding *set-map-lit-of-lits-of*[*symmetric*] *set-conv-nth*
  by (*smt Collect-empty-eq mem-Collect-eq*)

lemma *size-mset-2*: $size\ x1 = 2 \longleftrightarrow (\exists\, a\ b.\ x1 = \{\#a,\ b\#\})$
  by (*metis* (*no-types*, *hide-lams*) *Suc-eq-plus1 one-add-one size-1-singleton-mset size-Diff-singleton size-Suc-Diff1 size-eq-Suc-imp-eq-union size-single union-single-eq-diff union-single-eq-member*)

lemma *distinct-mset-size-2*: $distinct\text{-}mset\ \{\#a,\ b\#\} \longleftrightarrow a \neq b$
  unfolding *distinct-mset-def* **by** *auto*

lemma *wf-twl-cls-annotation-indepnedant*:
  assumes M: $map\ lit\text{-}of\ M = map\ lit\text{-}of\ M'$
  shows *wf-twl-cls* M (*TWL-Clause* W UW) $\longleftrightarrow$ *wf-twl-cls* M' (*TWL-Clause* W UW)
proof −
  have $lits\text{-}of\ M = lits\text{-}of\ M'$
    using *arg-cong*[*OF M*, *of set*] **by** (*simp add*: *lits-of-def*)
  then show *?thesis*
    by (*simp add*: *lits-of-def M*)
qed

lemma *wf-twl-cls-wf-twl-cls-tl*:
  assumes *wf*: *wf-twl-cls M C* **and** *n-d*: *no-dup M*
  shows *wf-twl-cls* (*tl M*) *C*
proof (*cases M*)
  case *Nil*
  then show *?thesis* **using** *wf*
    by (*cases C*) (*simp add*: *wf-twl-cls.simps*[*of tl -*])
next

case (*Cons l M'*) **note** *M* = *this*(*1*)
obtain *W UW* **where** *C*: *C* = *TWL-Clause W UW*
by (*cases C*)
{ fix *L L'*
assume
LW: *L* ∈# *W* **and**
LM: − *L* ∈ *lits-of M'* **and**
L'UW: *L'* ∈# *UW* **and**
count W L' = *0*
then have
L'M: − *L'* ∈ *lits-of M*
using *wf* **by** (*auto simp*: *C M*)
have *watched-decided-most-recently M C*
using *wf* **by** (*auto simp*: *C*)
then have
index (*map lit-of M*) (−*L*) ≤ *index* (*map lit-of M*) (−*L'*)
using *LM L'M L'UW LW* ‹*count W L'* = *0*›
by (*metis* (*no-types*, *lifting*) *C M bspec-mset insert-iff less-not-refl2 lits-of-cons watched-decided-most-recently.simps*)
then have − *L'* ∈ *lits-of M'*
using ‹*count W L'* = *0*› *LW L'M* **by** (*auto simp*: *C M split*: *split-if-asm*)
}
moreover
{
fix *L' L*
assume
L' ∈# *W* **and**
L ∈# *UW* **and**
L'M: − *L'* ∈ *lits-of M'* **and**
− *L* ∈ *lits-of M'* **and**
L ∉# *W*
moreover
have *lit-of l* ≠ − *L'*
using *n-d* **unfolding** *M*
by (*metis* (*no-types*) *L'M M Marked-Propagated-in-iff-in-lits-of defined-lit-map distinct.simps*(*2*) *list.simps*(*9*) *set-map*)
moreover have *watched-decided-most-recently M C*
using *wf* **by** (*auto simp*: *C*)
ultimately have *index* (*map lit-of M'*) (− *L'*) ≤ *index* (*map lit-of M'*) (− *L*)
by (*fastforce simp*: *M C split*: *split-if-asm*)
}
moreover have *distinct-mset W* **and** *size W* ≤ *2* **and** (*size W* < *2* ⟶ *set-mset UW* ⊆ *set-mset W*)
using *wf* **by** (*auto simp*: *C M*)
ultimately show *?thesis* **by** (*auto simp add*: *M C*)
qed

definition *wf-twl-state* :: ('*v*, '*lvl*, '*mark*) *twl-state-abs* ⇒ *bool* **where**
wf-twl-state S ⟷ (∀ *C* ∈# *clauses S*. *wf-twl-cls* (*trail S*) *C*) ∧ *no-dup* (*trail S*)

lemma *wf-candidates-propagate-sound*:
assumes *wf*: *wf-twl-state S* **and**
cand: (*L*, *C*) ∈ *candidates-propagate S*
shows *trail S* ⊨*as CNot* (*mset-set* (*set-mset C* − {*L*})) ∧ *undefined-lit* (*trail S*) *L*
proof

def $M \equiv$ *trail S*
def $N \equiv$ *init-clss S*
def $U \equiv$ *learned-clss S*

note *MNU-defs* [*simp*] = *M-def N-def U-def*

obtain *Cw* **where** *cw*:
  C = *raw-clause Cw*
  Cw $\in\#$ $N + U$
  watched Cw $-$ *mset-set* (*uminus* ‘ *lits-of M*) = {#*L*#}
  undefined-lit M L
  using *cand* **unfolding** *candidates-propagate-def MNU-defs* **by** *blast*

obtain *W UW* **where** *cw-eq*: *Cw* = *TWL-Clause W UW*
  by (*cases Cw*, *blast*)

have *l-w*: $L \in\# W$
  by (*metis Multiset.diff-le-self cw*(*3*) *cw-eq mset-leD multi-member-last twl-clause.sel*(*1*))

have *wf-c*: *wf-twl-cls M Cw*
  using *wf* ‹*Cw* $\in\#$ $N + U$› **unfolding** *wf-twl-state-def* **by** *simp*

have *w-nw*:
  distinct-mset W
  size W $< 2 \Longrightarrow$ *set-mset UW* $\subseteq$ *set-mset W*
  $\bigwedge L\ L'.\ L \in\# W \Longrightarrow -L \in$ *lits-of M* $\Longrightarrow L' \in\# UW \Longrightarrow L' \notin\# W \Longrightarrow -L' \in$ *lits-of M*
 using *wf-c* **unfolding** *cw-eq* **by** *auto*

have $\forall L' \in$ *set-mset C* $- \{L\}.\ -L' \in$ *lits-of M*
proof (*cases size W* < 2)
  case *True*
  moreover have *size W* $\neq 0$
    using *cw*(*3*) *cw-eq* **by** *auto*
  ultimately have *size W* $= 1$
    by *linarith*
  then have *w*: W = {#*L*#}
    by (*metis* (*no-types*, *lifting*) *Multiset.diff-le-self cw*(*3*) *cw-eq single-not-empty*
      size-1-singleton-mset subset-mset.add-diff-inverse union-is-single twl-clause.sel(*1*))
  from *True* **have** *set-mset UW* $\subseteq$ *set-mset W*
    using *w-nw*(*2*) **by** *blast*
  then show *?thesis*
    using *w cw*(*1*) *cw-eq* **by** *auto*
next
  case *sz2*: *False*
  show *?thesis*
  proof
    fix L'
    assume l': $L' \in$ *set-mset C* $- \{L\}$
    have *ex-la*: $\exists La.\ La \neq L \wedge La \in\# W$
    proof (*cases W*)
      case *empty*
      thus *?thesis*
        using *l-w* **by** *auto*
    next
      case *lb*: (*add W' Lb*)

show *?thesis*
proof (*cases W′*)
case *empty*
thus *?thesis*
using *lb sz2* **by** *simp*
next
case *lc*: (*add W′′ Lc*)
thus *?thesis*
by (*metis add-gr-0 count-union distinct-mset-single-add lb union-single-eq-member w-nw*(*1*))
qed
qed
then obtain *La* **where** *la*: $La \neq L$ $La \in\# W$
by *blast*
then have $La \in\#$ *mset-set* (*uminus* ‘ *lits-of M*)
using *cw*(*3*)[*unfolded cw-eq*, *simplified*, *folded M-def*]
by (*metis count-diff count-single diff-zero not-gr0*)
then have *nla*: $-La \in$ *lits-of M*
by *auto*
then show $-L' \in$ *lits-of M*

proof −
have *f1*: $L' \in$ *set-mset C*
using *l′* **by** *blast*
have *f2*: $L' \notin \{L\}$
using *l′* **by** *fastforce*
have $\bigwedge l\ L.$ − (*l*::′*a literal*) $\in L \vee l \notin$ *uminus* ‘ *L*
by *force*
then have $\bigwedge l.$ − $l \in$ *lits-of M* $\vee$ *count* {#*L*#} *l* = *count* (*C* − *UW*) *l*
by (*metis* (*no-types*) *add-diff-cancel-right′ count-diff count-mset-set*(*3*) *cw*(*1*) *cw*(*3*) *cw-eq diff-zero twl-clause.sel*(*2*))
then show *?thesis*
by (*smt comm-monoid-add-class.add-0 cw*(*1*) *cw-eq diff-union-cancelR ex-la f1 f2 insertCI less-numeral-extra*(*3*) *mem-set-mset-iff plus-multiset.rep-eq single.rep-eq twl-clause.sel*(*1*) *twl-clause.sel*(*2*) *w-nw*(*3*))
qed
qed
qed
then show *trail S* $\models$*as CNot* (*mset-set* (*set-mset C* − {*L*}))
unfolding *true-annots-def* **by** *auto*

show *undefined-lit* (*trail S*) *L*
using *cw*(*4*) *M-def* **by** *blast*
qed

lemma *wf-candidates-propagate-complete*:
assumes *wf*: *wf-twl-state S* **and**
c-mem: $C \in\#$ *raw-clauses S* **and**
l-mem: $L \in\# C$ **and**
unsat: *trail S* $\models$*as CNot* (*mset-set* (*set-mset C* − {*L*})) **and**
undef: *undefined-lit* (*trail S*) *L*
shows (*L*, *C*) $\in$ *candidates-propagate S*
proof −
def $M \equiv$ *trail S*
def $N \equiv$ *init-clss S*

def $U \equiv$ *learned-clss S*

note *MNU-defs* [*simp*] = *M-def N-def U-def*

obtain *Cw* **where** *cw*: C = *raw-clause Cw Cw* $\in\#$ $N + U$
  using *c-mem* **by** *force*

obtain *W UW* **where** *cw-eq*: *Cw* = *TWL-Clause W UW*
  by (*cases Cw*, *blast*)

have *wf-c*: *wf-twl-cls M Cw*
  using *wf cw*(*2*) **unfolding** *wf-twl-state-def* **by** *simp*

have *w-nw*:
  distinct-mset W
  size W $< 2 \Longrightarrow$ *set-mset UW* $\subseteq$ *set-mset W*
  $\bigwedge L\ L'.\ L \in\#\ W \Longrightarrow -L \in$ *lits-of M* $\Longrightarrow L' \in\#\ UW \Longrightarrow L' \notin\#\ W \Longrightarrow -L' \in$ *lits-of M*
 using *wf-c* **unfolding** *cw-eq* **by** *auto*

have *unit-set*: *set-mset* (W − *mset-set* (*uminus* ‘ *lits-of M*)) = $\{L\}$
proof
  show *set-mset* (W − *mset-set* (*uminus* ‘ *lits-of M*)) $\subseteq \{L\}$
  proof
    fix L'
    assume l': $L' \in$ *set-mset* (W − *mset-set* (*uminus* ‘ *lits-of M*))
    hence l'*-mem-w*: $L' \in$ *set-mset W*
      by *auto*
    have $L' \notin$ *uminus* ‘ *lits-of M*
      using *distinct-mem-diff-mset*[*OF w-nw*(*1*) l'] **by** *simp*
    then have $\neg$ $M \models a$ $\{\#{-L'}\#\}$
      using *image-iff* **by** *fastforce*
    moreover have $L' \in\#\ C$
      using *cw*(*1*) *cw-eq* l'*-mem-w* **by** *auto*
    ultimately have $L' = L$
      unfolding *M-def* **by** (*metis unsat*[*unfolded CNot-def true-annots-def*, *simplified*])
    then show $L' \in \{L\}$
      by *simp*
  qed
next
  show $\{L\} \subseteq$ *set-mset* (W − *mset-set* (*uminus* ‘ *lits-of M*))
  proof *clarify*
    have $L \in\#\ W$
    proof (*cases W*)
      case *empty*
      thus *?thesis*
        using *w-nw*(*2*) *cw*(*1*) *cw-eq l-mem* **by** *auto*
    next
      case (*add W′ La*)
      thus *?thesis*
      proof (*cases La* = L)
        case *True*
        thus *?thesis*
          using *add* **by** *simp*
      next
        case *False*

have $-La \in$ *lits-of M*
using *False add cw*(*1*) *cw-eq unsat*[*unfolded CNot-def true-annots-def*, *simplified*]
by *fastforce*
then show *?thesis*
by (*metis M-def Marked-Propagated-in-iff-in-lits-of add add.left-neutral count-union cw*(*1*) *cw-eq gr0I l-mem twl-clause.sel*(*1*) *twl-clause.sel*(*2*) *undef union-single-eq-member w-nw*(*3*))
qed
qed
moreover have $L \notin\#$ *mset-set* (*uminus* ‘ *lits-of M*)
using *Marked-Propagated-in-iff-in-lits-of undef* **by** *auto*
ultimately show $L \in$ *set-mset* ($W -$ *mset-set* (*uminus* ‘ *lits-of M*))
by *auto*
qed
qed
have *unit*: $W -$ *mset-set* (*uminus* ‘ *lits-of M*) $= \{\#L\#\}$
by (*metis distinct-mset-minus distinct-mset-set-mset-ident distinct-mset-singleton set-mset-single unit-set w-nw*(*1*))

show *?thesis*
unfolding *candidates-propagate-def* **using** *unit undef cw cw-eq* **by** *fastforce*
qed

lemma *wf-candidates-conflict-sound*:
assumes *wf*: *wf-twl-state S* **and**
cand: $C \in$ *candidates-conflict S*
shows *trail S* $\models$*as CNot C* $\wedge$ $C \in\#$ *image-mset raw-clause* (*clauses S*)
proof
def $M \equiv$ *trail S*
def $N \equiv$ *init-clss S*
def $U \equiv$ *learned-clss S*

note *MNU-defs* [*simp*] = *M-def N-def U-def*

obtain *Cw* **where** *cw*:
$C =$ *raw-clause Cw*
Cw $\in\#$ $N + U$
watched Cw $\subseteq\#$ *mset-set* (*uminus* ‘ *lits-of* (*trail S*))
using *cand*[*unfolded candidates-conflict-def*, *simplified*] **by** *auto*

obtain *W UW* **where** *cw-eq*: *Cw* = *TWL-Clause W UW*
by (*cases Cw*, *blast*)

have *wf-c*: *wf-twl-cls M Cw*
using *wf cw*(*2*) **unfolding** *wf-twl-state-def* **by** *simp*

have *w-nw*:
distinct-mset W
size $W < 2 \Longrightarrow$ *set-mset UW* $\subseteq$ *set-mset W*
$\bigwedge L\ L'.\ L \in\#\ W \Longrightarrow -L \in$ *lits-of M* $\Longrightarrow L' \in\#\ UW \Longrightarrow L' \notin\#\ W \Longrightarrow -L' \in$ *lits-of M*
using *wf-c* **unfolding** *cw-eq* **by** *auto*

have $\forall L \in\#\ C.\ -L \in$ *lits-of M*
proof (*cases* $W = \{\#\}$)
case *True*

```
    then have C = {#}
      using cw(1) cw-eq w-nw(2) by auto
    then show ?thesis
      by simp
  next
    case False
    then obtain La where la: La ∈# W
      using multiset-eq-iff by force
    show ?thesis
    proof
      fix L
      assume l: L ∈# C
      show −L ∈ lits-of M
      proof (cases L ∈# W)
        case True
        thus ?thesis
          using cw(3) cw-eq by fastforce
      next
        case False
        thus ?thesis
          by (smt M-def l add-diff-cancel-left' count-diff cw(1) cw(3) la cw-eq
            diff-zero elem-mset-set finite-imageI finite-lits-of-def gr0I imageE mset-leD
            uminus-of-uminus-id twl-clause.sel(1) twl-clause.sel(2) w-nw(3))
      qed
    qed
  qed
  then show trail S ⊨as CNot C
    unfolding CNot-def true-annots-def by auto

  show C ∈# image-mset raw-clause (clauses S)
    using cw by auto
qed

lemma wf-candidates-conflict-complete:
  assumes wf: wf-twl-state S and
    c-mem: C ∈# raw-clauses S and
    unsat: trail S ⊨as CNot C
  shows C ∈ candidates-conflict S
proof −
  def M ≡ trail S
  def N ≡ init-clss S
  def U ≡ learned-clss S

  note MNU-defs [simp] = M-def N-def U-def

  obtain Cw where cw: C = raw-clause Cw Cw ∈# N + U
    using c-mem by force

  obtain W UW where cw-eq: Cw = TWL-Clause W UW
    by (cases Cw, blast)

  have wf-c: wf-twl-cls M Cw
    using wf cw(2) unfolding wf-twl-state-def by simp

  have w-nw:
```

distinct-mset W
size W $<$ *2* $\Longrightarrow$ *set-mset UW* $\subseteq$ *set-mset W*
$\bigwedge L\ L'.\ L \in\#\ W \Longrightarrow -L \in$ *lits-of M* $\Longrightarrow L' \in\#\ UW \Longrightarrow L' \notin\#\ W \Longrightarrow -L' \in$ *lits-of M*
using *wf-c* **unfolding** *cw-eq* **by** *auto*

have $\bigwedge L.\ L \in\#\ C \Longrightarrow -L \in$ *lits-of M*
unfolding *M-def* **using** *unsat*[*unfolded CNot-def true-annots-def*, *simplified*] **by** *blast*
then have *set-mset C* $\subseteq$ *uminus ‘ lits-of M*
by (*metis imageI mem-set-mset-iff subsetI uminus-of-uminus-id*)
then have *set-mset W* $\subseteq$ *uminus ‘ lits-of M*
using *cw*(*1*) *cw-eq* **by** *auto*
then have *subset*: *W* $\subseteq\#$ *mset-set* (*uminus ‘ lits-of M*)
by (*simp add*: *w-nw*(*1*))

have *W* = *watched Cw*
using *cw-eq twl-clause.sel*(*1*) **by** *simp*
then show *?thesis*
using *MNU-defs cw*(*1*) *cw*(*2*) *subset candidates-conflict-def* **by** *blast*
qed

typedef *'v wf-twl* = {*S*::(*'v*, *nat*, *'v clause*) *twl-state-abs. wf-twl-state S*}
morphisms *rough-state-of-twl twl-of-rough-state*
proof −
have *TWL-State* ([]::(*'v*, *nat*, *'v clause*) *ann-literals*)
{#} {#} *0 None* ∈ {*S*:: (*'v*, *nat*, *'v clause*) *twl-state-abs. wf-twl-state S*}
by (*auto simp*: *wf-twl-state-def*)
then show *?thesis* **by** *auto*
qed

lemma [*code abstype*]:
twl-of-rough-state (*rough-state-of-twl S*) = *S*
by (*fact CDCL-Two-Watched-Literals.wf-twl.rough-state-of-twl-inverse*)

lemma *wf-twl-state-rough-state-of-twl*[*simp*]: *wf-twl-state* (*rough-state-of-twl S*)
using *rough-state-of-twl* **by** *auto*

abbreviation *candidates-conflict-twl* :: *'v wf-twl* ⇒ *'v literal multiset set* **where**
candidates-conflict-twl S ≡ *candidates-conflict* (*rough-state-of-twl S*)

abbreviation *candidates-propagate-twl* :: *'v wf-twl* ⇒ (*'v literal* × *'v clause*) *set* **where**
candidates-propagate-twl S ≡ *candidates-propagate* (*rough-state-of-twl S*)

abbreviation *trail-twl* :: *'a wf-twl* ⇒ (*'a*, *nat*, *'a literal multiset*) *ann-literal list* **where**
trail-twl S ≡ *trail* (*rough-state-of-twl S*)

abbreviation *clauses-twl* :: *'a wf-twl* ⇒ *'a literal multiset multiset* **where**
clauses-twl S ≡ *raw-clauses* (*rough-state-of-twl S*)

abbreviation *init-clss-twl* :: *'a wf-twl* ⇒ *'a literal multiset multiset* **where**
init-clss-twl S ≡ *raw-init-clss* (*rough-state-of-twl S*)

abbreviation *learned-clss-twl* :: *'a wf-twl* ⇒ *'a literal multiset multiset* **where**
learned-clss-twl S ≡ *raw-learned-clss* (*rough-state-of-twl S*)

abbreviation *backtrack-lvl-twl* **where**

backtrack-lvl-twl S ≡ *backtrack-lvl* (*rough-state-of-twl S*)

abbreviation *conflicting-twl* **where**
conflicting-twl S ≡ *conflicting* (*rough-state-of-twl S*)

lemma *wf-candidates-twl-conflict-complete*:
  assumes
    c-mem: *C* ∈# *clauses-twl S* **and**
    unsat: *trail-twl S* ⊨*as CNot C*
  shows *C* ∈ *candidates-conflict-twl S*
  using *c-mem unsat wf-candidates-conflict-complete wf-twl-state-rough-state-of-twl* **by** *blast*

abbreviation *update-backtrack-lvl* **where**
  update-backtrack-lvl k S ≡
   TWL-State (*trail S*) (*init-clss S*) (*learned-clss S*) *k* (*conflicting S*)

abbreviation *update-conflicting* **where**
  update-conflicting C S ≡ *TWL-State* (*trail S*) (*init-clss S*) (*learned-clss S*) (*backtrack-lvl S*) *C*

9.3 Abstract 2-WL

definition *tl-trail* **where**
  tl-trail S =
   TWL-State (*tl* (*trail S*)) (*init-clss S*) (*learned-clss S*) (*backtrack-lvl S*) (*conflicting S*)

locale *abstract-twl* =
  fixes
    watch :: (′*v*, *nat*, ′*v clause*) *twl-state-abs* ⇒ ′*v clause* ⇒ ′*v clause twl-clause* **and**
    rewatch :: (′*v*, *nat*, ′*v literal multiset*) *ann-literal* ⇒ (′*v*, *nat*, ′*v clause*) *twl-state-abs* ⇒
      ′*v clause twl-clause* ⇒ ′*v clause twl-clause* **and**
    linearize :: ′*v clauses* ⇒ ′*v clause list* **and**
    restart-learned :: (′*v*, *nat*, ′*v clause*) *twl-state-abs* ⇒ ′*v clause twl-clause multiset*
  assumes
    clause-watch: *no-dup* (*trail S*) ⟹ *raw-clause* (*watch S C*) = *C* **and**
    wf-watch: *no-dup* (*trail S*) ⟹ *wf-twl-cls* (*trail S*) (*watch S C*) **and**
    clause-rewatch: *raw-clause* (*rewatch L S C*′) = *raw-clause C*′ **and**
    wf-rewatch:
      no-dup (*trail S*) ⟹ *undefined-lit* (*trail S*) (*lit-of L*) ⟹ *wf-twl-cls* (*trail S*) *C*′ ⟹
        wf-twl-cls (*L* # *trail S*) (*rewatch L S C*′)
      and
    linearize: *mset* (*linearize N*) = *N* **and**
    restart-learned: *restart-learned S* ⊆# *learned-clss S*
begin

lemma *linearize-mempty*[*simp*]: *linearize* {#} = []
  using *linearize mset-zero-iff* **by** *blast*

definition
  cons-trail :: (′*v*, *nat*, ′*v clause*) *ann-literal* ⇒ (′*v*, *nat*, ′*v clause*) *twl-state-abs* ⇒
    (′*v*, *nat*, ′*v clause*) *twl-state-abs*
where
  cons-trail L S =
   TWL-State (*L* # *trail S*) (*image-mset* (*rewatch L S*) (*init-clss S*))
     (*image-mset* (*rewatch L S*) (*learned-clss S*)) (*backtrack-lvl S*) (*conflicting S*)

definition

add-init-cls :: *'v clause* ⇒ (*'v*, *nat*, *'v clause*) *twl-state-abs* ⇒
  (*'v*, *nat*, *'v clause*) *twl-state-abs*
where
  add-init-cls C S =
   TWL-State (*trail S*) ({#*watch S C*#} + *init-clss S*) (*learned-clss S*) (*backtrack-lvl S*)
    (*conflicting S*)

definition
  add-learned-cls :: *'v clause* ⇒ (*'v*, *nat*, *'v clause*) *twl-state-abs* ⇒
  (*'v*, *nat*, *'v clause*) *twl-state-abs*
where
  add-learned-cls C S =
   TWL-State (*trail S*) (*init-clss S*) ({#*watch S C*#} + *learned-clss S*) (*backtrack-lvl S*)
    (*conflicting S*)

definition
  remove-cls :: *'v clause* ⇒ (*'v*, *nat*, *'v clause*) *twl-state-abs* ⇒
  (*'v*, *nat*, *'v clause*) *twl-state-abs*
where
  remove-cls C S =
   TWL-State (*trail S*) (*filter-mset* (λ*D*. *raw-clause D* ≠ *C*) (*init-clss S*))
    (*filter-mset* (λ*D*. *raw-clause D* ≠ *C*) (*learned-clss S*)) (*backtrack-lvl S*)
    (*conflicting S*)

definition *init-state* :: *'v clauses* ⇒ (*'v*, *nat*, *'v clause*) *twl-state-abs* **where**
  init-state N = *fold add-init-cls* (*linearize N*) (*TWL-State* [] {#} {#} *0 None*)

lemma *unchanged-fold-add-init-cls*:
  trail (*fold add-init-cls Cs* (*TWL-State M N U k C*)) = *M*
  learned-clss (*fold add-init-cls Cs* (*TWL-State M N U k C*)) = *U*
  backtrack-lvl (*fold add-init-cls Cs* (*TWL-State M N U k C*)) = *k*
  conflicting (*fold add-init-cls Cs* (*TWL-State M N U k C*)) = *C*
  by (*induct Cs arbitrary*: *N*) (*auto simp*: *add-init-cls-def*)

lemma *unchanged-init-state*[*simp*]:
  trail (*init-state N*) = []
  learned-clss (*init-state N*) = {#}
  backtrack-lvl (*init-state N*) = *0*
  conflicting (*init-state N*) = *None*
  unfolding *init-state-def* **by** (*rule unchanged-fold-add-init-cls*)+

lemma *clauses-init-fold-add-init*:
  no-dup M ⟹
  image-mset raw-clause (*init-clss* (*fold add-init-cls Cs* (*TWL-State M N U k C*))) =
   mset Cs + *image-mset raw-clause N*
  by (*induct Cs arbitrary*: *N*) (*auto simp*: *add.assoc add-init-cls-def clause-watch*)

lemma *init-clss-init-state*[*simp*]: *image-mset raw-clause* (*init-clss* (*init-state N*)) = *N*
  unfolding *init-state-def* **by** (*simp add*: *clauses-init-fold-add-init linearize*)

definition *restart'* **where**
  restart' S = *TWL-State* [] (*init-clss S*) (*restart-learned S*) *0 None*
end

9.4 Instanciation of the previous locale

definition *watch-nat* :: (*nat*, *nat*, *nat clause*) *twl-state-abs* ⇒ *nat clause* ⇒
nat clause twl-clause **where**
watch-nat S C =
(*let*
C' = *remdups* (*sorted-list-of-set* (*set-mset C*));
negation-not-assigned = *filter* (λ*L*. −*L* ∉ *lits-of* (*trail S*)) C';
negation-assigned-sorted-by-trail = *filter* (λ*L*. *L* ∈# *C*) (*map* (λ*L*. −*lit-of L*) (*trail S*));
W = *take 2* (*negation-not-assigned* @ *negation-assigned-sorted-by-trail*);
UW = *sorted-list-of-multiset* (*C* − *mset W*)
in TWL-Clause (*mset W*) (*mset UW*))

lemma *list-cases2*:
fixes *l* :: *'a list*
assumes
l = [] ⟹ *P* **and**
⋀*x*. *l* = [*x*] ⟹ *P* **and**
⋀*x y xs*. *l* = *x* # *y* # *xs* ⟹ *P*
shows *P*
by (*metis assms list.collapse*)

lemma *filter-in-list-prop-verifiedD*:
assumes [*L*←*P* . *Q L*] = *l*
shows ∀ *x* ∈ *set l*. *x* ∈ *set P* ∧ *Q x*
using *assms* **by** *auto*

lemma *no-dup-filter-diff*:
assumes *n-d*: *no-dup M* **and** *H*: [*L*←*map* (λ*L*. − *lit-of L*) *M*. *L* ∈# *C*] = *l*
shows *distinct l*
unfolding *H*[*symmetric*]
apply (*rule distinct-filter*)
using *n-d* **by** (*induction M*) *auto*

lemma *watch-nat-lists-disjointD*:
assumes
l: [*L*←*remdups* (*sorted-list-of-set* (*set-mset C*)) . − *L* ∉ *lits-of* (*trail S*)] = *l* **and**
l': [*L*←*map* (λ*L*. − *lit-of L*) (*trail S*) . *L* ∈# *C*] = l'
shows ∀ *x* ∈ *set l*. ∀ *y* ∈ *set* l'. *x* ≠ *y*
by (*auto simp*: *l*[*symmetric*] l'[*symmetric*] *lits-of-def*)

lemma *watch-nat-list-cases-witness*[*consumes 2*, *case-names nil-nil nil-single nil-other*
single-nil single-other other]:
fixes
C :: *'v literal multiset* **and**
C' :: *'v literal list* **and**
S :: ((*'v*, *'b*, *'c*) *ann-literal*, *'d*, *'e*, *'f*) *twl-state*
defines
xs ≡ [*L*←*remdups* C'. − *L* ∉ *lits-of* (*trail S*)] **and**
ys ≡ [*L*←*map* (λ*L*. − *lit-of L*) (*trail S*) . *L* ∈# *C*]
assumes
n-d: *no-dup* (*trail S*) **and**
C': *set* C' = *set-mset C* **and**
nil-nil: *xs* = [] ⟹ *ys* = [] ⟹ *P* **and**
nil-single:

$\bigwedge a.\ xs = [] \Longrightarrow ys = [a] \Longrightarrow\ a \in\#\ C \Longrightarrow P$ **and**
nil-other: $\bigwedge a\ b\ ys'.\ xs = [] \Longrightarrow ys = a\ \#\ b\ \#\ ys' \Longrightarrow a \neq b \Longrightarrow P$ **and**
single-nil: $\bigwedge a.\ xs = [a] \Longrightarrow ys = [] \Longrightarrow P$ **and**
single-other: $\bigwedge a\ b\ ys'.\ xs = [a] \Longrightarrow ys = b\ \#\ ys' \Longrightarrow a \neq b \Longrightarrow P$ **and**
other: $\bigwedge a\ b\ xs'.\ xs = a\ \#\ b\ \#\ xs' \Longrightarrow a \neq b \Longrightarrow P$
shows P
proof −
note *xs-def*[*simp*] **and** *ys-def*[*simp*]
have *dist*: *distinct* [L←*remdups* C' . − $L \notin$ *lits-of* (*trail* S)]
by *auto*
then have H: $\bigwedge a\ xs.$ [L←*remdups* C' . − $L \notin$ *lits-of* (*trail* S)]
$\neq a\ \#\ a\ \#\ xs$
by *force*
show *?thesis*
apply (*cases* [L←*remdups* C'. − $L \notin$ *lits-of* (*trail* S)]
rule: *list-cases2*;
cases [L←*map* ($\lambda L.$ − *lit-of* L) (*trail* S) . $L \in\#\ C$] *rule*: *list-cases2*)
using *nil-nil* **apply** *simp*
using *nil-single* **apply** (*force dest*: *filter-in-list-prop-verifiedD*)
using *nil-other*
apply (*auto dest*: *filter-in-list-prop-verifiedD watch-nat-lists-disjointD*
no-dup-filter-diff[*OF n-d*] *simp*: H)[]
using *single-nil* **apply** *simp*
using *single-other* C' *xs-def ys-def* **apply** (*smt imageE image-eqI list.set-intros*(*1*) *lits-of-def*
mem-Collect-eq set-filter set-map uminus-of-uminus-id)
using *single-other* C' **unfolding** *xs-def ys-def* **apply** (*smt imageE image-eqI list.set-intros*(*1*)
lits-of-def mem-Collect-eq set-filter set-map uminus-of-uminus-id)
using *other xs-def ys-def* **by** (*metis* H)+
qed

lemma *watch-nat-list-cases* [*consumes 1*, *case-names nil-nil nil-single nil-other single-nil*
single-other other]:
fixes
C :: $'v$::*linorder literal multiset* **and**
S :: (($'v$, $'b$, $'c$) *ann-literal*, $'d$, $'e$, $'f$) *twl-state*
defines
$xs \equiv$ [L←*remdups* (*sorted-list-of-set* (*set-mset* C)) . − $L \notin$ *lits-of* (*trail* S)] **and**
$ys \equiv$ [L←*map* ($\lambda L.$ − *lit-of* L) (*trail* S) . $L \in\#\ C$]
assumes
n-d: *no-dup* (*trail* S) **and**
nil-nil: $xs = [] \Longrightarrow ys = [] \Longrightarrow P$ **and**
nil-single:
$\bigwedge a.\ xs = [] \Longrightarrow ys = [a] \Longrightarrow\ a \in\#\ C \Longrightarrow P$ **and**
nil-other: $\bigwedge a\ b\ ys'.\ xs = [] \Longrightarrow ys = a\ \#\ b\ \#\ ys' \Longrightarrow a \neq b \Longrightarrow P$ **and**
single-nil: $\bigwedge a.\ xs = [a] \Longrightarrow ys = [] \Longrightarrow P$ **and**
single-other: $\bigwedge a\ b\ ys'.\ xs = [a] \Longrightarrow ys = b\ \#\ ys' \Longrightarrow a \neq b \Longrightarrow P$ **and**
other: $\bigwedge a\ b\ xs'.\ xs = a\ \#\ b\ \#\ xs' \Longrightarrow a \neq b \Longrightarrow P$
shows P
using *watch-nat-list-cases-witness*[*OF n-d*, *of sorted-list-of-set* (*set-mset* C) C P]
nil-nil nil-single nil-other single-nil single-other other
unfolding *xs-def*[*symmetric*] *ys-def*[*symmetric*] **by** *auto*

lemma *watch-nat-lists-set-union-witness*:
fixes
C :: $'v$ *literal multiset* **and**

C' :: $'v$ *literal list* **and**
S :: $(('v, 'b, 'c)$ *ann-literal*, $'d$, $'e$, $'f)$ *twl-state*
defines
xs $\equiv$ [$L\leftarrow$*remdups* C'. $-$ $L \notin$ *lits-of* (*trail* S)] **and**
ys $\equiv$ [$L\leftarrow$*map* (λL. $-$ *lit-of* L) (*trail* S) . $L \in\#$ C]
assumes *n-d*: *no-dup* (*trail* S) **and** C': *set* C' = *set-mset* C
shows *set-mset* C = *set xs* $\cup$ *set ys*
using *n-d* C' *uminus-lit-swap* **unfolding** *xs-def ys-def* **by** (*auto simp*: *lits-of-def*)

lemma *watch-nat-lists-set-union*:
fixes
C :: $'v$::*linorder literal multiset* **and**
S :: $(('v, 'b, 'c)$ *ann-literal*, $'d$, $'e$, $'f)$ *twl-state*
defines
xs $\equiv$ [$L\leftarrow$*remdups* (*sorted-list-of-set* (*set-mset* C)). $-$ $L \notin$ *lits-of* (*trail* S)] **and**
ys $\equiv$ [$L\leftarrow$*map* (λL. $-$ *lit-of* L) (*trail* S) . $L \in\#$ C]
assumes *n-d*: *no-dup* (*trail* S)
shows *set-mset* C = *set xs* $\cup$ *set ys*
using *watch-nat-lists-set-union-witness*[*of* S (*sorted-list-of-set* (*set-mset* C)) C, *OF n-d*]
sorted-list-of-set xs-def ys-def **by** *blast*

lemma *mset-intersection-inclusion*: $A + (B - A) = B \longleftrightarrow A \subseteq\# B$
apply (*rule iffI*)
apply (*metis mset-le-add-left*)
by (*auto simp*: *ac-simps multiset-eq-iff subseteq-mset-def*)

lemma *clause-watch-nat*:
assumes *no-dup* (*trail* S)
shows *raw-clause* (*watch-nat* S C) = C
using *assms*
apply (*cases rule*: *watch-nat-list-cases*[*OF assms*(*1*), *of* C])
by (*auto dest*: *filter-in-list-prop-verifiedD simp*: *watch-nat-def Let-def*
mset-intersection-inclusion subseteq-mset-def)

lemma *set-mset-is-single-in-mset-is-single*:
set-mset $C = \{a\} \Longrightarrow x \in\# C \Longrightarrow x = a$
by *fastforce*

lemma *index-uminus-index-map-uminus*:
$-a \in$ *set* $L \Longrightarrow$ *index* L $(-a)$ = *index* (*map uminus* L) (a::$'a$ *literal*)
by (*induction* L) *auto*

lemma *index-filter*:
$a \in$ *set* $L \Longrightarrow b \in$ *set* $L \Longrightarrow P\ a \Longrightarrow P\ b \Longrightarrow$
index L $a \leq$ *index* L $b \longleftrightarrow$ *index* (*filter* P L) $a \leq$ *index* (*filter* P L) b
by (*induction* L) *auto*

lemma *wf-watch-witness*:
fixes C :: $'a$ *literal multiset* **and** C':: $'a$ *literal list* **and**
S :: $(('a, 'b, 'c)$ *ann-literal*, $'d$, $'e$, $'f)$ *twl-state*
defines
ass: *negation-not-assigned* $\equiv$ *filter* (λL. $-L \notin$ *lits-of* (*trail* S)) (*remdups* C') **and**
tr: *negation-assigned-sorted-by-trail* $\equiv$ *filter* (λL. $L \in\# C$) (*map* (λL. $-$*lit-of* L) (*trail* S))
defines

W: $W \equiv$ *take 2* (*negation-not-assigned* @ *negation-assigned-sorted-by-trail*)
assumes
n-d[*simp*]: *no-dup* (*trail S*) **and**
C': *set C'* = *set-mset C*
shows *wf-twl-cls* (*trail S*) (*TWL-Clause* (*mset W*) (*C* − *mset W*))
unfolding *wf-twl-cls.simps*
proof (*intro conjI*, *goal-cases*)
case *1*
then show *?case* **using** *n-d C' W* **unfolding** *ass tr*
by (*cases rule*: *watch-nat-list-cases-witness*[*of S C' C*])
(*auto dest*: *filter-in-list-prop-verifiedD*
simp: *distinct-mset-add-single*)
next
case *2*
then show *?case* **unfolding** *W* **by** *simp*
next
case *3*
then show *?case* **using** *n-d C'*
proof (*cases rule*: *watch-nat-list-cases-witness*[*of S C' C*])
case *nil-nil*
then have *set-mset C* = *set* [] ∪ *set* []
using *C' watch-nat-lists-set-union-witness n-d* **by** *metis*
then show *?thesis*
by *simp*
next
case (*nil-single a*)
then show *?thesis*
using *watch-nat-lists-set-union-witness*[*of S C' C*] *C' 3*
by (*auto dest*!: *arg-cong*[*of* - [] *set*] *simp*: *W ass tr*)
next
case *nil-other*
then show *?thesis*
using *3* **by** (*auto dest*!: *arg-cong*[*of* - [] *set*] *simp*: *W ass tr*)
next
case *single-nil*
show *?thesis*
using *watch-nat-lists-set-union-witness*[*of S C' C*] *C' 3 mset-leD*
by (*auto simp*: *W ass tr single-nil*)
next
case *single-other*
then show *?thesis*
using *3* **by** (*auto dest*!: *arg-cong*[*of* - [] *set*] *simp*: *W ass tr*)
next
case *other*
then show *?thesis*
using *3* **by** (*auto dest*!: *arg-cong*[*of* - [] *set*] *simp*: *W ass tr*)
qed
next
case *4* **note** -[*simp*] = *this*
{
fix *a* :: *'a literal* **and** *ys'* :: *'a literal list* **and** *L* :: *'a literal* **and**
L' :: *'a literal*
assume *a1*: [*L*←*remdups C'*. − *L* ∉ *lits-of* (*trail S*)] = [*a*]
assume *a2*: *set-mset C* = *insert L* (*insert a* (*set ys'*))
assume *a3*: *L'* ∈# *C*

assume *a4*: $a \neq L'$
have *set* $(L \# a \# ys') =$ *set-mset* C
using *a2* **by** *auto*
then have $L' \notin$ *set* $[l\leftarrow$*remdups* $C'. - l \notin$ *lits-of* (*trail* S)]
using *a4 a1* **by** (*metis list.set*(*1*) *list.set*(*2*) *singleton-iff*)
then have $- L' \in$ *lits-of* (*trail* S)
using *a3* C' **by** *simp*
} **note** H =*this*
show *?case*
using *n-d* C' **apply** (*cases rule*: *watch-nat-list-cases-witness*[*of* S C' C])
apply (*auto dest*: *filter-in-list-prop-verifiedD*
simp: *W ass tr lits-of-def* C' *filter-empty-conv*)[*4*]
using *watch-nat-lists-set-union-witness*[*of* S C' C] C'
by (*auto dest*: *filter-in-list-prop-verifiedD H simp*: *W ass tr*)
next
case *5*
from *n-d* C' **show** *?case*
proof (*cases rule*: *watch-nat-list-cases-witness*[*of* S C' C])
case *nil-nil*
then show *?thesis* **by** (*auto simp*: *W ass tr*)
next
case *nil-single*
then show *?thesis*
using *watch-nat-lists-set-union-witness*[*of* S C' C] C' **by** (*auto simp*: *W ass tr*)
next
case *nil-other*
then show *?thesis*
unfolding *watched-decided-most-recently.simps Ball-mset-def*
apply (*intro allI impI*)
apply (*subst index-uminus-index-map-uminus*,
simp add: *index-uminus-index-map-uminus lits-of-def o-def*)
apply (*subst index-uminus-index-map-uminus*,
simp add: *index-uminus-index-map-uminus lits-of-def o-def*)

apply (*subst index-filter*[*of* - - - $\lambda L.\ L \in\#\ C$])
by (*auto dest*: *filter-in-list-prop-verifiedD*
simp: *uminus-lit-swap lits-of-def o-def W ass tr*)
next
case *single-nil*
then show *?thesis*
using *watch-nat-lists-set-union-witness*[*of* S C' C] C' **by** (*auto simp*: *W ass tr*)
next
case *single-other*
then show *?thesis*
unfolding *watched-decided-most-recently.simps Ball-mset-def*
apply (*clarify*)
apply (*subst index-uminus-index-map-uminus*,
simp add: *index-uminus-index-map-uminus lits-of-def o-def*)
apply (*subst index-uminus-index-map-uminus*,
simp add: *index-uminus-index-map-uminus lits-of-def o-def*)

apply (*subst index-filter*[*of* - - - $\lambda L.\ L \in\#\ C$])
by (*auto dest*: *filter-in-list-prop-verifiedD*
simp: *W ass tr uminus-lit-swap lits-of-def o-def*)
next

case *other*
then show *?thesis*
unfolding *watched-decided-most-recently.simps*
apply *clarify*
apply (*subst index-uminus-index-map-uminus*,
simp add: *index-uminus-index-map-uminus lits-of-def o-def*)[*1*]
apply (*subst index-uminus-index-map-uminus*,
simp add: *index-uminus-index-map-uminus lits-of-def o-def*)[*1*]

apply (*subst index-filter*[*of* - - - $\lambda L.\ L \in\#\ C$])
by (*auto dest*: *filter-in-list-prop-verifiedD*
simp: *index-uminus-index-map-uminus lits-of-def o-def uminus-lit-swap*
W ass tr)
qed
qed

lemma *wf-watch-nat*: *no-dup* (*trail S*) $\Longrightarrow$ *wf-twl-cls* (*trail S*) (*watch-nat S C*)
using *wf-watch-witness*[*of S sorted-list-of-set* (*set-mset C*) *C*]
by (*metis List.finite-set mset-sorted-list-of-multiset set-sorted-list-of-multiset*
sorted-list-of-set watch-nat-def)

definition
rewatch-nat ::
(*nat*, *nat*, *nat literal multiset*) *ann-literal* $\Rightarrow$ (*nat*, *nat*, *nat clause*) *twl-state-abs* $\Rightarrow$
nat clause twl-clause $\Rightarrow$ *nat clause twl-clause*
where
rewatch-nat L S C =
(*if* $-$ *lit-of L* $\in\#$ *watched C then*
case filter ($\lambda L'.\ L' \notin\#$ *watched C* $\wedge$ $-$ $L' \notin$ *lits-of* (*L* # *trail S*))
(*sorted-list-of-multiset* (*unwatched C*)) *of*
[] $\Rightarrow$ *C*
| L' # - $\Rightarrow$
TWL-Clause (*watched C* $-$ {#$-$ *lit-of L*#} + {#L'#}) (*unwatched C* $-$ {#L'#} + {#$-$ *lit-of*
L#})
else
C)

lemma *clause-rewatch-witness*:
fixes *UW* :: *'a literal list* **and**
S :: ((*'a*, *'b*, *'c*) *ann-literal*, *'d*, *'e*, *'f*) *twl-state* **and**
L :: (*'a*, *'b*, *'c*) *ann-literal* **and** *C* :: *'a literal multiset twl-clause*
defines $C' \equiv$ (*if* $-$ *lit-of L* $\in\#$ *watched C then*
case filter ($\lambda L'.\ L' \notin\#$ *watched C* $\wedge$ $-$ $L' \notin$ *lits-of* (*L* # *trail S*)) *UW of*
[] $\Rightarrow$ *C*
| L' # - $\Rightarrow$
TWL-Clause (*watched C* $-$ {#$-$ *lit-of L*#} + {#L'#}) (*unwatched C* $-$ {#L'#} + {#$-$ *lit-of*
L#})
else
C)
assumes
UW: *set UW* = *set-mset* (*unwatched C*)
shows *raw-clause* C' = *raw-clause C*
using *UW* **unfolding** *C'-def* **by** (*auto simp*: *subset-mset.add-diff-assoc2 multiset-eq-iff*
split: *list.split dest*: *filter-in-list-prop-verifiedD*)

lemma *clause-rewatch-nat*: *raw-clause* (*rewatch-nat L S C*) = *raw-clause C*
  using *clause-rewatch-witness*[*of sorted-list-of-multiset* (*unwatched C*) *C* - *S*]
  by (*auto simp*: *rewatch-nat-def Let-def split*: *list.split split-if-asm*)

lemma *filter-sorted-list-of-multiset-Nil*:
  $[x \leftarrow$ *sorted-list-of-multiset* $M.\ p\ x] = [] \longleftrightarrow (\forall x \in\# M.\ \neg\ p\ x)$
  by *auto* (*metis empty-iff filter-set list.set*(*1*) *mem-set-mset-iff member-filter*
    set-sorted-list-of-multiset)

lemma *filter-sorted-list-of-multiset-ConsD*:
  $[x \leftarrow$ *sorted-list-of-multiset* $M.\ p\ x] = x\ \#\ xs \Longrightarrow p\ x$
  by (*metis filter-set insert-iff list.set*(*2*) *member-filter*)

lemma *mset-minus-single-eq-mempty*:
  $a - \{\#b\#\} = \{\#\} \longleftrightarrow a = \{\#b\#\} \lor a = \{\#\}$
  by (*metis Multiset.diff-cancel add.right-neutral diff-single-eq-union*
    diff-single-trivial zero-diff)

lemma *size-mset-le-2-cases*:
  assumes *size* $W \leq 2$
  shows $W = \{\#\} \lor (\exists a.\ W = \{\#a\#\}) \lor (\exists a\ b.\ W = \{\#a,b\#\})$
  by (*metis One-nat-def Suc-1 Suc-eq-plus1-left assms linorder-not-less nat-less-le*
    not-less-eq-eq le-iff-add size-1-singleton-mset
    size-eq-0-iff-empty size-mset-2)

lemma *filter-sorted-list-of-multiset-eqD*:
  assumes $[x \leftarrow$ *sorted-list-of-multiset* $A.\ p\ x] = x\ \#\ xs$ (**is** *?comp* = -)
  shows $x \in\# A$
proof −
  have $x \in$ *set ?comp*
    using *assms* **by** *simp*
  then have $x \in$ *set* (*sorted-list-of-multiset A*)
    by *simp*
  then show $x \in\# A$
    by *simp*
qed

lemma *clause-rewatch-witness'*:
  fixes *UWC* :: *'a literal list* **and**
    S :: ((*'a*, *'b*, *'c*) *ann-literal*, *'d*, *'e*, *'f*) *twl-state* **and**
    L :: (*'a*, *'b*, *'c*) *ann-literal* **and** *C* :: *'a literal multiset twl-clause*
  defines $C' \equiv$ (*if* − *lit-of L* $\in\#$ *watched C then*
      case filter ($\lambda L'.\ L' \notin\#$ *watched C* $\land$ − $L' \notin$ *lits-of* (*L* # *trail S*)) *UWC of*
        [] ⇒ *C*
      | *L'* # - ⇒
        TWL-Clause (*watched C* − {#− *lit-of L*#} + {#*L'*#}) (*unwatched C* − {#*L'*#} + {#− *lit-of*
L#})
    else
      C)
  assumes
    UWC: *set UWC* = *set-mset* (*unwatched C*) **and**
    wf: *wf-twl-cls* (*trail S*) *C* **and**
    n-d: *no-dup* (*trail S*) **and**
    undef: *undefined-lit* (*trail S*) (*lit-of L*)
  shows *wf-twl-cls* (*L* # *trail S*) *C'*

```
proof (cases − lit-of L ∈# watched C)
  case False
  then have wf-twl-cls (L # trail S) C
    apply (cases C)
    using wf n-d undef apply (clarify)
    unfolding wf-twl-cls.simps
    apply (intro conjI)
         apply blast
        apply blast
       apply blast
      apply (smt ball-mset-cong bspec-mset insert-iff lits-of-cons nat-neq-iff twl-clause.sel(1)
        uminus-of-uminus-id)
     apply (auto simp: Marked-Propagated-in-iff-in-lits-of)
    done
  then show ?thesis
    using False C′-def by simp
next
  case falsified: True

  let ?unwatched-nonfalsified =
    [L′← UWC. L′ ∉# watched C ∧ − L′ ∉ lits-of (L # trail S)]
  obtain W UW where C: C = TWL-Clause W UW
    by (cases C)

  show ?thesis
  proof (cases ?unwatched-nonfalsified)
    case Nil
    show ?thesis
      using falsified Nil
      apply (simp only: wf-twl-cls.simps if-True list.cases C C′-def)
      apply (intro conjI)
      proof goal-cases
        case 1
        then show ?case using wf C by simp
      next
        case 2
        then show ?case using wf C by simp
      next
        case 3
        then show ?case using wf C by simp
      next
        case 4
        have ⋀p l. filter p UWC ≠ [] ∨ l ∉ set-mset UW ∨ ¬ p l
          using UWC unfolding C by (metis (no-types) filter-empty-conv twl-clause.sel(2))
        then show ?case
          using 4(2) unfolding Ball-mset-def by (metis (lifting) mem-set-mset-iff twl-clause.sel(1))
      next
        case 5
        then show ?case

          using C apply simp
          using wf by (smt ball-msetI bspec-mset not-gr0 uminus-of-uminus-id
            watched-decided-most-recently.simps wf-twl-cls.simps)
      qed
  next
```

case (*Cons* L' *Ls*)
show *?thesis*
  unfolding *rewatch-nat-def* C'*-def*
  using *falsified Cons*
  apply (*simp only*: *wf-twl-cls.simps if-True list.cases C*)
  apply (*intro conjI*)
  proof *goal-cases*
    case *1*
    have *distinct-mset* (*watched* (*TWL-Clause W UW*))
      using *wf* **unfolding** *C* **by** *auto*
    moreover have $L' \notin\#$ *watched* (*TWL-Clause W UW*) $-$ {#$-$ *lit-of L*#}
      using *1*(*2*) *not-gr0* **by** (*fastforce dest*: *filter-in-list-prop-verifiedD*)
    ultimately show *?case*
      by (*auto simp*: *distinct-mset-single-add*)
  next
    case *2*
    then show *?case* **using** *wf C* **by** (*metis insert-DiffM2 size-single size-union twl-clause.sel*(*1*)
      wf-twl-cls.simps)
  next
    case *3*
    then show *?case*
      using *wf C UWC* **by** (*force simp*: *mset-minus-single-eq-mempty dest*: *subset-singletonD*)
  next
    case *4*
    have *H*: $\forall L \in\# W.\ - L \in$ *lits-of* (*trail S*) $\longrightarrow$
     ($\forall L' \in\# UW.$ *count W* $L' = 0 \longrightarrow - L' \in$ *lits-of* (*trail S*))
      using *wf* **by** (*auto simp*: *C*)
    have *W*: *size W* $\leq$ *2* **and** *W-UW*: *size W* $<$ *2* $\longrightarrow$ *set-mset UW* $\subseteq$ *set-mset W*
      using *wf* **by** (*auto simp*: *C*)

    have *distinct*: *distinct-mset W*
      using *wf* **by** (*auto simp*: *C*)
    show *?case*
      using *4*
      unfolding *C watched-decided-most-recently.simps Ball-mset-def twl-clause.sel*
      apply (*intro allI impI*)
      apply (*rename-tac xW xUW*)
      apply (*case-tac* $-$ *lit-of L* $=$ *xW*; *case-tac xW* $=$ *xUW*; *case-tac* $L' = xW$)
              apply (*auto simp*: *uminus-lit-swap*)[*2*]
            apply (*force dest*: *filter-in-list-prop-verifiedD*)
           using *H size-mset-le-2-cases*[*OF W*]
          using *distinct* **apply** (*fastforce split*: *split-if-asm simp*: *distinct-mset-size-2*)
         using *distinct* **apply** (*fastforce split*: *split-if-asm simp*: *distinct-mset-size-2*)
        using *distinct* **apply** (*fastforce split*: *split-if-asm simp*: *distinct-mset-size-2*)
       apply (*force dest*: *filter-in-list-prop-verifiedD*)
      using *size-mset-le-2-cases*[*OF W*] *H* **by** (*fastforce simp*: *uminus-lit-swap*
        dest: *filter-sorted-list-of-multiset-ConsD filter-sorted-list-of-multiset-eqD*)

  next
    case *5*
    have *H*: $\forall x.\ x \in\#\ W \longrightarrow - x \in$ *lits-of* (*trail S*) $\longrightarrow$ ($\forall x.\ x \in\#\ UW \longrightarrow$ *count W x* $= 0$
     $\longrightarrow - x \in$ *lits-of* (*trail S*))
      using *wf* **by** (*auto simp*: *C*)
    show *?case*
      unfolding *C watched-decided-most-recently.simps Ball-mset-def*

proof (*intro allI impI conjI*, *goal-cases*)
case (*1 xW x*)
show *?case*
proof (*cases* $-$ *lit-of* $L = xW$)
case *True*
then show *?thesis*
by (*cases* $xW = x$) (*auto simp*: *uminus-lit-swap*)
next
case *False* **note** LxW = *this*
have *f9*: $L' \in$ *set* [$l \leftarrow UWC$. $l \notin\#$ *watched* (*TWL-Clause W UW*)
$\wedge - l \notin$ *lits-of* (L # *trail S*)]
using *1*(*2*) *5* **by** *auto*
moreover then have *f11*: $- xW \in$ *lits-of* (*trail S*)
using *1*(*3*) LxW **unfolding** *lits-of-cons* **by** (*metis* (*no-types*) *insert-iff*
uminus-of-uminus-id)
moreover then have $xW \notin\# W$
using *f9 1*(*2*) *H* **by** (*auto simp*: *C UWC*)
ultimately have *False*
using *1* **by** *auto*
then show *?thesis*
by *fast*
qed
qed
qed
qed
qed

lemma *wf-rewatch-nat'*:
assumes
wf: *wf-twl-cls* (*trail S*) *C* **and**
n-d: *no-dup* (*trail S*) **and**
undef: *undefined-lit* (*trail S*) (*lit-of L*)
shows *wf-twl-cls* (L # *trail S*) (*rewatch-nat L S C*)
using *clause-rewatch-witness'*[*of sorted-list-of-multiset* (*unwatched C*) *C S L*]
assms **by** (*auto simp*: *rewatch-nat-def*)

interpretation *twl*: *abstract-twl watch-nat rewatch-nat sorted-list-of-multiset learned-clss*
apply *unfold-locales*
apply (*rule clause-watch-nat*; *simp*)
apply (*rule wf-watch-nat*; *simp*)
apply (*rule clause-rewatch-nat*)
apply (*rule wf-rewatch-nat'*; *simp*)
apply (*rule mset-sorted-list-of-multiset*)
apply (*rule subset-mset.order-refl*)
done

9.5 Interpretation for $cdcl_W.cdcl_W$

context *abstract-twl*
begin

9.5.1 Direct Interpretation

interpretation *rough-cdcl*: $state_W$ *trail raw-init-clss raw-learned-clss backtrack-lvl conflicting*

cons-trail tl-trail add-init-cls add-learned-cls remove-cls update-backtrack-lvl update-conflicting init-state restart′
apply *unfold-locales*
apply (*simp-all add*: *add-init-cls-def add-learned-cls-def clause-rewatch clause-watch cons-trail-def remove-cls-def restart′-def tl-trail-def*)
apply (*rule image-mset-subseteq-mono*[*OF restart-learned*])
done

interpretation *rough-cdcl*: $cdcl_W$ *trail raw-init-clss raw-learned-clss backtrack-lvl conflicting cons-trail tl-trail add-init-cls add-learned-cls remove-cls update-backtrack-lvl update-conflicting init-state restart′*
by *unfold-locales*

9.5.2 Opaque Type with Invariant

declare *rough-cdcl.state-simp*[*simp del*]

definition *cons-trail-twl* :: (*′v*, *nat*, *′v literal multiset*) *ann-literal* ⇒ *′v wf-twl* ⇒ *′v wf-twl*
where
cons-trail-twl L S ≡ *twl-of-rough-state* (*cons-trail L* (*rough-state-of-twl S*))

lemma *wf-twl-state-cons-trail*:
undefined-lit (*trail S*) (*lit-of L*) ⟹ *wf-twl-state S* ⟹ *wf-twl-state* (*cons-trail L S*)
unfolding *wf-twl-state-def* **by** (*auto simp*: *cons-trail-def wf-rewatch defined-lit-map*)

lemma *rough-state-of-twl-cons-trail*:
undefined-lit (*trail-twl S*) (*lit-of L*) ⟹
rough-state-of-twl (*cons-trail-twl L S*) = *cons-trail L* (*rough-state-of-twl S*)
using *rough-state-of-twl twl-of-rough-state-inverse wf-twl-state-cons-trail*
unfolding *cons-trail-twl-def* **by** *blast*

abbreviation *add-init-cls-twl* **where**
add-init-cls-twl C S ≡ *twl-of-rough-state* (*add-init-cls C* (*rough-state-of-twl S*))

lemma *wf-twl-add-init-cls*: *wf-twl-state S* ⟹ *wf-twl-state* (*add-init-cls L S*)
unfolding *wf-twl-state-def* **by** (*auto simp*: *wf-watch add-init-cls-def split*: *split-if-asm*)

lemma *rough-state-of-twl-add-init-cls*:
rough-state-of-twl (*add-init-cls-twl L S*) = *add-init-cls L* (*rough-state-of-twl S*)
using *rough-state-of-twl twl-of-rough-state-inverse wf-twl-add-init-cls* **by** *blast*

abbreviation *add-learned-cls-twl* **where**
add-learned-cls-twl C S ≡ *twl-of-rough-state* (*add-learned-cls C* (*rough-state-of-twl S*))

lemma *wf-twl-add-learned-cls*: *wf-twl-state S* ⟹ *wf-twl-state* (*add-learned-cls L S*)
unfolding *wf-twl-state-def* **by** (*auto simp*: *wf-watch add-learned-cls-def split*: *split-if-asm*)

lemma *rough-state-of-twl-add-learned-cls*:
rough-state-of-twl (*add-learned-cls-twl L S*) = *add-learned-cls L* (*rough-state-of-twl S*)
using *rough-state-of-twl twl-of-rough-state-inverse wf-twl-add-learned-cls* **by** *blast*

abbreviation *remove-cls-twl* **where**
remove-cls-twl C S ≡ *twl-of-rough-state* (*remove-cls C* (*rough-state-of-twl S*))

lemma *wf-twl-remove-cls*: *wf-twl-state S* ⟹ *wf-twl-state* (*remove-cls L S*)
unfolding *wf-twl-state-def* **by** (*auto simp*: *wf-watch remove-cls-def split*: *split-if-asm*)

lemma *rough-state-of-twl-remove-cls*:
rough-state-of-twl (*remove-cls-twl L S*) = *remove-cls L* (*rough-state-of-twl S*)
using *rough-state-of-twl twl-of-rough-state-inverse wf-twl-remove-cls* **by** *blast*

abbreviation *init-state-twl* **where**
init-state-twl N ≡ *twl-of-rough-state* (*init-state N*)

lemma *wf-twl-state-wf-twl-state-fold-add-init-cls*:
assumes *wf-twl-state S*
shows *wf-twl-state* (*fold add-init-cls N S*)
using *assms* **apply** (*induction N arbitrary*: *S*)
apply (*auto simp*: *wf-twl-state-def*)[]
by (*simp add*: *wf-twl-add-init-cls*)

lemma *wf-twl-state-epsilon-state*[*simp*]:
wf-twl-state (*TWL-State* [] {#} {#} *0 None*)
by (*auto simp*: *wf-twl-state-def*)

lemma *wf-twl-init-state*: *wf-twl-state* (*init-state N*)
unfolding *init-state-def* **by** (*auto intro*!: *wf-twl-state-wf-twl-state-fold-add-init-cls*)

lemma *rough-state-of-twl-init-state*:
rough-state-of-twl (*init-state-twl N*) = *init-state N*
by (*simp add*: *twl-of-rough-state-inverse wf-twl-init-state*)

abbreviation *tl-trail-twl* **where**
tl-trail-twl S ≡ *twl-of-rough-state* (*tl-trail* (*rough-state-of-twl S*))

lemma *wf-twl-state-tl-trail*: *wf-twl-state S* ⟹ *wf-twl-state* (*tl-trail S*)
by (*simp add*: *twl-of-rough-state-inverse wf-twl-init-state wf-twl-cls-wf-twl-cls-tl tl-trail-def wf-twl-state-def distinct-tl map-tl*)

lemma *rough-state-of-twl-tl-trail*:
rough-state-of-twl (*tl-trail-twl S*) = *tl-trail* (*rough-state-of-twl S*)
using *rough-state-of-twl twl-of-rough-state-inverse wf-twl-state-tl-trail* **by** *blast*

abbreviation *update-backtrack-lvl-twl* **where**
update-backtrack-lvl-twl k S ≡ *twl-of-rough-state* (*update-backtrack-lvl k* (*rough-state-of-twl S*))

lemma *wf-twl-state-update-backtrack-lvl*:
wf-twl-state S ⟹ *wf-twl-state* (*update-backtrack-lvl k S*)
unfolding *wf-twl-state-def* **by** *auto*

lemma *rough-state-of-twl-update-backtrack-lvl*:
rough-state-of-twl (*update-backtrack-lvl-twl k S*) = *update-backtrack-lvl k* (*rough-state-of-twl S*)
using *rough-state-of-twl twl-of-rough-state-inverse wf-twl-state-update-backtrack-lvl* **by** *fast*

abbreviation *update-conflicting-twl* **where**
update-conflicting-twl k S ≡ *twl-of-rough-state* (*update-conflicting k* (*rough-state-of-twl S*))

lemma *wf-twl-state-update-conflicting*:
wf-twl-state S ⟹ *wf-twl-state* (*update-conflicting k S*)
unfolding *wf-twl-state-def* **by** *auto*

lemma *rough-state-of-twl-update-conflicting*:
rough-state-of-twl (*update-conflicting-twl* *k* *S*) = *update-conflicting* *k*
(*rough-state-of-twl* *S*)
using *rough-state-of-twl* *twl-of-rough-state-inverse* *wf-twl-state-update-conflicting* **by** *fast*

abbreviation *raw-clauses-twl* **where**
raw-clauses-twl *S* ≡ *raw-clauses* (*rough-state-of-twl* *S*)

abbreviation *restart-twl* **where**
restart-twl *S* ≡ *twl-of-rough-state* (*restart'* (*rough-state-of-twl* *S*))

lemma *wf-wf-restart'*: *wf-twl-state* *S* ⟹ *wf-twl-state* (*restart'* *S*)
unfolding *restart'-def* *wf-twl-state-def* **apply** *standard*
apply *clarify*
apply (*rename-tac* *x*)
apply (*subgoal-tac* *wf-twl-cls* (*trail* *S*) *x*)
apply (*case-tac* *x*)
using *restart-learned* **by** *fastforce*+

lemma *rough-state-of-twl-restart-twl*:
rough-state-of-twl (*restart-twl* *S*) = *restart'* (*rough-state-of-twl* *S*)
by (*simp* *add*: *twl-of-rough-state-inverse* *wf-wf-restart'*)

interpretation $cdcl_W$*-twl-NOT*: *dpll-state*
λ*S*. *convert-trail-from-W* (*trail-twl* *S*)
raw-clauses-twl
λ*L* *S*. *cons-trail-twl* (*convert-ann-literal-from-NOT* *L*) *S*
λ*S*. *tl-trail-twl* *S*
λ*C* *S*. *add-learned-cls-twl* *C* *S*
λ*C* *S*. *remove-cls-twl* *C* *S*
apply *unfold-locales*
apply (*simp* *add*: *rough-state-of-twl-cons-trail*)
apply (*metis* *rough-state-of-twl-tl-trail* *rough-cdcl.tl-trail*)
apply (*metis* *rough-state-of-twl-add-learned-cls* *rough-cdcl.trail-add-cls*$_{NOT}$)
apply (*metis* *rough-state-of-twl-remove-cls* *rough-cdcl.trail-remove-cls*)
apply (*simp* *add*: *rough-state-of-twl-cons-trail*)
apply (*simp* *add*: *rough-state-of-twl-tl-trail*)
using *rough-cdcl.clauses-add-cls*$_{NOT}$ *rough-cdcl.clauses-def* *rough-state-of-twl-add-learned-cls*
apply *auto*[*1*]
using *rough-cdcl.clauses-def* *rough-cdcl.clauses-remove-cls* *rough-state-of-twl-remove-cls* **by** *auto*

interpretation $cdcl_W$*-twl*: $state_W$
trail-twl
init-clss-twl
learned-clss-twl
backtrack-lvl-twl
conflicting-twl
cons-trail-twl
tl-trail-twl
add-init-cls-twl
add-learned-cls-twl
remove-cls-twl
update-backtrack-lvl-twl

update-conflicting-twl
init-state-twl
restart-twl
apply *unfold-locales*
by (*simp-all add*: *rough-state-of-twl-cons-trail rough-state-of-twl-tl-trail*
rough-state-of-twl-add-init-cls rough-state-of-twl-add-learned-cls rough-state-of-twl-remove-cls
rough-state-of-twl-update-backtrack-lvl rough-state-of-twl-update-conflicting
rough-state-of-twl-init-state rough-state-of-twl-restart-twl
rough-cdcl.learned-clss-restart-state)

interpretation $cdcl_W$*-twl*: $cdcl_W$
trail-twl
init-clss-twl
learned-clss-twl
backtrack-lvl-twl
conflicting-twl
cons-trail-twl
tl-trail-twl
add-init-cls-twl
add-learned-cls-twl
remove-cls-twl
update-backtrack-lvl-twl
update-conflicting-twl
init-state-twl
restart-twl
by *unfold-locales*

sublocale $cdcl_W$
trail-twl
init-clss-twl
learned-clss-twl
backtrack-lvl-twl
conflicting-twl
cons-trail-twl
tl-trail-twl
add-init-cls-twl
add-learned-cls-twl
remove-cls-twl
update-backtrack-lvl-twl
update-conflicting-twl
init-state-twl
restart-twl
by (*rule* $cdcl_W$*-twl.*$cdcl_W$*-axioms*)

abbreviation *state-eq-twl* (**infix** $\sim TWL$ *51*) **where**
state-eq-twl S S' $\equiv$ *rough-cdcl.state-eq* (*rough-state-of-twl* S) (*rough-state-of-twl* S')
notation $cdcl_W$*-twl.state-eq* (**infix** $\sim$ *51*)
declare $cdcl_W$*-twl.state-simp*[*simp del*]
$cdcl_W$*-twl-NOT.state-simp*$_{NOT}$[*simp del*]

To avoid ambiguities:

no-notation *state-eq-twl* (**infix** $\sim$ *51*)

definition *propagate-twl* **where**
propagate-twl S S' $\longleftrightarrow$

$(\exists L\ C.\ (L,\ C) \in$ *candidates-propagate-twl S*
$\wedge\ S' \sim$ *cons-trail-twl* (*Propagated L C*) *S*
$\wedge$ *conflicting-twl S* = *None*)

lemma *propagate-twl-iff-propagate*:
assumes *inv*: *cdcl*$_W$*-twl.cdcl*$_W$*-all-struct-inv S*
shows *cdcl*$_W$*-twl.propagate S T* $\longleftrightarrow$ *propagate-twl S T* (**is** *?P* $\longleftrightarrow$ *?T*)
proof
assume *?P*
then obtain *C L* **where**
conflicting (*rough-state-of-twl S*) = *None* **and**
CL-Clauses: *C* + {#*L*#} ∈# *cdcl*$_W$*-twl.clauses S* **and**
tr-CNot: *trail-twl S* ⊨*as CNot C* **and**
undef-lot: *undefined-lit* (*trail-twl S*) *L* **and**
T ∼ *cons-trail-twl* (*Propagated L* (*C* + {#*L*#})) *S*
unfolding *cdcl*$_W$*-twl.propagate.simps* **by** *blast*
have *distinct-mset* (*C* + {#*L*#})
using *inv CL-Clauses* **unfolding** *cdcl*$_W$*-twl.cdcl*$_W$*-all-struct-inv-def*
cdcl$_W$*-twl.distinct-cdcl*$_W$*-state-def cdcl*$_W$*-twl.clauses-def distinct-mset-set-def*
by (*metis* (*no-types*, *lifting*) *add-gr-0 mem-set-mset-iff plus-multiset.rep-eq*)
then have *C-L-L*: *mset-set* (*set-mset* (*C* + {#*L*#}) − {*L*}) = *C*
by (*metis Un-insert-right add-diff-cancel-left′ add-diff-cancel-right′*
distinct-mset-set-mset-ident finite-set-mset insert-absorb2 mset-set.insert-remove
set-mset-single set-mset-union)
have (*L*, *C*+{#*L*#}) ∈ *candidates-propagate-twl S*
apply (*rule wf-candidates-propagate-complete*)
using *rough-state-of-twl* **apply** *auto*[]
using *CL-Clauses* **unfolding** *cdcl*$_W$*-twl.clauses-def* **apply** *auto*[]
apply *simp*
using *C-L-L tr-CNot* **apply** *simp*
using *undef-lot* **apply** *blast*
done
show *?T* **unfolding** *propagate-twl-def*
apply (*rule exI*[*of - L*], *rule exI*[*of - C*+{#*L*#}])
apply (*auto simp*: ‹(*L*, *C*+{#*L*#}) ∈ *candidates-propagate-twl S*›
‹*conflicting* (*rough-state-of-twl S*) = *None*›)
using ‹*T* ∼ *cons-trail-twl* (*Propagated L* (*C* + {#*L*#})) *S*› *cdcl*$_W$*-twl.state-eq-backtrack-lvl*
cdcl$_W$*-twl.state-eq-conflicting cdcl*$_W$*-twl.state-eq-init-clss*
cdcl$_W$*-twl.state-eq-learned-clss cdcl*$_W$*-twl.state-eq-trail rough-cdcl.state-eq-def* **by** *blast*
next
assume *?T*
then obtain *L C* **where**
LC: (*L*, *C*) ∈ *candidates-propagate-twl S* **and**
T: *T* ∼ *cons-trail-twl* (*Propagated L C*) *S* **and**
confl: *conflicting* (*rough-state-of-twl S*) = *None*
unfolding *propagate-twl-def* **by** *auto*
have [*simp*]: *C* − {#*L*#} + {#*L*#} = *C*
using *LC* **unfolding** *candidates-propagate-def*
by *clarify* (*metis add.commute add-diff-cancel-right′ count-diff insert-DiffM*
multi-member-last not-gr0 zero-diff)
have *C* ∈# *raw-clauses-twl S*
using *LC* **unfolding** *candidates-propagate-def rough-cdcl.clauses-def* **by** *auto*
then have *distinct-mset C*
using *inv* **unfolding** *cdcl*$_W$*-twl.cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-twl.distinct-cdcl*$_W$*-state-def*
cdcl$_W$*-twl.clauses-def distinct-mset-set-def rough-cdcl.clauses-def* **by** *auto*

then have *C-L-L*: *mset-set* (*set-mset* $C - \{L\}$) $=$ $C - \{\#L\#\}$
by (*metis* ‹$C - \{\#L\#\} + \{\#L\#\} = C$› *add-left-imp-eq diff-single-trivial distinct-mset-set-mset-ident finite-set-mset mem-set-mset-iff mset-set.remove multi-self-add-other-not-self union-commute*)

show *?P*
apply (*rule* $cdcl_W$*-twl.propagate.intros*[*of - trail-twl S init-clss-twl S learned-clss-twl S backtrack-lvl-twl S* $C-\{\#L\#\}$ *L*])
using *confl* **apply** *auto*[]
using *LC* **unfolding** *candidates-propagate-def* **apply** (*auto simp*: $cdcl_W$*-twl.clauses-def*)[]
using *wf-candidates-propagate-sound*[*OF - LC*] *rough-state-of-twl* **apply** (*simp add*: *C-L-L*)
using *wf-candidates-propagate-sound*[*OF - LC*] *rough-state-of-twl* **apply** *simp*
using *T* **unfolding** $cdcl_W$*-twl.state-eq-def rough-cdcl.state-eq-def* **by** *auto*
qed
no-notation *CDCL-Two-Watched-Literals.twl.state-eq-twl* (**infix** $\sim$*TWL 51*)
definition *conflict-twl* **where**
conflict-twl S S' $\longleftrightarrow$
($\exists$ *C*. $C \in$ *candidates-conflict-twl S*
$\wedge$ $S' \sim$ *update-conflicting-twl* (*Some C*) *S*
$\wedge$ *conflicting-twl S* $=$ *None*)

lemma *conflict-twl-iff-conflict*:
shows $cdcl_W$*-twl.conflict S T* $\longleftrightarrow$ *conflict-twl S T* (**is** *?C* $\longleftrightarrow$ *?T*)
proof
assume *?C*
then obtain *M N U k C* **where**
S: *rough-cdcl.state* (*rough-state-of-twl S*) $=$ (*M*, *N*, *U*, *k*, *None*) **and**
C: $C \in\#$ $cdcl_W$*-twl.clauses S* **and**
M-C: $M \models$*as CNot C* **and**
T: $T \sim$ *update-conflicting-twl* (*Some C*) *S*
by *auto*
have $C \in$ *candidates-conflict-twl S*
apply (*rule wf-candidates-conflict-complete*)
apply *simp*
using *C* **apply** (*auto simp*: $cdcl_W$*-twl.clauses-def*)[]
using *M-C S* **by** *auto*
moreover have $T \sim$ *twl-of-rough-state* (*update-conflicting* (*Some C*) (*rough-state-of-twl S*))
using *T* **unfolding** *rough-cdcl.state-eq-def* $cdcl_W$*-twl.state-eq-def* **by** *auto*
ultimately show *?T*
using *S* **unfolding** *conflict-twl-def* **by** *auto*
next
assume *?T*
then obtain *C* **where**
C: $C \in$ *candidates-conflict-twl S* **and**
T: $T \sim$ *update-conflicting-twl* (*Some C*) *S* **and**
confl: *conflicting-twl S* $=$ *None*
unfolding *conflict-twl-def* **by** *auto*
have $C \in\#$ $cdcl_W$*-twl.clauses S*
using *C* **unfolding** *candidates-conflict-def* $cdcl_W$*-twl.clauses-def* **by** *auto*
moreover have *trail-twl S* $\models$*as CNot C*
using *wf-candidates-conflict-sound*[*OF - C*] **by** *auto*
ultimately show *?C* **apply** −
apply (*rule* $cdcl_W$*-twl.conflict.conflict-rule*[*of - - - - - C*])
using *confl T* **unfolding** *rough-cdcl.state-eq-def* $cdcl_W$*-twl.state-eq-def* **by** *auto*
qed

inductive $cdcl_W$*-twl* :: *'v wf-twl* ⇒ *'v wf-twl* ⇒ *bool* **for** *S* :: *'v wf-twl* **where**
propagate: *propagate-twl S S'* ⟹ $cdcl_W$*-twl S S'* |
conflict: *conflict-twl S S'* ⟹ $cdcl_W$*-twl S S'* |
other: $cdcl_W$*-twl*.$cdcl_W$*-o S S'* ⟹ $cdcl_W$*-twl S S'*|
rf: $cdcl_W$*-twl*.$cdcl_W$*-rf S S'* ⟹ $cdcl_W$*-twl S S'*

lemma $cdcl_W$*-twl-iff-*$cdcl_W$:
  assumes $cdcl_W$*-twl*.$cdcl_W$*-all-struct-inv S*
  shows $cdcl_W$*-twl S T* ⟷ $cdcl_W$*-twl*.$cdcl_W$ *S T*
  by (*simp add*: *assms* $cdcl_W$*-twl*.$cdcl_W$*.simps* $cdcl_W$*-twl.simps conflict-twl-iff-conflict*
    propagate-twl-iff-propagate)

lemma *rtranclp-*$cdcl_W$*-twl-all-struct-inv-inv*:
  assumes $cdcl_W$*-twl*** *S T* **and** $cdcl_W$*-twl*.$cdcl_W$*-all-struct-inv S*
  shows $cdcl_W$*-twl*.$cdcl_W$*-all-struct-inv T*
  using *assms* **by** (*induction rule*: *rtranclp-induct*)
  (*simp-all add*: $cdcl_W$*-twl-iff-*$cdcl_W$ $cdcl_W$*-twl*.$cdcl_W$*-all-struct-inv-inv*)

lemma *rtranclp-*$cdcl_W$*-twl-iff-rtranclp-*$cdcl_W$:
  assumes $cdcl_W$*-twl*.$cdcl_W$*-all-struct-inv S*
  shows $cdcl_W$*-twl*** *S T* ⟷ $cdcl_W$*-twl*.$cdcl_W$** *S T* (**is** *?T* ⟷ *?W*)
proof
  assume *?W*
  then show *?T*
    proof (*induction rule*: *rtranclp-induct*)
      case *base*
      then show *?case* **by** *simp*
    next
      case (*step T U*) **note** *st* = *this*(*1*) **and** *cdcl* = *this*(*2*) **and** *IH* = *this*(*3*)
      have $cdcl_W$*-twl T U*
        using *assms st cdcl* $cdcl_W$*-twl.rtranclp-*$cdcl_W$*-all-struct-inv-inv* $cdcl_W$*-twl-iff-*$cdcl_W$
        by *blast*
      then show *?case* **using** *IH* **by** *auto*
    qed
next
  assume *?T*
  then show *?W*
    proof (*induction rule*: *rtranclp-induct*)
      case *base*
      then show *?case* **by** *simp*
    next
      case (*step T U*) **note** *st* = *this*(*1*) **and** *cdcl* = *this*(*2*) **and** *IH* = *this*(*3*)
      have $cdcl_W$*-twl*.$cdcl_W$ *T U*
        using *assms st cdcl rtranclp-*$cdcl_W$*-twl-all-struct-inv-inv* $cdcl_W$*-twl-iff-*$cdcl_W$
        by *blast*
      then show *?case* **using** *IH* **by** *auto*
    qed
qed

interpretation $cdcl_{NOT}$*-twl*: *backjumping-ops*
  λS. convert-trail-from-W (*trail-twl S*)
  abstract-twl.raw-clauses-twl
  λL (*S*:: *'v wf-twl*).
    cons-trail-twl

    (*convert-ann-literal-from-NOT L*) (*S*:: *'v wf-twl*)
  tl-trail-twl
  add-learned-cls-twl
  remove-cls-twl
  λC - - (*S*:: *'v wf-twl*) -. $C \in$ *candidates-conflict-twl S*
  by *unfold-locales*

lemma *reduce-trail-to*$_{NOT}$*-skip-beginning-twl*:
  assumes *trail-twl S* = *convert-trail-from-NOT* (*F'* @ *F*)
  shows *trail-twl* (*cdcl*$_W$*-twl.reduce-trail-to*$_{NOT}$ *F S*) = *convert-trail-from-NOT F*
  using *assms* **by** (*induction F' arbitrary*: *S*) *auto*

lemma *reduce-trail-to*$_{NOT}$*-trail-tl-trail-twl-decomp*[*simp*]:
  trail-twl S = *convert-trail-from-NOT* (*F'* @ *Marked K* () # *F*) $\Longrightarrow$
    trail-twl (*cdcl*$_W$*-twl.reduce-trail-to*$_{NOT}$ *F* (*tl-trail-twl S*)) = *convert-trail-from-NOT F*
  apply (*rule reduce-trail-to*$_{NOT}$*-skip-beginning-twl*[*of - tl* (*F'* @ *Marked K* () # [])])
  by (*cases F'*) (*auto simp add*:*tl-append rough-cdcl.reduce-trail-to*$_{NOT}$*-skip-beginning*)

lemma *trail-twl-reduce-trail-to*$_{NOT}$*-drop*:
  trail-twl (*cdcl*$_W$*-twl.reduce-trail-to*$_{NOT}$ *F S*) =
   (*if length* (*trail-twl S*) $\geq$ *length F*
   then drop (*length* (*trail-twl S*) $-$ *length F*) (*trail-twl S*)
   else [])
  apply (*induction F S rule*: *cdcl*$_W$*-twl.reduce-trail-to*$_{NOT}$*.induct*)
  apply (*rename-tac F S*)
  apply (*case-tac trail-twl S*)
   apply *auto*[]
  apply (*rename-tac list*)
  apply (*case-tac Suc* (*length list*) > *length F*)
   prefer *2* **apply** *simp*
  apply (*subgoal-tac Suc* (*length list*) $-$ *length F* = *Suc* (*length list* $-$ *length F*))
   apply *simp*
  apply *simp*
  done

interpretation *cdcl*$_{NOT}$*-twl*: *dpll-with-backjumping-ops*
  λS. *convert-trail-from-W* (*trail-twl S*)
  abstract-twl.raw-clauses-twl
  λL *S*.
   cons-trail-twl
    (*convert-ann-literal-from-NOT L*) *S*
  tl-trail-twl
  add-learned-cls-twl
  remove-cls-twl
  λL *S*. *lit-of L* $\in$ *fst* ‘ *candidates-propagate-twl S*
  λS. *no-dup* (*trail-twl S*)
  λC - - *S* -. $C \in$ *candidates-conflict-twl S*
proof (*unfold-locales*, *goal-cases*)
  case (*1 C' S C F' K F L*) **note** *n-d* = *this*(*1*) **and** *n-d'* = *this*(*2*) **and** *undef* = *this*(*6*)
  let *?T'* = (*cons-trail* (*Propagated L* {#}) (*rough-state-of-twl* (*cdcl*$_W$*-twl.reduce-trail-to*$_{NOT}$ *F S*)))
  let *?T* = (*cons-trail-twl* (*Propagated L* {#}) (*cdcl*$_W$*-twl.reduce-trail-to*$_{NOT}$ *F S*))
  have *tr-F-S*: *map lit-of* (*trail-twl* (*cdcl*$_W$*-twl.reduce-trail-to*$_{NOT}$ *F S*)) =
   map lit-of (*convert-trail-from-NOT F*)
   apply (*subst trail-twl-reduce-trail-to*$_{NOT}$*-drop*[*of F S*])
   using *1*(*1*) *arg-cong*[*OF 1*(*3*), *of length*] *arg-cong*[*OF 1*(*3*), *of map lit-of*]

by (*auto simp*: *o-def drop-map*[*symmetric*])

have *no-dup* (*trail-twl S*)
using *1*(*1*) **by** *blast*
have *wf-twl-state* (*rough-state-of-twl* ($cdcl_W$*-twl.reduce-trail-to*$_{NOT}$ *F S*))
using *wf-twl-state-rough-state-of-twl* **by** *blast*
moreover have *undef'*: *undefined-lit* (*trail-twl* ($cdcl_W$*-twl.reduce-trail-to*$_{NOT}$ *F S*)) *L*
using *undef arg-cong*[*OF tr-F-S*, *of map atm-of*] **unfolding** *defined-lit-map image-set*
by (*simp add*: *o-def*)
ultimately have *wf-twl-state ?T'*
by (*simp-all add*: *wf-twl-state-cons-trail*)
then have *init-clss-twl ?T* = *init-clss-twl* ($cdcl_W$*-twl.reduce-trail-to*$_{NOT}$ *F S*)
using *1*(*6*) **by** (*simp add*: *undef'*)
then have [*simp*]: *init-clss-twl ?T* = *init-clss-twl S*
by (*simp add*: $cdcl_W$*-twl.reduce-trail-to*$_{NOT}$*-reduce-trail-convert*)

have *learned-clss-twl ?T* = *learned-clss-twl* ($cdcl_W$*-twl.reduce-trail-to*$_{NOT}$ *F S*)
by (*simp add*: *undef'*)
moreover have *learned-clss-twl* ($cdcl_W$*-twl.reduce-trail-to*$_{NOT}$ *F S*)
= *learned-clss-twl S*
by (*simp add*: $cdcl_W$*-twl.reduce-trail-to*$_{NOT}$*-reduce-trail-convert*)
ultimately have [*simp*]: *learned-clss-twl ?T* = *learned-clss-twl S*
by *simp*
have *tr-L-F-S*: *map lit-of* (*trail-twl ?T*)
= *map lit-of* (*Propagated L* {#} # *convert-trail-from-NOT F*)
using *undef' tr-F-S* **by** (*simp add*: *o-def*)
have *C-confl-cand*: *C* ∈ *candidates-conflict-twl S*
apply(*rule wf-candidates-twl-conflict-complete*)
using *1*(*1*,*4*) **apply** (*simp add*: *rough-cdcl.clauses-def*)
using *1*(*5*) **by** (*simp add*: *tr-L-F-S true-annots-true-cls lits-of-convert-trail-from-NOT*)

have $cdcl_{NOT}$*-twl.backjump S*
(*cons-trail-twl* (*convert-ann-literal-from-NOT* (*Propagated L* ()))
($cdcl_W$*-twl.reduce-trail-to*$_{NOT}$ *F S*))
apply (*rule* $cdcl_{NOT}$*-twl.backjump.intros*[*of S F' K F - L C*, *OF 1*(*3*) *- 1*(*4–6*) *- 1*(*8–9*)])
unfolding $cdcl_W$*-twl-NOT.state-eq*$_{NOT}$*-def* **apply** (*metis convert-ann-literal-from-NOT.simps*(*1*))
using *1*(*7*) *1*(*3*) **apply** *presburger*
using *C-confl-cand* **by** *simp*
then show *?case*
by *blast*
qed

interpretation $cdcl_{NOT}$*-twl*: *dpll-with-backjumping*
λS. *convert-trail-from-W* (*trail-twl S*)
abstract-twl.raw-clauses-twl
λL (*S*:: *'v wf-twl*).
cons-trail-twl
(*convert-ann-literal-from-NOT L*) (*S*:: *'v wf-twl*)
tl-trail-twl
add-learned-cls-twl
remove-cls-twl
λL *S*. *lit-of L* ∈ *fst* ‘ *candidates-propagate-twl S*
λS. *no-dup* (*trail-twl S*)
λC - - (*S*:: *'v wf-twl*) -. *C* ∈ *candidates-conflict-twl S*
apply *unfold-locales*

using *cdcl*$_{NOT}$*-twl.dpll-bj-no-dup* **by** (*simp add*: *o-def*)
end

end

10 Implementation for 2 Watched-Literals

theory *CDCL-Two-Watched-Literals-Implementation*
imports *CDCL-Two-Watched-Literals DPLL-CDCL-W-Implementation*
begin

type-synonym *'v conc-twl-state* =
((*'v*, *nat*, *'v literal list*) *ann-literal*, *'v literal list twl-clause list*, *nat*, *'v literal list*)
twl-state

fun *convert* :: (*'a*, *'b*, *'c list*) *ann-literal* ⇒ (*'a*, *'b*, *'c multiset*) *ann-literal* **where**
convert (*Propagated L C*) = *Propagated L* (*mset C*) |
convert (*Marked K i*) = *Marked K i*

abbreviation *convert-tr* :: (*'a*, *'b*, *'c list*) *ann-literals* ⇒ (*'a*, *'b*, *'c multiset*) *ann-literals*
where
convert-tr ≡ *map convert*

abbreviation *convertC* :: *'a literal list option* ⇒ *'a clause option* **where**
convertC ≡ *map-option mset*

fun *raw-clause-l* :: *'v list twl-clause* ⇒ *'v multiset twl-clause* **where**
raw-clause-l (*TWL-Clause UW W*) = *TWL-Clause* (*mset W*) (*mset UW*)

abbreviation *convert-clss* :: *'v literal list twl-clause list* ⇒ *'v clause twl-clause multiset*
where
convert-clss S ≡ *mset* (*map raw-clause-l S*)

fun *raw-state-of-conc* :: *'v conc-twl-state* ⇒ (*'v*, *nat*, *'v clause*) *twl-state-abs* **where**
raw-state-of-conc (*TWL-State M N U k C*) =
TWL-State (*convert-tr M*) (*convert-clss N*) (*convert-clss U*) *k* (*map-option mset C*)

lemma
raw-state-of-conc (*tl-trail S*) = *tl-trail* (*raw-state-of-conc S*)
unfolding *tl-trail-def* **by** (*induction S*) (*auto simp*: *map-tl*)

typedef *'v conv-twl-state* = {*S*:: *'v conc-twl-state*. *wf-twl-state* (*raw-state-of-conc S*)}
morphisms *list-twl-state-of cls-twl-state*
proof −
have *TWL-State* [] [] [] *0 None* ∈ {*S*:: *'v conc-twl-state*. *wf-twl-state* (*raw-state-of-conc S*)}
by (*auto simp*: *wf-twl-state-def*)
then show *?thesis* **by** *blast*
qed
term *list-twl-state-of*

definition *watch-list* :: *'v conv-twl-state* ⇒ *'v literal list* ⇒ *'v literal list twl-clause* **where**
watch-list S' C =
(*let*
M = *trail* (*list-twl-state-of S'*);
C' = *remdups C*;

$negation\text{-}not\text{-}assigned = filter\ (\lambda L.\ -L \notin lits\text{-}of\ M)\ C';$
$negation\text{-}assigned\text{-}sorted\text{-}by\text{-}trail = filter\ (\lambda L.\ L \in set\ C)\ (map\ (\lambda L.\ -lit\text{-}of\ L)\ M);$
$W = take\ 2\ (negation\text{-}not\text{-}assigned\ @\ negation\text{-}assigned\text{-}sorted\text{-}by\text{-}trail);$
$UW = foldl\ (\lambda a\ l.\ remove1\ l\ a)\ C\ W$
in $TWL\text{-}Clause\ W\ UW)$

lemma *wf-watch-nat*: $no\text{-}dup\ (trail\ (list\text{-}twl\text{-}state\text{-}of\ S)) \Longrightarrow$
$wf\text{-}twl\text{-}cls\ (trail\ (list\text{-}twl\text{-}state\text{-}of\ S))\ (raw\text{-}clause\text{-}l\ (watch\text{-}list\ S\ C))$
apply (*simp only*: *watch-list-def Let-def raw-clause-l.simps*)
using *wf-watch-witness*[*of* (*list-twl-state-of S*) *C mset C*]
oops

end