Formalisation of Ground Resolution and CDCL in Isabelle/HOL

Mathias Fleury and Jasmin Blanchette

April 16, 2016

Contents

0.1 Partial Clausal Logic

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

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

begin

0.1.1 Decided Literals

Definition

datatype $('v, 'mark)$ *ann-lit* =
  is-decided: *Decided* (*lit-of*: *'v literal*) |
  is-proped: *Propagated* (*lit-of*: *'v literal*) (*mark-of*: *'mark*)

lemma *ann-lit-list-induct*[*case-names Nil Decided Propagated*]:
  assumes P [] **and**
  $\bigwedge L\ xs.\ P\ xs \Longrightarrow P\ (Decided\ L\ \#\ xs)$ **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-decided-ex-Decided*:
  $is\text{-}decided\ L \Longrightarrow (\bigwedge K.\ L = Decided\ K \Longrightarrow P) \Longrightarrow P$
  by (*cases L*) *auto*

type-synonym $('v, 'm)$ *ann-lits* = $('v, 'm)$ *ann-lit list*

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

abbreviation *lits-of-l* :: $('a, 'b)$ *ann-lits* ⇒ *'a literal set* **where**
lits-of-l Ls ≡ *lits-of* (*set Ls*)

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

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

lemma *lits-of-l-Un*[*simp*]:
  lits-of (l ∪ l') = *lits-of* l ∪ *lits-of* l'
  unfolding *lits-of-def* **by** *auto*

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

abbreviation *unmark* **where**
unmark ≡ (λ*a*. {#*lit-of a*#})

abbreviation *unmark-s* **where**
unmark-s M ≡ *unmark* ‘ *M*

abbreviation *unmark-l* **where**
unmark-l M ≡ *unmark-s* (*set M*)

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

lemma *lits-of-l-empty-is-empty*[*iff*]:
  lits-of-l M = {} ⟷ *M* = []
  by (*induct M*) (*auto simp*: *lits-of-def*)

Entailment

definition *true-annot* :: ('*a*, '*m*) *ann-lits* ⇒ '*a clause* ⇒ *bool* (**infix** ⊨*a* *49*) **where**
  I ⊨*a* *C* ⟷ (*lits-of-l I*) ⊨ *C*

definition *true-annots* :: ('*a*, '*m*) *ann-lits* ⇒ '*a clauses* ⇒ *bool* (**infix** ⊨*as* *49*) **where**
  I ⊨*as* *CC* ⟷ (∀ *C* ∈ *CC*. *I* ⊨*a* *C*)

lemma *true-annot-empty-model*[*simp*]:
  ¬[] ⊨*a* ψ
  unfolding *true-annot-def true-cls-def* **by** *simp*

lemma *true-annot-empty*[*simp*]:
  ¬*I* ⊨*a* {#}
  unfolding *true-annot-def true-cls-def* **by** *simp*

lemma *empty-true-annots-def*[*iff*]:
  [] ⊨*as* ψ ⟷ ψ = {}
  unfolding *true-annots-def* **by** *auto*

lemma *true-annots-empty*[*simp*]:
  I ⊨*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*:
$I \models as\ CC \longleftrightarrow lits\text{-}of\text{-}l\ I \models s\ CC$
unfolding *true-annots-def Ball-def true-annot-def true-clss-def* **by** *auto*

lemma *in-lit-of-true-annot*:
$a \in lits\text{-}of\text{-}l\ M \longleftrightarrow M \models a\ \{\#a\#\}$
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*:
$I \models s\ unmark\text{-}l\ MLs \Longrightarrow lits\text{-}of\text{-}l\ MLs \subseteq I$
unfolding *true-clss-def lits-of-def* **by** *auto*

lemma *true-annot-true-clss-cls*:
$MLs \models a\ \psi \Longrightarrow set\ (map\ unmark\ MLs) \models p\ \psi$
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*:
$MLs \models as\ \psi \Longrightarrow set\ (map\ unmark\ MLs) \models ps\ \psi$
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-decided-true-cls*[*iff*]:
$map\ Decided\ M \models as\ N \longleftrightarrow set\ M \models s\ N$
proof –
have $*$: $lit\text{-}of$ ` $Decided$ ` $set\ M = set\ M$ **unfolding** *lits-of-def* **by** *force*
show *?thesis* **by** (*simp add*: *true-annots-true-cls* $*$ *lits-of-def*)

qed

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

lemma *true-annots-true-clss-clss*:
$A \models as\ \Psi \Longrightarrow$ *unmark-l* $A \models ps\ \Psi$
unfolding *true-clss-clss-def true-annots-def true-clss-def*
by (*auto dest*!: *true-clss-singleton-lit-of-implies-incl*
simp: *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*: *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*

Defined and undefined literals

We introduce the functions *defined-lit* and *undefined-lit* to know whether a literal is defined with respect to a list of decided literals (aka a trail in most cases).

Remark that *undefined* already exists and is a completely different Isabelle function.

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

abbreviation *undefined-lit* :: $('a, 'm)$ *ann-lits* $\Rightarrow$ *'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-decided-or-proped*:
assumes $x \in$ *set* I
shows
(*Decided* ($-$ *lit-of* x) $\in$ *set* I)
$\vee$ (*Decided* (*lit-of* x) $\in$ *set* I)
$\vee$ ($\exists l$. *Propagated* ($-$ *lit-of* x) $l \in$ *set* I)
$\vee$ ($\exists l$. *Propagated* (*lit-of* x) $l \in$ *set* I)
using *assms ann-lit.exhaust-sel* **by** *metis*

lemma *literal-is-lit-of-decided*:

assumes $L = \textit{lit-of } x$
shows $(x = \textit{Decided } L) \vee (\exists\, l'.\ x = \textit{Propagated } L\ l')$
using *assms* **by** (*cases x*) *auto*

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

lemma *consistent-inter-true-annots-satisfiable*:
consistent-interp (*lits-of-l 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* $\in$ ($\lambda l.$ *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-decided-or-proped*)

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

lemma *Decided-Propagated-in-iff-in-lits-of-l*:
defined-lit I L $\longleftrightarrow$ ($L \in$ *lits-of-l I* $\vee$ $-L \in$ *lits-of-l I*)
unfolding *lits-of-def* **by** (*metis lits-of-def true-annot-iff-decided-or-true-lit true-lit-def*)

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

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

0.1.2 Backtracking

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

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

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

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

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

lemma *backtrack-split-some-is-decided-then-snd-has-hd*:
$\exists\, l{\in}set\ M.$ *is-decided* $l \Longrightarrow \exists\, M'\ L'\ M''.$ *backtrack-split* $M = (M'', L' \# M')$
by (*metis backtrack-snd-empty-not-decided 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-decided*) M, *dropWhile* (*Not o is-decided*) M)
by (*induction M rule*: *ann-lit-list-induct*) *auto*

0.1.3 Decomposition with respect to the First Decided Literals

In this section we define a function that returns a decomposition with the first decided literal. This function is useful to define the backtracking of DPLL.

Definition

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

fun *get-all-ann-decomposition* :: ('*a*, '*m*) *ann-lits*
⇒ (('*a*, '*m*) *ann-lits* × ('*a*, '*m*) *ann-lits*) *list* **where**
get-all-ann-decomposition (*Decided L # Ls*) =
(*Decided L # Ls*, []) # *get-all-ann-decomposition Ls* |
get-all-ann-decomposition (*Propagated L P# Ls*) =
(*apsnd* ((*op #*) (*Propagated L P*)) (*hd* (*get-all-ann-decomposition Ls*)))
tl (*get-all-ann-decomposition Ls*) |
get-all-ann-decomposition [] = [([], [])]

value *get-all-ann-decomposition* [*Propagated A5 B5*, *Decided C4*, *Propagated A3 B3*,
Propagated A2 B2, *Decided C1*, *Propagated A0 B0*]

Now we can prove several simple properties about the function.

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

lemma *get-all-ann-decomposition-never-empty-sym*[*iff*]:
[] = *get-all-ann-decomposition* M ⟷ *False*
using *get-all-ann-decomposition-never-empty*[*of M*] **by** *presburger*

lemma *get-all-ann-decomposition-decomp*:
hd (*get-all-ann-decomposition* S) = $(a, c) \Longrightarrow S = c\ @\ a$
proof (*induct S arbitrary*: *a c*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons x A*)
then show *?case* **by** (*cases x*; *cases hd* (*get-all-ann-decomposition A*)) *auto*
qed

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

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

This functions says that the first element is either empty or starts with a decided element of the list.

lemma *get-all-ann-decomposition-length-1-fst-empty-or-length-1*:
  assumes *get-all-ann-decomposition* $M = (a, b)$ # []
  shows $a = [] \lor$ (*length* $a = 1 \land$ *is-decided* (*hd a*) $\land$ *hd* $a \in$ *set* M)
  using *assms*
proof (*induct M arbitrary*: *a b rule*: *ann-lit-list-induct*)
  case *Nil* **then show** *?case* **by** *simp*
next
  case (*Decided L mark*)
  then show *?case* **by** *simp*
next
  case (*Propagated L mark M*)
  then show *?case* **by** (*cases get-all-ann-decomposition M*) *force+*
qed

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

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

lemma *tl-get-all-ann-decomposition-skip-some*:
  assumes $x \in$ *set* (*tl* (*get-all-ann-decomposition M1*))
  shows $x \in$ *set* (*tl* (*get-all-ann-decomposition* (*M0* @ *M1*)))
  using *assms*
  by (*induct M0 rule*: *ann-lit-list-induct*)
    (*auto simp add*: *list.set-sel*(*2*))

lemma *hd-get-all-ann-decomposition-skip-some*:
  assumes $(x, y) =$ *hd* (*get-all-ann-decomposition M1*)
  shows $(x, y) \in$ *set* (*get-all-ann-decomposition* (*M0* @ *Decided K* # *M1*))

  using *assms*
proof (*induction M0 rule*: *ann-lit-list-induct*)
  case *Nil*
  then show *?case* **by** *auto*
next
  case (*Decided L M0*)
  then show *?case* **by** *auto*
next
  case (*Propagated L C M0*) **note** $xy = this(1)[OF\ this(2-)]$ **and** $hd = this(2)$
  then show *?case*
    by (*cases get-all-ann-decomposition* (*M0* @ *Decided K* # *M1*))
      (*auto dest*!: *get-all-ann-decomposition-decomp*
        arg-cong[*of get-all-ann-decomposition - - hd*])
qed

lemma *in-get-all-ann-decomposition-in-get-all-ann-decomposition-prepend*:
  $(a, b) \in set\ (get\text{-}all\text{-}ann\text{-}decomposition\ M') \Longrightarrow$
    $\exists\, b'.\ (a, b' @ b) \in set\ (get\text{-}all\text{-}ann\text{-}decomposition\ (M\ @\ M'))$
  apply (*induction M rule*: *ann-lit-list-induct*)
    apply (*metis append-Nil*)
   apply *auto*[]
  by (*rename-tac* L' *m xs*, *case-tac get-all-ann-decomposition* (*xs* @ M')) *auto*

lemma *in-get-all-ann-decomposition-decided-or-empty*:
  assumes $(a, b) \in set\ (get\text{-}all\text{-}ann\text{-}decomposition\ M)$
  shows $a = [] \lor (is\text{-}decided\ (hd\ a))$
  using *assms*
proof (*induct M arbitrary*: *a b rule*: *ann-lit-list-induct*)
  case *Nil* **then show** *?case* **by** *simp*
next
  case (*Decided l M*)
  then show *?case* **by** *auto*
next
  case (*Propagated l mark M*)
  then show *?case* **by** (*cases get-all-ann-decomposition M*) *force+*
qed

lemma *get-all-ann-decomposition-remove-undecided-length*:
  assumes $\forall\, l \in set\ M'.\ \neg is\text{-}decided\ l$
  shows $length\ (get\text{-}all\text{-}ann\text{-}decomposition\ (M'\ @\ M'')) = length\ (get\text{-}all\text{-}ann\text{-}decomposition\ M'')$
  using *assms* **by** (*induct* M' *arbitrary*: M'' *rule*: *ann-lit-list-induct*) *auto*

lemma *get-all-ann-decomposition-not-is-decided-length*:
  assumes $\forall\, l \in set\ M'.\ \neg is\text{-}decided\ l$
  shows $1 + length\ (get\text{-}all\text{-}ann\text{-}decomposition\ (Propagated\ (-L)\ P\ \#\ M))$
 $= length\ (get\text{-}all\text{-}ann\text{-}decomposition\ (M'\ @\ Decided\ L\ \#\ M))$
 using *assms get-all-ann-decomposition-remove-undecided-length* **by** *fastforce*

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

lemma *get-all-ann-decomposition-except-last-choice-equal*:

assumes $\forall l \in$ *set* M'. $\neg$*is-decided* l
shows *tl* (*get-all-ann-decomposition* (*Propagated* ($-L$) P # M))
= *tl* (*tl* (*get-all-ann-decomposition* (M' @ *Decided* L # M)))
using *assms* **by** (*induct* M' *rule*: *ann-lit-list-induct*) *auto*

lemma *get-all-ann-decomposition-hd-hd*:
assumes *get-all-ann-decomposition* *Ls* = (M, C) # (*M0*, *M0'*) # l
shows *tl* M = *M0'* @ *M0* $\wedge$ *is-decided* (*hd* M)
using *assms*
proof (*induct* *Ls* *arbitrary*: M C *M0* *M0'* l)
case *Nil*
then show *?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 = *Decided* L
have *Ls* = *M0'* @ *M0*
using g a **by** (*force intro*: *get-all-ann-decomposition-decomp*)
then have *tl* M = *M0'* @ *M0* $\wedge$ *is-decided* (*hd* M) **using** g a **by** *auto*
}
moreover {
fix L P
assume a: a = *Propagated* L P
have *tl* M = *M0'* @ *M0* $\wedge$ *is-decided* (*hd* M)
using *IH* *Cons.prems* **unfolding** a **by** (*cases get-all-ann-decomposition Ls*) *auto*
}
ultimately show *?case* **by** (*cases* a) *auto*
qed

lemma *get-all-ann-decomposition-exists-prepend*[*dest*]:
assumes (a, b) $\in$ *set* (*get-all-ann-decomposition* M)
shows $\exists$ c. M = c @ b @ a
using *assms* **apply** (*induct* M *rule*: *ann-lit-list-induct*)
apply *simp*
by (*rename-tac* L' *xs*, *case-tac get-all-ann-decomposition xs*;
auto dest!: *arg-cong*[*of get-all-ann-decomposition - - hd*]
get-all-ann-decomposition-decomp)+

lemma *get-all-ann-decomposition-incl*:
assumes (a, b) $\in$ *set* (*get-all-ann-decomposition* M)
shows *set* $b \subseteq$ *set* M **and** *set* $a \subseteq$ *set* M
using *assms get-all-ann-decomposition-exists-prepend* **by** *fastforce*+

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

lemma *union-in-get-all-ann-decomposition-is-subset*:
assumes (a, b) $\in$ *set* (*get-all-ann-decomposition* M)
shows *set* $a \cup$ *set* $b \subseteq$ *set* M
using *assms* **by** *force*

lemma *Decided-cons-in-get-all-ann-decomposition-append-Decided-cons*:
  $\exists$ *M1 M2*. (*Decided K* # *M1*, *M2*) $\in$ *set* (*get-all-ann-decomposition* (*c* @ *Decided K* # *c'*))
  apply (*induction c rule*: *ann-lit-list-induct*)
    apply *auto*[*2*]
  apply (*rename-tac L xs*,
      case-tac hd (*get-all-ann-decomposition* (*xs* @ *Decided K* # *c'*)))
  apply (*case-tac get-all-ann-decomposition* (*xs* @ *Decided K* # *c'*))
  by *auto*

lemma *fst-get-all-ann-decomposition-prepend-not-decided*:
  assumes $\forall$ *m*$\in$*set MS*. $\neg$ *is-decided m*
  shows *set* (*map fst* (*get-all-ann-decomposition M*))
    = *set* (*map fst* (*get-all-ann-decomposition* (*MS* @ *M*)))
    using *assms* **apply** (*induction MS rule*: *ann-lit-list-induct*)
    apply *auto*[*2*]
    by (*rename-tac L m xs*; *case-tac get-all-ann-decomposition* (*xs* @ *M*)) *simp-all*

Entailment of the Propagated by the Decided Literal

lemma *get-all-ann-decomposition-snd-union*:
  set M = $\bigcup$(*set* ' *snd* ' *set* (*get-all-ann-decomposition M*)) $\cup$ {*L* |*L*. *is-decided L* $\wedge$ *L* $\in$ *set M*}
  (**is** *?M M* = *?U M* $\cup$ *?Ls M*)
proof (*induct M rule*: *ann-lit-list-induct*)
  case *Nil*
  then show *?case* **by** *simp*
next
  case (*Decided L M*) **note** *IH* = *this*(*1*)
  then have *Decided L* $\in$ *?Ls* (*Decided L* #*M*) **by** *auto*
  moreover have *?U* (*Decided L* #*M*) = *?U M* **by** *auto*
  moreover have *?M M* = *?U M* $\cup$ *?Ls M* **using** *IH* **by** *auto*
  ultimately show *?case* **by** *auto*
next
  case (*Propagated L m M*)
  then show *?case* **by** (*cases* (*get-all-ann-decomposition M*)) *auto*
qed

definition *all-decomposition-implies* :: *'a literal multiset set*
  $\Rightarrow$ ((*'a*, *'m*) *ann-lits* $\times$ (*'a*, *'m*) *ann-lits*) *list* $\Rightarrow$ *bool* **where**
 all-decomposition-implies N S $\longleftrightarrow$ ($\forall$(*Ls*, *seen*) $\in$ *set S*. *unmark-l Ls* $\cup$ *N* $\models$*ps unmark-l 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*)] $\longleftrightarrow$ *unmark-l Ls* $\cup$ *N* $\models$*ps unmark-l seen*
  unfolding *all-decomposition-implies-def* **by** *auto*

lemma *all-decomposition-implies-append*[*iff*]:
  all-decomposition-implies N (*S* @ *S'*)
    $\longleftrightarrow$ (*all-decomposition-implies N S* $\wedge$ *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'*)
    $\longleftrightarrow$ (*all-decomposition-implies N* [(*Ls*, *seen*)] $\wedge$ *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') $\longleftrightarrow$
    (*unmark-l* (*fst* l) $\cup$ N $\models$*ps* *unmark-l* (*snd* l) $\wedge$
      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-ann-decomposition* M)
  shows $N \cup \{$*unmark* L |L. *is-decided* $L \wedge L \in$ *set* $M\}$
    $\models$*ps* *unmark* ‘ $\bigcup$(*set* ‘ *snd* ‘ *set* (*get-all-ann-decomposition* M))
using *assms*
proof (*induct length* (*get-all-ann-decomposition* M) *arbitrary*: M)
  case *0*
  then show *?case* **by** *auto*
next
  case (*Suc n*) **note** *IH* = *this*(*1*) **and** *length* = *this*(*2*) **and** *decomp* = *this*(*3*)
  consider
      (*le1*) *length* (*get-all-ann-decomposition* M) $\leq$ *1*
    | (*gt1*) *length* (*get-all-ann-decomposition* M) $>$ *1*
    by *arith*
  then show *?case*
    proof *cases*
      case *le1*
      then obtain a b **where** g: *get-all-ann-decomposition* M = (a, b) # []
        by (*cases get-all-ann-decomposition* M) *auto*
      moreover {
        assume a = []
        then have *?thesis* **using** *Suc.prems* g **by** *auto*
      }
      moreover {
        assume l: *length* a = *1* **and** m: *is-decided* (*hd* a) **and** *hd*: *hd* $a \in$ *set* M
        then have *unmark* (*hd* a) $\in \{$*unmark* L |L. *is-decided* $L \wedge L \in$ *set* $M\}$ **by** *auto*
        then have H: *unmark-l* $a \cup N \subseteq N \cup \{$*unmark* L |L. *is-decided* $L \wedge L \in$ *set* $M\}$
          using l **by** (*cases* a) *auto*
        have *f1*: *unmark-l* $a \cup N$ $\models$*ps* *unmark-l* b
          using *decomp* **unfolding** *all-decomposition-implies-def* g **by** *simp*
        have *?thesis*
          apply (*rule true-clss-clss-subset*) **using** *f1* H g **by** *auto*
      }
      ultimately show *?thesis*
        using *get-all-ann-decomposition-length-1-fst-empty-or-length-1* **by** *blast*
    next
      case *gt1*
      then obtain *Ls0 seen0* M' **where**
        Ls0: *get-all-ann-decomposition* M = (*Ls0*, *seen0*) # *get-all-ann-decomposition* M' **and**
        length': *length* (*get-all-ann-decomposition* M') = n **and**
        M'*-in-*M: *set* $M' \subseteq$ *set* M
        using *length* **by** (*induct* M *rule*: *ann-lit-list-induct*) (*auto simp*: *subset-insertI2*)
      let *?d* = $\bigcup$(*set* ‘ *snd* ‘ *set* (*get-all-ann-decomposition* M'))
      let *?unM* = $\{$*unmark* L |L. *is-decided* $L \wedge L \in$ *set* $M\}$
      let *?unM'* = $\{$*unmark* L |L. *is-decided* $L \wedge L \in$ *set* $M'\}$
      {
        assume n = *0*
        then have *get-all-ann-decomposition* M' = [] **using** *length'* **by** *auto*
        then have *?thesis* **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-ann-decomposition M′* = (*Ls1*, *seen1*) # *l*
using *length′* **by** (*induct M′ rule*: *ann-lit-list-induct*) *auto*

have *all-decomposition-implies N* (*get-all-ann-decomposition M′*)
using *decomp* **unfolding** *Ls0* **by** *auto*
then have *N*: *N* ∪ *?unM′* ⊨*ps unmark-s ?d*
using *IH length′* **by** *auto*
have *l*: *N* ∪ *?unM′* ⊆ *N* ∪ *?unM*
using *M′-in-M* **by** *auto*
from *true-clss-clss-subset*[*OF this N*]
have Ψ*N*: *N* ∪ *?unM* ⊨*ps unmark-s ?d* **by** *auto*
have *is-decided* (*hd Ls0*) **and** *LS*: *tl Ls0* = *seen1* @ *Ls1*
using *get-all-ann-decomposition-hd-hd*[*of M*] **unfolding** *Ls0 Ls1* **by** *auto*

have *LSM*: *seen1* @ *Ls1* = *M′* **using** *get-all-ann-decomposition-decomp*[*of M′*] *Ls1* **by** *auto*
have *M′*: *set M′* = *?d* ∪ {*L* |*L*. *is-decided L* ∧ *L* ∈ *set M′*}
using *get-all-ann-decomposition-snd-union* **by** *auto*

{
assume *Ls0* ≠ []
then have *hd Ls0* ∈ *set M*
using *get-all-ann-decomposition-fst-empty-or-hd-in-M Ls0* **by** *blast*
then have *N* ∪ *?unM* ⊨*p unmark* (*hd Ls0*)
using ‹*is-decided* (*hd Ls0*)› **by** (*metis* (*mono-tags*, *lifting*) *UnCI mem-Collect-eq true-clss-cls-in*)
} **note** *hd-Ls0* = *this*

have *l*: *unmark* ‘ (*?d* ∪ {*L* |*L*. *is-decided L* ∧ *L* ∈ *set M′*}) = *unmark-s ?d* ∪ *?unM′*
by *auto*
have *N* ∪ *?unM′* ⊨*ps unmark* ‘ (*?d* ∪ {*L* |*L*. *is-decided L* ∧ *L* ∈ *set M′*})
unfolding *l* **using** *N* **by** (*auto simp*: *all-in-true-clss-clss*)
then have *t*: *N* ∪ *?unM′* ⊨*ps unmark-l* (*tl Ls0*)
using *M′* **unfolding** *LS LSM* **by** *auto*
then have *N* ∪ *?unM* ⊨*ps unmark-l* (*tl Ls0*)
using *M′-in-M true-clss-clss-subset*[*OF - t*, *of N* ∪ *?unM*] **by** *auto*
then have *N* ∪ *?unM* ⊨*ps unmark-l Ls0*
using *hd-Ls0* **by** (*cases Ls0*) *auto*

moreover have *unmark-l Ls0* ∪ *N* ⊨*ps unmark-l seen0*
using *decomp* **unfolding** *Ls0* **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* ∪ *?unM* ⊨*ps unmark-l seen0*
by (*meson true-clss-clss-left-right true-clss-clss-union-and true-clss-clss-union-l-r*)

moreover have *unmark* ‘ (*set seen0* ∪ *?d*) = *unmark-l seen0* ∪ *unmark-s ?d*
by *auto*
ultimately have *?thesis* **using** Ψ*N* **unfolding** *Ls0* **by** *simp*
}
ultimately show *?thesis* **by** *auto*
qed
qed

lemma *all-decomposition-implies-propagated-lits-are-implied*:
  assumes *all-decomposition-implies N* (*get-all-ann-decomposition M*)
  shows $N \cup \{$*unmark L* $|$*L. is-decided L* $\wedge$ $L \in$ *set M*$\}$ $\models$*ps unmark-l M*
    (**is** *?I* $\models$*ps ?A*)
proof −
  have *?I* $\models$*ps unmark-s* $\{L\ |$*L. is-decided L* $\wedge$ $L \in$ *set M*$\}$
    by (*auto intro*: *all-in-true-clss-clss*)
  moreover have *?I* $\models$*ps unmark* ' $\bigcup$(*set* ' *snd* ' *set* (*get-all-ann-decomposition M*))
    using *all-decomposition-implies-trail-is-implied assms* **by** *blast*
  ultimately have $N \cup \{$*unmark m* $|$*m. is-decided m* $\wedge$ $m \in$ *set M*$\}$
    $\models$*ps unmark* ' $\bigcup$(*set* ' *snd* ' *set* (*get-all-ann-decomposition M*))
      $\cup$ *unmark* ' $\{m\ |$*m. is-decided m* $\wedge$ $m \in$ *set M*$\}$
      by *blast*
  then show *?thesis*
    by (*metis* (*no-types*) *get-all-ann-decomposition-snd-union*[*of M*] *image-Un*)
qed

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

0.1.4 Negation of Clauses

We define the negation of a *'a Partial-Clausal-Logic.clause*: it converts it from the a single clause to a set of clauses, wherein each clause is a single negated literal.

definition *CNot* :: *'v clause* $\Rightarrow$ *'v clauses* **where**
CNot ψ = { {#−*L*#} | *L. L* ∈# ψ }

lemma *in-CNot-uminus*[*iff*]:
  shows {#*L*#} ∈ *CNot* ψ $\longleftrightarrow$ −*L* ∈# ψ
  unfolding *CNot-def* **by** *force*

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

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

lemma *in-CNot-implies-uminus*:
  assumes *L* ∈# *D* **and** *M* $\models$*as CNot D*
  shows *M* $\models$*a* {#−*L*#} **and** −*L* ∈ *lits-of-l M*
  using *assms* **by** (*auto simp*: *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*]:
  (∀ *x*∈*CNot D. P x*) $\longleftrightarrow$ (∀ *L*∈# *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* φ
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* φ
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-l M*
by (*metis assms atm-of-uminus image-eqI in-CNot-implies-uminus*(*1*) *true-annot-singleton*)

lemma *true-annots-CNot-all-uminus-atms-defined*:
assumes $M \models as$ *CNot T* **and** *a1*: $-L \in\#\ T$
shows *atm-of* $L \in$ *atm-of* ‘ *lits-of-l 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#}} ∪ $B \models p$ {#}
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* ∪ *CNot* {#L#}) **and**
cons: *consistent-interp I* **and**
I: $I \models s$ *B*
have *total-over-m I* ({{#L#}} ∪ *B*) **using** *tot* **by** *auto*
then have $\neg I \models s$ *insert* {#L#} *B*
using *assms cons* **unfolding** *true-clss-cls-def* **by** *simp*
then show $I \models s$ *CNot* {#L#}
using *tot I* **by** (*cases L*) *auto*
qed

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

lemma *true-annot-CNot-diff*:
  $I \models as\ CNot\ C \Longrightarrow I \models as\ CNot\ (C - C')$
  by (*auto simp*: *true-annots-true-cls-def-iff-negation-in-model dest*: *in-diffD*)

lemma *CNot-mset-replicate*[*simp*]:
  $CNot\ (mset\ (replicate\ n\ L)) = (if\ n = 0\ then\ \{\}\ else\ \{\{\#-L\#\}\})$
  by (*induction n*) *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*

The following lemma is very useful when in the goal appears an axioms like $-L = K$: this lemma allows the simplifier to rewrite L.

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-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-interp ?I*
    using *cons* **unfolding** *consistent-interp-def*
    by (*auto simp*: *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*: *total-over-m-def total-over-set-def*)

  then have *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*
  then have $?I \models CC + \{\#L\#\}$ **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*

then have ¬*A* ⊨*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)
then have ¬*?I* ⊨ *CC* **using** ‹*?I* ⊨*s CNot CC*› *cons′ consistent-CNot-not* **by** *blast*
ultimately have *?I* ⊨ {#*L*#} **by** *blast*
then show *I* ⊨ {#*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* ⊨*as CNot A* **and** *LA*: *lit-of L* ∉# *A* −*lit-of L* ∉# *A*
shows *M* ⊨*as CNot A*
using *LM* **unfolding** *true-annots-def Ball-def*
proof (*intro allI impI*)
fix *l*
assume *H*: ∀ *x*. *x* ∈ *CNot A* ⟶ *L* # *M* ⊨*a x* **and** *l*: *l* ∈ *CNot A*
then have *L* # *M* ⊨*a l* **by** *auto*
then show *M* ⊨*a l* **using** *LA l* **by** (*cases L*) (*auto simp*: *CNot-def*)
qed

lemma *true-clss-clss-union-false-true-clss-clss-cnot*:
A ∪ {*B*} ⊨*ps* {{#}} ⟷ *A* ⊨*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′*⊨*a D* **and** *atm-of* (*lit-of a*) ∉ *atms-of D*
shows *M′* ⊨*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′*⊨*a D* **and** ∀ *x*∈*atms-of D*. *x* ∉ *atm-of* ‘ *lits-of-l M*
shows *M′* ⊨*a D*
using *assms* **by** (*induct M*) (*auto dest*: *true-annot-remove-hd-if-notin-vars*)

lemma *true-annots-remove-if-notin-vars*:
assumes *M* @ *M′*⊨*as D* **and** ∀ *x*∈*atms-of-ms D*. *x* ∉ *atm-of* ‘ *lits-of-l M*
shows *M′* ⊨*as D* **unfolding** *true-annots-def*
using *assms* **unfolding** *true-annots-def atms-of-ms-def*
by (*force dest*: *true-annot-remove-if-notin-vars*)

lemma *all-variables-defined-not-imply-cnot*:
assumes
∀ *s* ∈ *atms-of-ms* {*B*}. *s* ∈ *atm-of* ‘ *lits-of-l A* **and**
¬ *A* ⊨*a B*
shows *A* ⊨*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* ∈# *B* **and** ¬ *lits-of-l A* ⊨*l* − *L*
then have *atm-of L* ∈ *atm-of* ‘ *lits-of-l A*
using *assms*(*1*) **by** (*simp add*: *atm-of-lit-in-atms-of lits-of-def*)
then have *L* ∈ *lits-of-l A* ∨ −*L* ∈ *lits-of-l A*
using *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set* **by** *metis*
then have *L* ∈ *lits-of-l A* **using** ‹ ¬ *lits-of-l A* ⊨*l* − *L*› **by** *auto*

then show *False*
using *LB assms*(*2*) **unfolding** *true-annot-def true-lit-def true-cls-def Bex-def*
by *blast*
qed

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

0.1.5 Other

abbreviation *no-dup L* ≡ *distinct* (*map* (λ*l*. *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-l*:
assumes *no-dup M*
shows *length M* = *card* (*atm-of* ' *lits-of-l M*)
using *assms* **unfolding** *lits-of-def* **by** (*induct M*) (*auto simp add*: *image-image*)

lemma *distinct-consistent-interp*:
no-dup M ⟹ *consistent-interp* (*lits-of-l M*)
proof (*induct M*)
case *Nil*
show *?case* **by** *auto*
next
case (*Cons L M*)
then have *a1*: *consistent-interp* (*lits-of-l M*) **by** *auto*
have *a2*: *atm-of* (*lit-of L*) ∉ (λ*l*. *atm-of* (*lit-of l*)) ' *set M* **using** *Cons.prems* **by** *auto*
have *undefined-lit M* (*lit-of L*)
using *a2* **unfolding** *defined-lit-map* **by** *fastforce*
then show *?case*
using *a1* **by** *simp*
qed

lemma *distinct-get-all-ann-decomposition-no-dup*:
assumes (*a*, *b*) ∈ *set* (*get-all-ann-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* (*L* # *M*)
using *assms*(*1*) *in-CNot-implies-uminus*(*2*) **by** *blast*
moreover
have *atm-of* (*lit-of L*) ∉ *atm-of* ' *lits-of-l M*
using *assms*(*3*) **unfolding** *lits-of-def* **by** *force*
then have − *lit-of L* ∉ *lits-of-l M* **unfolding** *lits-of-def*
by (*metis* (*no-types*) *atm-of-uminus imageI*)

ultimately have $\forall\ l \in\#\ A.\ -l \in$ *lits-of-l M*
using *assms(2)* **by** (*metis insert-iff list.simps(15) lits-of-insert uminus-of-uminus-id*)
then show *?thesis* **by** (*auto simp add: true-annots-def*)
qed

0.1.6 Extending Entailments to multisets

We have defined previous entailment with respect to sets, but we also need a multiset version depending on the context. The conversion is simple using the function *set-mset* (in this direction, there is no loss of information).

abbreviation *true-annots-mset* (**infix** $\models$*asm 50*) **where**
$I \models$*asm* $C \equiv I \models$*as* (*set-mset C*)

abbreviation *true-clss-clss-m*:: *'v clause multiset* $\Rightarrow$ *'v clause multiset* $\Rightarrow$ *bool* (**infix** $\models$*psm 50*)
where
$I \models$*psm* $C \equiv$ *set-mset I* $\models$*ps* (*set-mset C*)

Analog of $[\![$*?N* $\models$*ps ?B*; *?A* $\subseteq$ *?B*$]\!] \Longrightarrow$ *?N* $\models$*ps ?A*

lemma *true-clss-clssm-subsetE*: $N \models$*psm* $B \Longrightarrow A \subseteq\# B \Longrightarrow N \models$*psm* A
using *set-mset-mono true-clss-clss-subsetE* **by** *blast*

abbreviation *true-clss-cls-m*:: *'a clause multiset* $\Rightarrow$ *'a clause* $\Rightarrow$ *bool* (**infix** $\models$*pm 50*) **where**
$I \models$*pm* $C \equiv$ *set-mset I* $\models$*p C*

abbreviation *distinct-mset-mset* :: *'a multiset multiset* $\Rightarrow$ *bool* **where**
distinct-mset-mset $\Sigma \equiv$ *distinct-mset-set* (*set-mset* Σ)

abbreviation *all-decomposition-implies-m* **where**
all-decomposition-implies-m A B $\equiv$ *all-decomposition-implies* (*set-mset A*) *B*

abbreviation *atms-of-mm* :: *'a literal multiset multiset* $\Rightarrow$ *'a set* **where**
atms-of-mm U $\equiv$ *atms-of-ms* (*set-mset U*)

Other definition using *Union-mset*

lemma *atms-of-mm U* $\equiv$ *set-mset* ($\bigcup\#$ *image-mset* (*image-mset atm-of*) *U*)
unfolding *atms-of-ms-def* **by** (*auto simp: atms-of-def*)

abbreviation *true-clss-m*:: *'a interp* $\Rightarrow$ *'a clause multiset* $\Rightarrow$ *bool* (**infix** $\models$*sm 50*) **where**
$I \models$*sm* $C \equiv I \models$*s set-mset C*

abbreviation *true-clss-ext-m* (**infix** $\models$*sextm 49*) **where**
$I \models$*sextm* $C \equiv I \models$*sext set-mset C*

end
theory *CDCL-Abstract-Clause-Representation*
imports *Main Partial-Clausal-Logic*
begin

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

0.1.7 Abstract Clause Representation

We will abstract the representation of clause and clauses via two locales. We expect our representation to behave like multiset, but the internal representation can be done using list or whatever other representation.

We assume the following:

- there is an equivalent to adding and removing a literal and to taking the union of clauses.

locale *raw-cls* =
  fixes
    mset-cls :: *'cls* ⇒ *'v clause*
begin
end

locale *raw-ccls-union* =
  fixes
    mset-cls :: *'cls* ⇒ *'v clause* **and**
    union-cls :: *'cls* ⇒ *'cls* ⇒ *'cls* **and**
    remove-clit :: *'v literal* ⇒ *'cls* ⇒ *'cls*
  assumes
    mset-ccls-union-cls[*simp*]: *mset-cls* (*union-cls C D*) = *mset-cls C* #∪ *mset-cls D* **and**
    remove-clit[*simp*]: *mset-cls* (*remove-clit L C*) = *remove1-mset L* (*mset-cls C*)
begin
end

Instantiation of the previous locale, in an unnamed context to avoid polluting with simp rules

context
begin
  interpretation *list-cls*: *raw-cls mset*
    by *unfold-locales*

  interpretation *cls-cls*: *raw-cls id*
    by *unfold-locales*

  interpretation *list-cls*: *raw-ccls-union mset*
    union-mset-list remove1
    by *unfold-locales* (*auto simp*: *union-mset-list ex-mset*)

  interpretation *cls-cls*: *raw-ccls-union id op* #∪ *remove1-mset*
    by *unfold-locales* (*auto simp*: *union-mset-list*)
end

Over the abstract clauses, we have the following properties:

- We can insert a clause
- We can take the union (used only in proofs for the definition of *clauses*)
- there is an operator indicating whether the abstract clause is contained or not
- if a concrete clause is contained the abstract clauses, then there is an abstract clause

locale *raw-clss* =
  raw-cls mset-cls
  for
    mset-cls :: *'cls* ⇒ *'v clause* +
  fixes

$mset\text{-}clss$:: $'clss \Rightarrow 'v\ clauses$ **and**
$union\text{-}clss$ :: $'clss \Rightarrow 'clss \Rightarrow 'clss$ **and**
$in\text{-}clss$ :: $'cls \Rightarrow 'clss \Rightarrow bool$ **and**
$insert\text{-}clss$ :: $'cls \Rightarrow 'clss \Rightarrow 'clss$ **and**
$remove\text{-}from\text{-}clss$ :: $'cls \Rightarrow 'clss \Rightarrow 'clss$
assumes
$insert\text{-}clss[simp]$: $mset\text{-}clss\ (insert\text{-}clss\ L\ C) = mset\text{-}clss\ C + \{\#mset\text{-}cls\ L\#\}$ **and**
$union\text{-}clss[simp]$: $mset\text{-}clss\ (union\text{-}clss\ C\ D) = mset\text{-}clss\ C + mset\text{-}clss\ D$ **and**
$mset\text{-}clss\text{-}union\text{-}clss[simp]$: $mset\text{-}clss\ (insert\text{-}clss\ C'\ D) = \{\#mset\text{-}cls\ C'\#\} + mset\text{-}clss\ D$ **and**
$in\text{-}clss\text{-}mset\text{-}clss[dest]$: $in\text{-}clss\ a\ C \Longrightarrow mset\text{-}cls\ a \in\#\ mset\text{-}clss\ C$ **and**
$in\text{-}mset\text{-}clss\text{-}exists\text{-}preimage$: $b \in\#\ mset\text{-}clss\ C \Longrightarrow \exists\, b'.\ in\text{-}clss\ b'\ C \wedge mset\text{-}cls\ b' = b$ **and**
$remove\text{-}from\text{-}clss\text{-}mset\text{-}clss[simp]$:
$mset\text{-}clss\ (remove\text{-}from\text{-}clss\ a\ C) = mset\text{-}clss\ C - \{\#mset\text{-}cls\ a\#\}$ **and**
$in\text{-}clss\text{-}union\text{-}clss[simp]$:
$in\text{-}clss\ a\ (union\text{-}clss\ C\ D) \longleftrightarrow in\text{-}clss\ a\ C \vee in\text{-}clss\ a\ D$
begin

end

experiment
begin
fun *remove-first* **where**
$remove\text{-}first$ - [] = [] |
$remove\text{-}first\ C\ (C'\ \#\ L) = (if\ mset\ C = mset\ C'\ then\ L\ else\ C'\ \#\ remove\text{-}first\ C\ L)$

lemma *mset-map-mset-remove-first*:
$mset\ (map\ mset\ (remove\text{-}first\ a\ C)) = remove1\text{-}mset\ (mset\ a)\ (mset\ (map\ mset\ C))$
by (*induction C*) (*auto simp*: *ac-simps remove1-mset-single-add*)

interpretation *clss-clss*: *raw-clss id*
$id\ op + op \in\#\ \lambda L\ C.\ C + \{\#L\#\}\ remove1\text{-}mset$
by *unfold-locales* (*auto simp*: *ac-simps*)

interpretation *list-clss*: *raw-clss mset*
$\lambda L.\ mset\ (map\ mset\ L)\ op\ @\ \lambda L\ C.\ L \in set\ C\ op\ \#$
remove-first
by *unfold-locales* (*auto simp*: *ac-simps union-mset-list mset-map-mset-remove-first ex-mset*)
end

end

Chapter 1

NOT's CDCL and DPLL

theory *CDCL-WNOT-Measure*
imports *Main List-More*
begin

The organisation of the development is the following:

- `CDCL_WNOT_Measure.thy` contains the measure used to show the termination the core of CDCL.
- `CDCL_NOT.thy` contains the specification of the rules: the rules are defined, and we proof the correctness and termination for some strategies CDCL.
- `DPLL_NOT.thy` contains the DPLL calculus based on the CDCL version.
- `DPLL_W.thy` contains Weidenbach's version of DPLL and the proof of equivalence between the two DPLL versions.

1.1 Measure

This measure show the termination of the core of CDCL: each step improves the number of literals we know for sure.

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**
$\mu_C\ s\ b\ M \equiv (\sum i{=}0..{<}length\ M.\ M!i * b\hat{}\ (s + i - length\ M))$

lemma μ_C*-Nil*[*simp*]:
  $\mu_C\ s\ b\ [] = 0$
  unfolding μ_C*-def* **by** *auto*

lemma μ_C*-single*[*simp*]:
  $\mu_C\ s\ b\ [L] = L * b\ \hat{}\ (s - Suc\ 0)$
  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*:
(∑ *i=1..<Suc j. f i*) = (∑ *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*) = (∑ *i=0..<length* (*L#M*). (*L#M*)!*i* ∗ *b*^ (*s* +*i* − *length* (*L#M*)))
unfolding μ_C*-def* **by** *blast*
also have ... = (∑ *i=0..<1*. (*L#M*)!*i* ∗ *b*^ (*s* +*i* − *length* (*L#M*)))
+ (∑ *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*)
+ (∑ *i=1..<length* (*L#M*). (*L#M*)!*i* ∗ *b*^ (*s* +*i* − *length* (*L#M*)))
by *auto*
moreover {
have (∑ *i=1..<length* (*L#M*). (*L#M*)!*i* ∗ *b*^ (*s* +*i* − *length* (*L#M*))) =
(∑ *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 ... = (∑ *i=0..<length* (*M*). *M*!*i* ∗ *b*^ (*s* + *i* − *length M*))
by *auto*
finally have (∑ *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′*) = (∑ *i=0..<length* (*M@M′*). (*M@M′*)!*i* ∗ *b*^ (*s* +*i* − *length* (*M@M′*)))
unfolding μ_C*-def* **by** *blast*
moreover then have ... = (∑ *i=0..< length M*. (*M@M′*)!*i* ∗ *b*^ (*s* +*i* − *length* (*M@M′*)))
+ (∑ *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 ∀ *i*∈{*0..< length M*}. (*M@M′*)!*i* ∗ *b*^ (*s* +*i* − *length* (*M@M′*)) = *M* ! *i* ∗ *b* ^ (*s* − *length M′*
+ *i* − *length M*)
using ‹*s* ≥ *length* (*M@M′*)› **by** (*auto simp add*: *nth-append ac-simps*)
then have μ_C (*s* − *length M′*) *b M* = (∑ *i=0..< length M*. (*M@M′*)!*i* ∗ *b*^ (*s* +*i* − *length*
(*M@M′*)))
unfolding μ_C*-def* **by** *auto*
ultimately have μ_C *s b* (*M@M′*)= μ_C (*s* − *length M′*) *b M*
+ (∑ *i=length M..<length* (*M@M′*). (*M@M′*)!*i* ∗ *b*^ (*s* +*i* − *length* (*M@M′*)))
by *auto*
moreover {
have (∑ *i=length M..<length* (*M@M′*). (*M@M′*)!*i* ∗ *b*^ (*s* +*i* − *length* (*M@M′*))) =
(∑ *i=0..<length M′*. *M′*!*i* ∗ *b*^ (*s* + *i* − *length M′*))
unfolding *length-append set-sum-atLeastLessThan-add* **by** *auto*
then have (∑ *i=length M..<length* (*M@M′*). (*M@M′*)!*i* ∗ *b*^ (*s* +*i* − *length* (*M@M′*))) = μ_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*)

Copy of `~~/src/HOL/ex/NatSum.thy` (but generalized to $0 \leq k$)

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-diff-assoc2 add.commute assms(4) mult.assoc power-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*: *ac-simps*)
      also have $\ldots = (\sum i{=}0..{<}length\ M.\ b\hat{}\ i) * b\hat{}(s - length\ M) * (b{-}1)$
         by (*simp add*: *setsum-left-distrib setsum-right-distrib ac-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*: *ac-simps*)

      also
        have $(\sum i{=}0..{<}length\ M.\ b\hat{}\ i){*}\ (b{-}1) = b\ \hat{}\ (length\ M) - 1$
          using *sum-of-powers*[*of b length M*] ‹$b{>}1$›
          by (*auto simp add*: *ac-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 ‹$b>1$› **by** *auto*
also have ... $= b \,\hat{}\, s$
by (*metis assms*(*4*) *le-add-diff-inverse power-add*)
finally show *?thesis* **unfolding** μ_C*-def* **by** (*auto simp add*: *ac-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-le: $\forall\, i < length\ M.\ M!i < b$ **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* ‹$b > 0$› **by** *auto*
next
case *M*
show *?thesis* **using** μ_C*-bounded-non-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 $(\sum i=0..<n.\ M\ !\ i * (0::nat) \,\hat{}\, i) \leq M\ !\ 0$
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 $\mu_C\ s\ 0\ M = 0$ **unfolding** μ_C*-def* **by** *auto*}
ultimately show *?thesis* **using** *assms* **unfolding** μ_C*-def* **by** *linarith*
qed

lemma *finite-bounded-pair-list*:
fixes b :: *nat*
shows *finite* $\{(ys, xs).\ length\ xs < s \wedge length\ ys < s \wedge$
$(\forall\, i< length\ xs.\ xs\ !\ i < b) \wedge (\forall\, i< length\ ys.\ ys\ !\ i < b)\}$

proof −
  have H: $\{(ys, xs).\ length\ xs < s \wedge length\ ys < s \wedge$
    $(\forall i< length\ xs.\ xs\ !\ i < b) \wedge (\forall i< length\ ys.\ ys\ !\ i < b)\}$
    $\subseteq$
    $\{xs.\ length\ xs < s \wedge (\forall i< length\ xs.\ xs\ !\ i < b)\} \times$
    $\{xs.\ length\ xs < s \wedge (\forall i< length\ xs.\ xs\ !\ i < b)\}$
    by *auto*
  moreover have *finite* $\{xs.\ length\ xs < s \wedge (\forall i< length\ xs.\ xs\ !\ i < b)\}$
    by (*rule finite-bounded-list*)
  ultimately show *?thesis* **by** (*auto simp*: *finite-subset*)
qed

definition νNOT :: *nat* ⇒ *nat* ⇒ (*nat list* × *nat list*) *set* **where**
$\nu NOT\ s\ base = \{(ys, xs).\ length\ xs < s \wedge length\ ys < s \wedge$
  $(\forall i< length\ xs.\ xs\ !\ i < base) \wedge (\forall i< length\ ys.\ ys\ !\ i < base) \wedge$
  $(ys, xs) \in lenlex\ less\text{-}than\}$

lemma *finite-*νNOT[*simp*]:
  finite (νNOT *s base*)
proof −
  have $\nu NOT\ s\ base \subseteq \{(ys, xs).\ length\ xs < s \wedge length\ ys < s \wedge$
    $(\forall i< length\ xs.\ xs\ !\ i < base) \wedge (\forall i< length\ ys.\ ys\ !\ i < base)\}$
    by (*auto simp*: νNOT*-def*)
  moreover have *finite* $\{(ys, xs).\ length\ xs < s \wedge length\ ys < s \wedge$
    $(\forall i< length\ xs.\ xs\ !\ i < base) \wedge (\forall i< length\ ys.\ ys\ !\ i < base)\}$
      by (*rule finite-bounded-pair-list*)
  ultimately show *?thesis* **by** (*auto simp*: *finite-subset*)
qed

lemma *acyclic-*νNOT: *acyclic* (νNOT *s base*)
  apply (*rule acyclic-subset*[*of lenlex less-than* νNOT *s base*])
    apply (*rule wf-acyclic*)
  by (*auto simp*: νNOT*-def*)

lemma *wf-*νNOT: *wf* (νNOT *s base*)
  by (*rule finite-acyclic-wf*) (*auto simp*: *acyclic-*νNOT)

end
theory *CDCL-NOT*
imports *CDCL-Abstract-Clause-Representation List-More Wellfounded-More CDCL-WNOT-Measure*
  Partial-Annotated-Clausal-Logic
begin

1.2 NOT's CDCL

1.2.1 Auxiliary Lemmas and Measure

We define here some more simplification rules, or rules that have been useful as help for some tactic

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 *atms-of-ms-single-atm-of*[*simp*]:
  atms-of-ms {*unmark L* |*L. P L*} = *atm-of* ‘ {*lit-of L* |*L. P L*}

unfolding *atms-of-ms-def* **by** *force*

lemma *atms-of-uminus-lit-atm-of-lit-of*:
atms-of $\{\#\ -$*lit-of* $x.\ x \in\#\ A\#\}$ = *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 (*unmark-s A*) = *atm-of* ‘ (*lit-of* ‘ *A*)
unfolding *atms-of-ms-def* **by** *auto*

1.2.2 Initial definitions

The state

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

locale *dpll-state-ops* =
fixes
trail :: $'st \Rightarrow ('v,\ unit)$ *ann-lits* **and**
$clauses_{NOT}$ :: $'st \Rightarrow\ 'v$ *clauses* **and**
prepend-trail :: $('v,\ unit)$ *ann-lit* $\Rightarrow\ 'st \Rightarrow\ 'st$ **and**
tl-trail :: $'st \Rightarrow 'st$ **and**
$add\text{-}cls_{NOT}$ :: $'v$ *clause* $\Rightarrow\ 'st \Rightarrow\ 'st$ **and**
$remove\text{-}cls_{NOT}$ :: $'v$ *clause* $\Rightarrow\ 'st \Rightarrow\ 'st$
begin

end

NOT’s state is basically a pair composed of the trail (i.e. the candidate model) and the set of clauses. We abstract this state to convert this state to other states. like Weidenbach’s five-tuple.

locale *dpll-state* =
dpll-state-ops
trail $clauses_{NOT}$ *prepend-trail* *tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ — related to the state
for
trail :: $'st \Rightarrow ('v,\ unit)$ *ann-lits* **and**
$clauses_{NOT}$ :: $'st \Rightarrow\ 'v$ *clauses* **and**
prepend-trail :: $('v,\ unit)$ *ann-lit* $\Rightarrow\ 'st \Rightarrow\ 'st$ **and**
tl-trail :: $'st \Rightarrow 'st$ **and**
$add\text{-}cls_{NOT}$ :: $'v$ *clause* $\Rightarrow\ 'st \Rightarrow\ 'st$ **and**
$remove\text{-}cls_{NOT}$ :: $'v$ *clause* $\Rightarrow\ 'st \Rightarrow\ 'st$ +
assumes
trail-prepend-trail[*simp*]:
$\bigwedge st\ L.$ *trail* (*prepend-trail* $L\ st$) = L # *trail st*
and
tl-trail[*simp*]: *trail* (*tl-trail* S) = *tl* (*trail* S) **and**
$trail\text{-}add\text{-}cls_{NOT}$[*simp*]: $\bigwedge st\ C.$ *trail* ($add\text{-}cls_{NOT}$ $C\ st$) = *trail st* **and**
$trail\text{-}remove\text{-}cls_{NOT}$[*simp*]: $\bigwedge st\ C.$ *trail* ($remove\text{-}cls_{NOT}$ $C\ st$) = *trail st* **and**

clauses-prepend-trail[*simp*]:
$\bigwedge st\ L.$ $clauses_{NOT}$ (*prepend-trail* $L\ st$) = $clauses_{NOT}$ *st*
and
clauses-tl-trail[*simp*]: $\bigwedge st.$ $clauses_{NOT}$ (*tl-trail st*) = $clauses_{NOT}$ *st* **and**
$clauses\text{-}add\text{-}cls_{NOT}$[*simp*]:
$\bigwedge st\ C.$ $clauses_{NOT}$ ($add\text{-}cls_{NOT}$ $C\ st$) = $\{\#C\#\}$ + $clauses_{NOT}$ *st* **and**
$clauses\text{-}remove\text{-}cls_{NOT}$[*simp*]:
$\bigwedge st\ C.$ $clauses_{NOT}$ ($remove\text{-}cls_{NOT}$ $C\ st$) = *removeAll-mset* C ($clauses_{NOT}$ *st*)

begin

We define the following function doing the backtrack in the trail:

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*]

Then we need several lemmas about the *reduce-trail-to*$_{NOT}$.

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

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

lemma *reduce-trail-to*$_{NOT}$*-length*:
length M = *length M'* $\Longrightarrow$ *reduce-trail-to*$_{NOT}$ *M S* = *reduce-trail-to*$_{NOT}$ *M' S*
apply (*induction M S rule*: *reduce-trail-to*$_{NOT}$*.induct*)
by (*simp add*: *reduce-trail-to*$_{NOT}$*.simps*)

abbreviation *trail-weight* **where**
trail-weight S ≡ *map* ((λ*l*. *1* + *length l*) *o snd*) (*get-all-ann-decomposition* (*trail S*))

As we are defining abstract states, the Isabelle equality about them is too strong: we want the weaker equivalence stating that two states are equal if they cannot be distinguished, i.e. given the getter *trail* and *clauses*$_{NOT}$ do not distinguish them.

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

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

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

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

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

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

lemma *reduce-trail-to*$_{NOT}$*-state-eq*$_{NOT}$*-compatible*:
  assumes *ST*: *S* ∼ *T*
  shows *reduce-trail-to*$_{NOT}$ *F S* ∼ *reduce-trail-to*$_{NOT}$ *F T*
proof −
  have *clauses*$_{NOT}$ (*reduce-trail-to*$_{NOT}$ *F S*) = *clauses*$_{NOT}$ (*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

end

Definition of the operation

Each possible is in its own locale.

locale *propagate-ops* =
  dpll-state trail clauses$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$
  for
    trail :: *'st* ⇒ (*'v*, *unit*) *ann-lits* **and**
    clauses$_{NOT}$ :: *'st* ⇒ *'v clauses* **and**
    prepend-trail :: (*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
    tl-trail :: *'st* ⇒*'st* **and**
    add-cls$_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
    remove-cls$_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* +
  fixes
    propagate-cond :: (*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *bool*
begin
inductive *propagate*$_{NOT}$ :: *'st* ⇒ *'st* ⇒ *bool* **where**
propagate$_{NOT}$[*intro*]: *C* + {#*L*#} ∈# *clauses*$_{NOT}$ *S* ⟹ *trail S* ⊨*as CNot C*
    ⟹ *undefined-lit* (*trail S*) *L*
    ⟹ *propagate-cond* (*Propagated L* ()) *S*
    ⟹ *T* ∼ *prepend-trail* (*Propagated L* ()) *S*
    ⟹ *propagate*$_{NOT}$ *S T*
inductive-cases *propagate*$_{NOT}$*E*[*elim*]: *propagate*$_{NOT}$ *S T*

end

locale *decide-ops* =
  dpll-state trail clauses$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$
  for
    trail :: *'st* ⇒ (*'v*, *unit*) *ann-lits* **and**
    clauses$_{NOT}$ :: *'st* ⇒ *'v clauses* **and**
    prepend-trail :: (*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
    tl-trail :: *'st* ⇒*'st* **and**
    add-cls$_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
    remove-cls$_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st*
begin
inductive *decide*$_{NOT}$ :: *'st* ⇒ *'st* ⇒ *bool* **where**
decide$_{NOT}$[*intro*]: *undefined-lit* (*trail S*) *L* ⟹ *atm-of L* ∈ *atms-of-mm* (*clauses*$_{NOT}$ *S*)
  ⟹ *T* ∼ *prepend-trail* (*Decided L*) *S*
  ⟹ *decide*$_{NOT}$ *S T*

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

locale *backjumping-ops* =
  dpll-state trail $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
  for
    trail :: $'st \Rightarrow ('v,\ unit)\ ann\text{-}lits$ **and**
    $clauses_{NOT}$ :: $'st \Rightarrow\ 'v\ clauses$ **and**
    prepend-trail :: $('v,\ unit)\ ann\text{-}lit \Rightarrow\ 'st \Rightarrow\ 'st$ **and**
    tl-trail :: $'st \Rightarrow 'st$ **and**
    $add\text{-}cls_{NOT}$ :: $'v\ clause \Rightarrow\ 'st \Rightarrow\ 'st$ **and**
    $remove\text{-}cls_{NOT}$ :: $'v\ clause \Rightarrow\ 'st \Rightarrow\ 'st$ +
  fixes
    backjump-conds :: $'v\ clause \Rightarrow\ 'v\ clause \Rightarrow\ 'v\ literal \Rightarrow\ 'st \Rightarrow\ 'st \Rightarrow bool$
begin

inductive *backjump* **where**
$trail\ S = F'\ @\ Decided\ K\#\ F$
  $\Longrightarrow T \sim prepend\text{-}trail\ (Propagated\ L\ ())\ (reduce\text{-}trail\text{-}to_{NOT}\ F\ S)$
  $\Longrightarrow C \in\#\ clauses_{NOT}\ S$
  $\Longrightarrow trail\ S \models as\ CNot\ C$
  $\Longrightarrow undefined\text{-}lit\ F\ L$
  $\Longrightarrow atm\text{-}of\ L \in atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \cup atm\text{-}of\ `\ (lits\text{-}of\text{-}l\ (trail\ S))$
  $\Longrightarrow clauses_{NOT}\ S \models pm\ C' + \{\#L\#\}$
  $\Longrightarrow F \models as\ CNot\ C'$
  $\Longrightarrow backjump\text{-}conds\ C\ C'\ L\ S\ T$
  $\Longrightarrow backjump\ S\ T$
inductive-cases *backjumpE*: *backjump S T*

The condition $atm\text{-}of\ L \in atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \cup atm\text{-}of\ `\ lits\text{-}of\text{-}l\ (trail\ S)$ is not implied by the the condition $clauses_{NOT}\ S \models pm\ C' + \{\#L\#\}$ (no negation).

end

1.2.3 DPLL with backjumping

locale *dpll-with-backjumping-ops* =
  propagate-ops trail $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ *propagate-conds* +
  decide-ops trail $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ +
  backjumping-ops trail $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$ *backjump-conds*
  for
    trail :: $'st \Rightarrow ('v,\ unit)\ ann\text{-}lits$ **and**
    $clauses_{NOT}$ :: $'st \Rightarrow\ 'v\ clauses$ **and**
    prepend-trail :: $('v,\ unit)\ ann\text{-}lit \Rightarrow\ 'st \Rightarrow\ 'st$ **and**
    tl-trail :: $'st \Rightarrow 'st$ **and**
    $add\text{-}cls_{NOT}$ :: $'v\ clause \Rightarrow\ 'st \Rightarrow\ 'st$ **and**
    $remove\text{-}cls_{NOT}$ :: $'v\ clause \Rightarrow\ 'st \Rightarrow\ 'st$ **and**
    inv :: $'st \Rightarrow bool$ **and**
    backjump-conds :: $'v\ clause \Rightarrow\ 'v\ clause \Rightarrow\ 'v\ literal \Rightarrow\ 'st \Rightarrow\ 'st \Rightarrow bool$ **and**
    propagate-conds :: $('v,\ unit)\ ann\text{-}lit \Rightarrow\ 'st \Rightarrow bool$ +
  assumes
    bj-can-jump:
    $\bigwedge S\ C\ F'\ K\ F\ L.$
      $inv\ S \Longrightarrow$
      $no\text{-}dup\ (trail\ S) \Longrightarrow$
      $trail\ S = F'\ @\ Decided\ K\ \#\ F \Longrightarrow$
      $C \in\#\ clauses_{NOT}\ S \Longrightarrow$

$trail\ S \models as\ CNot\ C \Longrightarrow$
$undefined\text{-}lit\ F\ L \Longrightarrow$
$atm\text{-}of\ L \in atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \cup atm\text{-}of\ `\ (lits\text{-}of\text{-}l\ (F' @ Decided\ K \# F)) \Longrightarrow$
$clauses_{NOT}\ S \models pm\ C' + \{\#L\#\} \Longrightarrow$
$F \models as\ CNot\ C' \Longrightarrow$
$\neg no\text{-}step\ backjump\ S$
begin

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

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

Definition

We define dpll with backjumping:

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

lemmas $dpll\text{-}bj\text{-}induct = dpll\text{-}bj.induct[split\text{-}format(complete)]$
thm $dpll\text{-}bj\text{-}induct[OF\ dpll\text{-}with\text{-}backjumping\text{-}ops\text{-}axioms]$
lemma $dpll\text{-}bj\text{-}all\text{-}induct[consumes\ 2,\ case\text{-}names\ decide_{NOT}\ propagate_{NOT}\ backjump]$:
fixes $S\ T :: 'st$
assumes
$dpll\text{-}bj\ S\ T$ **and**
$inv\ S$
$\bigwedge L\ T.\ undefined\text{-}lit\ (trail\ S)\ L \Longrightarrow atm\text{-}of\ L \in atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S)$
$\Longrightarrow T \sim prepend\text{-}trail\ (Decided\ L)\ S$
$\Longrightarrow P\ S\ T$ **and**
$\bigwedge C\ L\ T.\ C + \{\#L\#\} \in\# clauses_{NOT}\ S \Longrightarrow trail\ S \models as\ CNot\ C \Longrightarrow undefined\text{-}lit\ (trail\ S)\ L$
$\Longrightarrow T \sim prepend\text{-}trail\ (Propagated\ L\ ())\ S$
$\Longrightarrow P\ S\ T$ **and**
$\bigwedge C\ F'\ K\ F\ L\ C'\ T.\ C \in\# clauses_{NOT}\ S \Longrightarrow F' @ Decided\ K \# F \models as\ CNot\ C$
$\Longrightarrow trail\ S = F' @ Decided\ K \# F$
$\Longrightarrow undefined\text{-}lit\ F\ L$
$\Longrightarrow atm\text{-}of\ L \in atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \cup atm\text{-}of\ `\ (lits\text{-}of\text{-}l\ (F' @ Decided\ K \# F))$
$\Longrightarrow clauses_{NOT}\ S \models pm\ C' + \{\#L\#\}$
$\Longrightarrow F \models as\ CNot\ C'$
$\Longrightarrow T \sim prepend\text{-}trail\ (Propagated\ L\ ())\ (reduce\text{-}trail\text{-}to_{NOT}\ F\ S)$
$\Longrightarrow P\ S\ T$
shows $P\ S\ T$
apply $(induct\ T\ rule:\ dpll\text{-}bj\text{-}induct[OF\ local.dpll\text{-}with\text{-}backjumping\text{-}ops\text{-}axioms])$
apply $(rule\ assms(1))$
using $assms(3)$ **apply** $blast$
apply $(elim\ propagate_{NOT}E)$ **using** $assms(4)$ **apply** $blast$
apply $(elim\ backjumpE)$ **using** $assms(5)\ \langle inv\ S\rangle$ **by** $simp$

Basic properties

First, some better suited induction principle **lemma** $dpll\text{-}bj\text{-}clauses$:
assumes $dpll\text{-}bj\ S\ T$ **and** $inv\ S$
shows $clauses_{NOT}\ S = clauses_{NOT}\ T$
using $assms$ **by** $(induction\ rule:\ dpll\text{-}bj\text{-}all\text{-}induct)\ auto$

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 $I \models sm\ clauses_{NOT}\ S \longleftrightarrow I \models sm\ clauses_{NOT}\ T$
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-mm* ($clauses_{NOT}$ S) = *atms-of-mm* ($clauses_{NOT}$ 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-l* (*trail S*)) ⊆ *atms-of-mm* ($clauses_{NOT}$ S)
shows *atm-of* ‘ (*lits-of-l* (*trail T*)) ⊆ *atms-of-mm* ($clauses_{NOT}$ 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-mm ($clauses_{NOT}$ S) ⊆ A **and**
atm-of ‘ (*lits-of-l* (*trail S*)) ⊆ A
shows *atm-of* ‘ (*lits-of-l* (*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* ($clauses_{NOT}$ S) (*get-all-ann-decomposition* (*trail S*))
shows *all-decomposition-implies-m* ($clauses_{NOT}$ T) (*get-all-ann-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* = $clauses_{NOT}$ S
obtain *a y l* **where** *ay*: *get-all-ann-decomposition ?M′* = (*a*, *y*) # *l*
by (*cases get-all-ann-decomposition ?M′*) *fastforce+*
then have *M′*: *?M′* = *y* @ *a* **using** *get-all-ann-decomposition-decomp*[*of ?M′*] **by** *auto*
have *M*: *get-all-ann-decomposition* (*trail S*) = (*a*, *tl y*) # *l*

using *ay undef* **by** (*cases get-all-ann-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-ann-decomposition-decomp* **by** *force*
have *a-Un-N-M*: *unmark-l a* ∪ *set-mset ?N* ⊨*ps unmark-l* (*tl y*)
using *decomp ay* **unfolding** *all-decomposition-implies-def* **by** (*simp add*: *M*)+

moreover have *unmark-l 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 *unmark-l ?M'* ⊨*ps CNot C*
using ‹*trail S* ⊨*as CNot C*› *undef* **by** (*auto simp add*: *true-annots-true-clss-clss*)
have *a1*: *unmark-l a* ∪ *unmark-l* (*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*)
then have *unmark-l a* ∪ *set-mset* ($clauses_{NOT}$ *S*) ⊨*ps unmark-l a* ∪ *unmark-l* (*tl y*)
using *a-Un-N-M true-clss-clss-def* **by** *blast*
then show *unmark-l a* ∪ *set-mset* ($clauses_{NOT}$ *S*) ⊨*ps CNot C*
using *a1* **by** (*meson true-clss-clss-left-right true-clss-clss-union-and*
true-clss-clss-union-l-r)
qed
ultimately have *unmark-l a* ∪ *set-mset ?N* ⊨*ps unmark-l ?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* ($clauses_{NOT}$ *S*) (*get-all-ann-decomposition F*)
using *decomp* **unfolding** *tr all-decomposition-implies-def*
by (*metis* (*no-types*, *lifting*) *get-all-ann-decomposition.simps*(*1*)
get-all-ann-decomposition-never-empty hd-Cons-tl insert-iff list.sel(*3*) *list.set*(*2*)
tl-get-all-ann-decomposition-skip-some)

obtain *a b li* **where** *F*: *get-all-ann-decomposition F* = (*a*, *b*) # *li*
by (*cases get-all-ann-decomposition F*) *auto*
have *F* = *b* @ *a*
using *get-all-ann-decomposition-decomp*[*of F a b*] *F* **by** *auto*
have *a-N-b*:*unmark-l a* ∪ *set-mset* ($clauses_{NOT}$ *S*) ⊨*ps unmark-l b*
using *decomp* **unfolding** *all-decomposition-implies-def* **by** (*auto simp add*: *F*)

have *F-D*: *unmark-l F* ⊨*ps CNot D*
using ‹*F* ⊨*as CNot D*› **by** (*simp add*: *true-annots-true-clss-clss*)
then have *unmark-l a* ∪ *unmark-l b* ⊨*ps CNot D*
unfolding ‹*F* = *b* @ *a*› **by** (*simp add*: *image-Un sup.commute*)
have *a-N-CNot-D*: *unmark-l a* ∪ *set-mset* ($clauses_{NOT}$ *S*) ⊨*ps CNot D* ∪ *unmark-l 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-l a* ∪ *set-mset* ($clauses_{NOT}$ *S*) ⊨*p D*+{#*L*#}
by (*simp add*: *N-C*)

have *unmark-l a* ∪ *set-mset* (*clauses*$_{NOT}$ *S*) ⊨*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

Termination

Using a proper measure **lemma** *length-get-all-ann-decomposition-append-Decided*:
length (*get-all-ann-decomposition* (*F'* @ *Decided K* # *F*)) =
  length (*get-all-ann-decomposition F'*)
  + *length* (*get-all-ann-decomposition* (*Decided K* # *F*))
  − *1*
by (*induction F' rule*: *ann-lit-list-induct*) *auto*

lemma *take-length-get-all-ann-decomposition-decided-sandwich*:
take (*length* (*get-all-ann-decomposition F*))
  (*map* (*f o snd*) (*rev* (*get-all-ann-decomposition* (*F'* @ *Decided K* # *F*))))
  =
  map (*f o snd*) (*rev* (*get-all-ann-decomposition F*))

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

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

lemma *length-in-get-all-ann-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-ann-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-ann-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-ann-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) ann-lits* **and** *N* :: *'v clauses*
  assumes
    dpll-bj S T **and**
    inv S **and**
    NA: *atms-of-mm* ($clauses_{NOT}$ *S*) ⊆ *atms-of-ms A* **and**
    MA: *atm-of ‘ lits-of-l (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-l (Propagated L () # trail S)* ⊆ *atms-of-ms A*
    using $propagate_{NOT}$ *dpll-bj-atms-in-trail-in-set* $bj\text{-}propagate_{NOT}$ *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-ann-decomposition (trail S) = (a, b) # l*
    by (*cases get-all-ann-decomposition (trail S)*) *auto*
  have *b-le-M*: *length b* ≤ *length (trail S)*
    using *get-all-ann-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-l[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-l (Decided L # (trail S))* ⊆ *atms-of-ms A*
    using *dpll-bj-atms-in-trail-in-set* $bj\text{-}decide_{NOT}$ $decide_{NOT}.decide_{NOT}$*[OF* $decide_{NOT}.hyps$*]* *NA MA MC*
    by *auto*

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

by (*cases get-all-ann-decomposition* (*trail S*)) *auto*

then have *length* (*Decided L* # (*trail S*)) $\leq$ *card* (*atms-of-ms A*)
using *incl finite* **unfolding** *no-dup-length-eq-card-atm-of-lits-of-l*[*OF no-dup*]
by (*simp add*: *card-mono*)
show *?case* **using** *T undef-L* **by** (*simp add*: 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-l* (*Propagated L* () # *F*) $\subseteq$ *atms-of-ms A*
using *dpll-bj-atms-in-trail-in-set NA MA L* **by** (*auto simp*: *tr-S*)

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-ann-decomposition* (*trail S*) = (*a*, *b*) # *l*
by (*cases get-all-ann-decomposition* (*trail S*)) *auto*
have *b-le-M*: *length b* $\leq$ *length* (*trail S*)
using *get-all-ann-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*) $\leq$ *card* (*atms-of-ms A*)
using *incl finite* **unfolding** *no-dup-length-eq-card-atm-of-lits-of-l*[*OF no-dup*]
by (*simp add*: *card-mono*)
have *tr-S-le-A*: *length* (*trail S*) $\leq$ *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-l*)
obtain *a b l* **where** *F*: *get-all-ann-decomposition F* = (*a*, *b*) # *l*
by (*cases get-all-ann-decomposition F*) *auto*
then have *F* = *b* @ *a*
using *get-all-ann-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* (λ*a*. *Suc* (*length* (*snd a*))) (*rev* (*get-all-ann-decomposition* (*F′* @ *Decided K* # *F*)))
= *map* (λ*a*. *Suc* (*length* (*snd a*))) (*rev* (*get-all-ann-decomposition F*)) @ *rem*
using *take-length-get-all-ann-decomposition-decided-sandwich*[*of F* λ*a*. *Suc* (*length a*) *F′ K*]
unfolding *o-def* **by** (*metis append-take-drop-id*)
then have *rem*: *map* (λ*a*. *Suc* (*length* (*snd a*)))
(*get-all-ann-decomposition* (*F′* @ *Decided K* # *F*))
= *rev rem* @ *map* (λ*a*. *Suc* (*length* (*snd a*))) ((*get-all-ann-decomposition F*))
by (*simp add*: *rev-map*[*symmetric*] *rev-swap*)
have *length* (*rev rem* @ *map* (λ*a*. *Suc* (*length* (*snd a*))) (*get-all-ann-decomposition F*))
$\leq$ *Suc* (*card* (*atms-of-ms A*))
using *arg-cong*[*OF rem*, *of length*] *tr-S-le-A*
length-get-all-ann-decomposition-length[*of F′* @ *Decided K* # *F*] *tr-S* **by** *auto*
moreover
{ 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* $\Longrightarrow$ *rev 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 $\forall$ *i*<*length rem*. *rev rem* ! *i* < *card* (*atms-of-ms A*) + *2*
using *tr-S-le-A length-in-get-all-ann-decomposition-bounded*[*of - S*] **unfolding** *tr-S*
by (*force simp add*: *o-def rem dest*!: *H intro*: *length-get-all-ann-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-mm* (*clauses*$_{NOT}$ *S*) $\subseteq$ *atms-of-ms A* **and**
M-A: *atm-of* ' *lits-of-l* (*trail S*) $\subseteq$ *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*))
− 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*)
proof −
let *?b* = *2*+*card* (*atms-of-ms A*)
let *?s* = *1*+*card* (*atms-of-ms A*)
let *?μ* = μ_C *?s ?b*
have *M'-A*: *atm-of* ' *lits-of-l* (*trail T*) $\subseteq$ *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* $\Longrightarrow$ *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 S*) $\leq$ *card* (*atms-of-ms A*)
by (*simp add*: *fin-A M-A card-mono no-dup-length-eq-card-atm-of-lits-of-l nd*)
have *l-M'-A*: *length* (*trail T*) $\leq$ *card* (*atms-of-ms A*)
by (*simp add*: *fin-A M'-A card-mono no-dup-length-eq-card-atm-of-lits-of-l nd'*)
have *l-trail-weight-M*: *length* (*trail-weight T*) $\leq$ *1*+*card* (*atms-of-ms A*)
using *l-M'-A length-get-all-ann-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-ann-decomposition-bounded*[*of - T*] *l-M'-A*
by (*metis* (*no-types*, *lifting*) *H Nat.le-trans add-2-eq-Suc' not-le not-less-eq-eq*)

from *dpll-bj-trail-mes-increasing-prop*[*OF dpll inv N-A M-A nd fin-A*]
have μ_C *?s ?b* (*trail-weight S*) < μ_C *?s ?b* (*trail-weight T*) **by** *simp*
moreover from μ_C*-bounded*[*OF bounded-M l-trail-weight-M*]
have μ_C *?s ?b* (*trail-weight T*) $\leq$ *?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-mm* (*clauses*$_{NOT}$ *S*) $\subseteq$ *atms-of-ms A* ∧ *atm-of* ' *lits-of-l* (*trail S*) $\subseteq$ *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*))
λ*S*. μ_C (*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) ∈ ?A

  have fin-A: finite (atms-of-ms A)
    using fin by auto
  have
    dpll-bj: dpll-bj a b and
    N-A: atms-of-mm (clausesNOT a) ⊆ atms-of-ms A and
    M-A: atm-of ‘ lits-of-l (trail a) ⊆ 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-l (trail b) ⊆ 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 ⟹ 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 a) ≤ card (atms-of-ms A)
    by (simp add: fin-A M-A card-mono no-dup-length-eq-card-atm-of-lits-of-l nd)
  have l-M'-A: length (trail b) ≤ card (atms-of-ms A)
    by (simp add: fin-A M'-A card-mono no-dup-length-eq-card-atm-of-lits-of-l nd')
  have l-trail-weight-M: length (trail-weight b) ≤ 1+card (atms-of-ms A)
     using l-M'-A length-get-all-ann-decomposition-length[of trail b] by auto
  have bounded-M: ∀ i<length (trail-weight b). (trail-weight b)! i < card (atms-of-ms A) + 2
    using length-in-get-all-ann-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 μC ?s ?b (trail-weight a) < μC ?s ?b (trail-weight b) by simp
  moreover from μC-bounded[OF bounded-M l-trail-weight-M]
    have μC ?s ?b (trail-weight b) ≤ ?b ^ ?s by auto
  ultimately show ?b ^ ?s ≤ ?b ^ ?s ∧
           μC ?s ?b (trail-weight b) ≤ ?b ^ ?s ∧
           μC ?s ?b (trail-weight a) < μC ?s ?b (trail-weight b)
    by blast
qed
```

Normal Forms

We prove that given a normal form of DPLL, with some structural invariants, then 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* rule tells us that every variable in N has a value.
2. The assumption $\neg$ M $\models$*as* N implies that there is conflict.
3. There is at least one decision in the trail (otherwise, M would be a model of the set of clauses 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 clause set* **and** *S T* :: *'st*
  assumes
    atms-of-mm (*clauses*$_{NOT}$ *S*) $\subseteq$ *atms-of-ms A* **and**
    atm-of ` *lits-of-l* (*trail S*) $\subseteq$ *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*$_{NOT}$ *S*) (*get-all-ann-decomposition* (*trail S*))
  shows *unsatisfiable* (*set-mset* (*clauses*$_{NOT}$ *S*))
    $\vee$ (*trail S* $\models$*asm clauses*$_{NOT}$ *S* $\wedge$ *satisfiable* (*set-mset* (*clauses*$_{NOT}$ *S*)))
proof −
  let *?N* = *set-mset* (*clauses*$_{NOT}$ *S*)
  let *?M* = *trail S*
  consider
      (*sat*) *satisfiable ?N* **and** *?M* $\models$*as ?N*
    | (*sat'*) *satisfiable ?N* **and** $\neg$ *?M* $\models$*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* $\in$ *?N* **and** $\neg$*?M* $\models$*a C* **using** *M* **unfolding** *true-annots-def* **by** *auto*
      obtain *I* :: *'v literal set* **where**
        I $\models$*s ?N* **and**
        cons: *consistent-interp I* **and**
        tot: *total-over-m I ?N* **and**
        atm-I-N: *atm-of* `*I* $\subseteq$ *atms-of-ms ?N*
        using *sat* **unfolding** *satisfiable-def-min* **by** *auto*
      let *?I* = *I* $\cup$ {*P*| *P*. *P* $\in$ *lits-of-l ?M* $\wedge$ *atm-of P* $\notin$ *atm-of* ` *I*}
      let *?O* = {*unmark L* |*L*. *is-decided L* $\wedge$ *L* $\in$ *set ?M* $\wedge$ *atm-of* (*lit-of L*) $\notin$ *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* $\cup$ *unmark-l ?M*)
        using *tot atm-I-N* **unfolding** *total-over-m-def total-over-set-def*
        by (*fastforce simp*: *image-iff lits-of-def*)
      have {*P* |*P*. *P* $\in$ *lits-of-l ?M* $\wedge$ *atm-of P* $\notin$ *atm-of* ` *I*} $\models$*s ?O*
        using ⟨*I*$\models$*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* $\models$*s ?N* $\cup$ *?O*
        using ⟨*I*$\models$*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*: *lits-of-def elim*!: *is-decided-ex-Decided*)

have *atms-N-M*: *atms-of-ms ?N* ⊆ *atm-of ‘ lits-of-l ?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-l ?M*
      by *auto*
    have *undefined-lit ?M* (*Pos l*)
      using *l-M* **by** (*metis Decided-Propagated-in-iff-in-lits-of-l*
        atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set literal.sel(*1*))
    from *bj-decide*$_{NOT}$[*OF decide*$_{NOT}$[*OF this*]] **show** *False*
      using *l-N n-s* **by** (*metis literal.sel*(*1*) *state-eq*$_{NOT}$*-ref*)
  qed
have *?M* ⊨*as CNot C*
  apply (*rule all-variables-defined-not-imply-cnot*)
  using ‹*C* ∈ *set-mset* (*clauses*$_{NOT}$ *S*)› ‹¬ *trail S* ⊨*a C*›
    atms-N-M **by** (*auto dest*: *atms-of-atms-of-ms-mono*)
have ∃ *l* ∈ *set ?M*. *is-decided l*
  proof (*rule ccontr*)
    let *?O* = {*unmark L* |*L*. *is-decided L* ∧ *L* ∈ *set ?M* ∧ *atm-of* (*lit-of L*) ∉ *atms-of-ms ?N*}
    have *ϑ*[*iff*]: ⋀*I*. *total-over-m I* (*?N* ∪ *?O* ∪ *unmark-l ?M*)
      ⟷ *total-over-m I* (*?N* ∪*unmark-l ?M*)
      unfolding *total-over-set-def total-over-m-def atms-of-ms-def* **by** *blast*
    assume ¬ *?thesis*
    then have [*simp*]:{*unmark L* |*L*. *is-decided L* ∧ *L* ∈ *set ?M*}
      = {*unmark L* |*L*. *is-decided L* ∧ *L* ∈ *set ?M* ∧ *atm-of* (*lit-of L*) ∉ *atms-of-ms ?N*}
      by *auto*
    then have *?N* ∪ *?O* ⊨*ps unmark-l ?M*
      using *all-decomposition-implies-propagated-lits-are-implied*[*OF decomp*] **by** *auto*

    then have *?I* ⊨*s unmark-l ?M*
      using *cons-I′ I′-N tot-I′* ‹*?I* ⊨*s ?N* ∪ *?O*› **unfolding** *ϑ true-clss-clss-def* **by** *blast*
    then have *lits-of-l ?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*) *ann-lits* **where**
  M-K: *?M* = *F′* @ *Decided K* # *F* **and**
  nm: ∀ *f*∈*set F′*. ¬*is-decided f*
  unfolding *is-decided-def* **by** (*metis* (*full-types*) *old.unit.exhaust*)
let *?K* = *Decided K* :: (*′v*, *unit*) *ann-lit*
have *?K* ∈ *set ?M*
  unfolding *M-K* **by** *auto*
let *?C* = *image-mset lit-of* {#*L*∈#*mset ?M*. *is-decided L* ∧ *L*≠*?K*#} :: *′v clause*
let *?C′* = *set-mset* (*image-mset* (λ*L*::*′v literal*. {#*L*#}) (*?C* + *unmark ?K*))
have *?N* ∪ {*unmark L* |*L*. *is-decided L* ∧ *L* ∈ *set ?M*} ⊨*ps unmark-l ?M*
  using *all-decomposition-implies-propagated-lits-are-implied*[*OF decomp*] **.**

moreover have C': $?C' = \{unmark\ L\ |L.\ is\text{-}decided\ L \wedge L \in set\ ?M\}$
  unfolding $M\text{-}K$ **by** *standard force+*
ultimately have $N\text{-}C\text{-}M$: $?N \cup ?C' \models ps\ unmark\text{-}l\ ?M$
  by *auto*
have $N\text{-}M\text{-}False$: $?N \cup (\lambda L.\ unmark\ L)\ `\ (set\ ?M) \models ps\ \{\{\#\}\}$
  using M ‹$?M \models as\ CNot\ C$› ‹$C \in ?N$› **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** $M\text{-}K$ **by** (*simp add*: *defined-lit-map*)
moreover
  have $?N \cup ?C' \models ps\ \{\{\#\}\}$
    proof −
      have A: $?N \cup ?C' \cup unmark\text{-}l\ ?M = ?N \cup unmark\text{-}l\ ?M$
        unfolding $M\text{-}K$ **by** *auto*
      show *?thesis*
        using *true-clss-clss-left-right*[*OF N-C-M*, *of* $\{\{\#\}\}$] *N-M-False* **unfolding** A **by** *auto*
    qed
  have $?N \models p\ image\text{-}mset\ uminus\ ?C + \{\#{-K}\#\}$
    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* $(?N \cup \{image\text{-}mset\ uminus\ ?C + \{\#-\ K\#\}\})$) **and**
        cons: *consistent-interp* I **and**
        $I \models s\ ?N$
      have $(K \in I \wedge -K \notin I) \vee (-K \in I \wedge K \notin I)$
        using *cons tot* **unfolding** *consistent-interp-def* **by** (*cases* K) *auto*
      have $\{a \in set\ (trail\ S).\ is\text{-}decided\ a \wedge a \neq Decided\ K\} =$
        $set\ (trail\ S) \cap \{L.\ is\text{-}decided\ L \wedge L \neq Decided\ K\}$
        by *auto*
      then have tot': *total-over-set* I
          $(atm\text{-}of\ `\ lit\text{-}of\ `\ (set\ ?M \cap \{L.\ is\text{-}decided\ L \wedge L \neq Decided\ K\}))$
        using *tot* **by** (*auto simp add*: *atms-of-uminus-lit-atm-of-lit-of*)
      { fix $x :: ('v,\ unit)\ ann\text{-}lit$
        assume
          a3: $lit\text{-}of\ x \notin I$ **and**
          a1: $x \in set\ ?M$ **and**
          a4: *is-decided* x **and**
          a5: $x \neq Decided\ K$
        then have $Pos\ (atm\text{-}of\ (lit\text{-}of\ x)) \in I \vee Neg\ (atm\text{-}of\ (lit\text{-}of\ x)) \in I$
          using *a5 a4* tot' *a1* **unfolding** *total-over-set-def atms-of-s-def* **by** *blast*
        moreover have *f6*: $Neg\ (atm\text{-}of\ (lit\text{-}of\ x)) = -\ Pos\ (atm\text{-}of\ (lit\text{-}of\ x))$
          by *simp*
        ultimately have $-\ lit\text{-}of\ x \in I$
          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 $\neg I \models s\ ?C'$
        using ‹$?N \cup ?C' \models ps\ \{\{\#\}\}$› *tot cons* ‹$I \models s\ ?N$›
        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 $I \models image\text{-}mset\ uminus\ ?C + \{\#-\ K\#\}$
        unfolding *true-clss-def true-cls-def* **using** ‹$(K \in I \wedge -K \notin I) \vee (-K \in I \wedge K \notin I)$›
        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-can-jump*[*of S F′ K F C −K*
image-mset uminus (*image-mset lit-of* {# *L* :# *mset ?M*. *is-decided L* ∧ *L* ≠ *Decided K*#})]
‹*C*∈*?N*› *n-s* ‹*?M* ⊨*as CNot C*› *bj-backjump inv* ‹*no-dup* (*trail S*)› **unfolding** *M-K* **by** *auto*
then show *?thesis* **by** *fast*
qed *auto*
qed

end — End of *dpll-with-backjumping-ops*

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

lemma *rtranclp-dpll-bj-inv*:
assumes $dpll\text{-}bj^{**}$ *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 $dpll\text{-}bj^{**}$ *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
$dpll\text{-}bj^{**}$ *S T* **and** *inv S*
shows *atms-of-mm* ($clauses_{NOT}$ *S*) = *atms-of-mm* ($clauses_{NOT}$ *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
$dpll\text{-}bj^{**}$ *S T* **and**
inv S **and**
atm-of ‘ (*lits-of-l* (*trail S*)) ⊆ *atms-of-mm* ($clauses_{NOT}$ *S*)
shows *atm-of* ‘ (*lits-of-l* (*trail T*)) ⊆ *atms-of-mm* ($clauses_{NOT}$ *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 $dpll\text{-}bj^{**}\ S\ T$ **and** *inv S*
shows $I \models sm\ clauses_{NOT}\ S \longleftrightarrow I \models sm\ clauses_{NOT}\ T$
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
$dpll\text{-}bj^{**}\ S\ T$ **and**
inv S
$atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \subseteq A$ **and**
atm-of ' (*lits-of-l* (*trail S*)) $\subseteq A$
shows *atm-of* ' (*lits-of-l* (*trail T*)) $\subseteq A$
using *assms* **by** (*induction rule*: *rtranclp-induct*)
(*auto dest*: *rtranclp-dpll-bj-inv*
simp: *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
$dpll\text{-}bj^{**}\ S\ T$ **and**
inv S
all-decomposition-implies-m ($clauses_{NOT}$ *S*) (*get-all-ann-decomposition* (*trail S*))
shows *all-decomposition-implies-m* ($clauses_{NOT}$ *T*) (*get-all-ann-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*:
$\{(T, S).\ dpll\text{-}bj^{++}\ S\ T$
$\wedge$ $atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \subseteq atms\text{-}of\text{-}ms\ A \wedge atm\text{-}of$ ' $lits\text{-}of\text{-}l\ (trail\ S) \subseteq atms\text{-}of\text{-}ms\ A$
$\wedge\ no\text{-}dup\ (trail\ S) \wedge inv\ S\}$
$\subseteq \{(T, S).\ dpll\text{-}bj\ S\ T \wedge atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \subseteq atms\text{-}of\text{-}ms\ A$
$\wedge\ atm\text{-}of$ ' $lits\text{-}of\text{-}l\ (trail\ S) \subseteq atms\text{-}of\text{-}ms\ A \wedge no\text{-}dup\ (trail\ S) \wedge inv\ S\}^+$
(**is** $?A \subseteq ?B^+$)
proof *standard*
fix *x*
assume *x-A*: $x \in ?A$
obtain *S T*::'*st* **where**
x[*simp*]: $x = (T, S)$ **by** (*cases x*) *auto*
have
$dpll\text{-}bj^{++}\ S\ T$ **and**
$atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \subseteq atms\text{-}of\text{-}ms\ A$ **and**
atm-of ' *lits-of-l* (*trail S*) $\subseteq$ *atms-of-ms A* **and**
no-dup (*trail S*) **and**
inv S
using *x-A* **by** *auto*
then show $x \in ?B^+$ **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-mm* (*clauses*$_{NOT}$ *S*) = *atms-of-mm* (*clauses*$_{NOT}$ *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-l* (*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-mm* (*clauses*$_{NOT}$ *S*) ⊆ *atms-of-ms A* ∧ *atm-of* ' *lits-of-l* (*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*$_{NOT}$ *S* ⟷ *I*⊨*sextm clauses*$_{NOT}$ *T*
by (*simp add*: *dpll-bj-clauses*)

lemma *rtranclp-dpll-bj-sat-ext-iff*:
dpll-bj** *S T* ⟹ *inv S* ⟹ *I*⊨*sextm clauses*$_{NOT}$ *S* ⟷ *I*⊨*sextm clauses*$_{NOT}$ *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 clause set* **and** *S T* :: *'st*
assumes
full: *full dpll-bj S T* **and**
atms-S: *atms-of-mm* (*clauses*$_{NOT}$ *S*) ⊆ *atms-of-ms A* **and**
atms-trail: *atm-of* ' *lits-of-l* (*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*$_{NOT}$ *S*) (*get-all-ann-decomposition* (*trail S*))
shows *unsatisfiable* (*set-mset* (*clauses*$_{NOT}$ *S*))
∨ (*trail T* ⊨*asm clauses*$_{NOT}$ *S* ∧ *satisfiable* (*set-mset* (*clauses*$_{NOT}$ *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-mm* (*clauses*$_{NOT}$ *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-l* (*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*$_{NOT}$ *T*) (*get-all-ann-decomposition* (*trail T*))
    using ⟨*inv S*⟩ *decomp rtranclp-dpll-bj-all-decomposition-implies-inv st* **by** *blast*

ultimately have *unsatisfiable* (*set-mset* (*clauses*$_{NOT}$ *T*))
∨ (*trail T* ⊨*asm clauses*$_{NOT}$ *T* ∧ *satisfiable* (*set-mset* (*clauses*$_{NOT}$ *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 clause set* **and** *S T* :: *'st*
assumes
full: *full dpll-bj S T* **and**
trail S = [] **and**
clauses$_{NOT}$ *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-bj*$^{++}$ *S T* **and** *inv*: *inv S* **and**
N-A: *atms-of-mm* (*clauses*$_{NOT}$ *S*) ⊆ *atms-of-ms A* **and**
M-A: *atm-of* ‘ *lits-of-l* (*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-mm* (*clauses*$_{NOT}$ *S*) = *atms-of-mm* (*clauses*$_{NOT}$ *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-mm* (*clauses*$_{NOT}$ *T*) ⊆ *atms-of-ms A*
using *N-A* **by** *auto*
moreover have *M-A'*: *atm-of* ‘ *lits-of-l* (*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 — End of *dpll-with-backjumping*

1.2.4 CDCL

In this section we will now define the conflict driven clause learning above DPLL: we first introduce the rules learn and forget, and the add these rules to the DPLL calculus.

Learn and Forget

Learning adds a new clause where all the literals are already included in the clauses.

locale *learn-ops* =
  dpll-state trail $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
  for
    trail :: $'st \Rightarrow ('v, unit)$ *ann-lits* **and**
    $clauses_{NOT}$ :: $'st \Rightarrow 'v$ *clauses* **and**
    prepend-trail :: $('v, unit)$ *ann-lit* $\Rightarrow 'st \Rightarrow 'st$ **and**
    tl-trail :: $'st \Rightarrow 'st$ **and**
    $add\text{-}cls_{NOT}$ :: $'v$ *clause* $\Rightarrow 'st \Rightarrow 'st$ **and**
    $remove\text{-}cls_{NOT}$ :: $'v$ *clause* $\Rightarrow 'st \Rightarrow 'st$ +
  fixes
    learn-cond :: $'v$ *clause* $\Rightarrow 'st \Rightarrow$ *bool*
begin
inductive *learn* :: $'st \Rightarrow 'st \Rightarrow$ *bool* **where**
$learn_{NOT}$*-rule*: $clauses_{NOT}$ $S \models pm\ C \Longrightarrow$
  atms-of $C \subseteq$ *atms-of-mm* $(clauses_{NOT}\ S) \cup$ *atm-of* ‘ (*lits-of-l* (*trail* S)) $\Longrightarrow$
  learn-cond $C\ S \Longrightarrow$
  $T \sim add\text{-}cls_{NOT}\ C\ S \Longrightarrow$
  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 $\mu_C\ A\ B$ (*trail-weight* S) = $\mu_C\ A\ B$ (*trail-weight* T)
  using *assms* **by** (*auto elim*: $learn_{NOT}E$)
end

Forget removes an information that can be deduced from the context (e.g. redundant clauses, tautologies)

locale *forget-ops* =
  dpll-state trail $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
  for
    trail :: $'st \Rightarrow ('v, unit)$ *ann-lits* **and**
    $clauses_{NOT}$ :: $'st \Rightarrow 'v$ *clauses* **and**
    prepend-trail :: $('v, unit)$ *ann-lit* $\Rightarrow 'st \Rightarrow 'st$ **and**
    tl-trail :: $'st \Rightarrow 'st$ **and**
    $add\text{-}cls_{NOT}$ :: $'v$ *clause* $\Rightarrow 'st \Rightarrow 'st$ **and**
    $remove\text{-}cls_{NOT}$ :: $'v$ *clause* $\Rightarrow 'st \Rightarrow 'st$ +
  fixes
    forget-cond :: $'v$ *clause* $\Rightarrow 'st \Rightarrow$ *bool*
begin
inductive $forget_{NOT}$ :: $'st \Rightarrow 'st \Rightarrow$ *bool* **where**
$forget_{NOT}$:
  removeAll-mset $C(clauses_{NOT}\ S) \models pm\ C \Longrightarrow$
  forget-cond $C\ S \Longrightarrow$
  $C \in\#\ clauses_{NOT}\ S \Longrightarrow$
  $T \sim remove\text{-}cls_{NOT}\ C\ S \Longrightarrow$

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$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$ *learn-cond* +
forget-ops trail clauses$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$ *forget-cond*
for
trail :: *'st* ⇒ (*'v*, *unit*) *ann-lits* **and**
clauses$_{NOT}$ :: *'st* ⇒ *'v clauses* **and**
prepend-trail :: (*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒*'st* **and**
add-cls$_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
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

Definition of CDCL

locale *conflict-driven-clause-learning-ops* =
dpll-with-backjumping-ops trail clauses$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$
inv backjump-conds propagate-conds +
learn-and-forget$_{NOT}$ *trail clauses*$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$ *learn-cond*
forget-cond
for
trail :: *'st* ⇒ (*'v*, *unit*) *ann-lits* **and**
clauses$_{NOT}$ :: *'st* ⇒ *'v clauses* **and**
prepend-trail :: (*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒*'st* **and**
add-cls$_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
remove-cls$_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
inv :: *'st* ⇒ *bool* **and**
backjump-conds :: *'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool* **and**
propagate-conds :: (*'v*, *unit*) *ann-lit* ⇒ *'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:
$\bigwedge C\ T.\ clauses_{NOT}\ S \models pm\ C \Longrightarrow$
$atms\text{-}of\ C \subseteq atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \cup atm\text{-}of\ `\ (lits\text{-}of\text{-}l\ (trail\ S)) \Longrightarrow$
$T \sim add\text{-}cls_{NOT}\ C\ S \Longrightarrow$
$P\ S\ T$ **and**
forgetting: $\bigwedge C\ T.\ removeAll\text{-}mset\ C\ (clauses_{NOT}\ S) \models pm\ C \Longrightarrow$
$C \in\#\ clauses_{NOT}\ S \Longrightarrow$
$T \sim remove\text{-}cls_{NOT}\ C\ S \Longrightarrow$
$P\ S\ T$
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-l* (*trail T*))
using $cdcl_{NOT}$*-no-dup*[*OF assms*] *distinct-consistent-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 means that some variable of the trail might not be present 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 $atms\text{-}of\text{-}mm\ (clauses_{NOT}\ T) \subseteq atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \cup atm\text{-}of\ `\ (lits\text{-}of\text{-}l\ (trail\ S))$
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 $atm\text{-}of\ `\ (lits\text{-}of\text{-}l\ (trail\ S)) \subseteq atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S)$
shows $atm\text{-}of\ `\ (lits\text{-}of\text{-}l\ (trail\ T)) \subseteq atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S)$
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**
$atms\text{-}of\text{-}mm\ (clauses_{NOT}\ S) \subseteq A$ **and**
$atm\text{-}of\ `\ (lits\text{-}of\text{-}l\ (trail\ S)) \subseteq A$
shows $atm\text{-}of\ `\ (lits\text{-}of\text{-}l\ (trail\ T)) \subseteq A$
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 ($clauses_{NOT}$ *S*) (*get-all-ann-decomposition* (*trail S*))
shows
all-decomposition-implies-m ($clauses_{NOT}$ *T*) (*get-all-ann-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-ann-decomposition* (*trail T*))
then have *unmark-l a* ∪ *set-mset* ($clauses_{NOT}$ *S*) ⊨*ps unmark-l b*
using *decomp T* **by** (*auto simp add*: *all-decomposition-implies-def*)
moreover
have *a1*:*C* ∈ *set-mset* ($clauses_{NOT}$ *S*)
using *C* **by** *blast*
have $clauses_{NOT}$ *T* = $clauses_{NOT}$ ($remove\text{-}cls_{NOT}$ *C S*)
using *T* $state\text{-}eq_{NOT}$*-clauses* **by** *blast*
then have *set-mset* ($clauses_{NOT}$ *T*) ⊨*ps set-mset* ($clauses_{NOT}$ *S*)
using *a1* **by** (*metis* (*no-types*) $clauses\text{-}remove\text{-}cls_{NOT}$ *cls-C insert-Diff order-refl*
set-mset-minus-replicate-mset(*1*) *true-clss-clss-def true-clss-clss-insert*)
ultimately show *unmark-l a* ∪ *set-mset* ($clauses_{NOT}$ *T*)
⊨*ps unmark-l 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* $clauses_{NOT}$ *S* ⟷ *I*⊨*sextm* $clauses_{NOT}$ *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* $clauses_{NOT}$ *S* **and**
I ⊆ *J* **and**
tot: *total-over-m J* (*set-mset* ({#*C*#} + $clauses_{NOT}$ *S*)) **and**
cons: *consistent-interp J*
then have *J* ⊨*sm* $clauses_{NOT}$ *S* **unfolding** *true-clss-ext-def* **by** *auto*

moreover
with ⟨$clauses_{NOT}$ *S*⊨*pm C*⟩ **have** *J* ⊨ *C*

using *tot cons* **unfolding** *true-clss-cls-def* **by** *auto*
ultimately have $J \models sm\ \{\#C\#\} + clauses_{NOT}\ S$ **by** *auto*
}
then have H: $I \models sextm\ (clauses_{NOT}\ S) \Longrightarrow I \models sext\ insert\ C\ (set\text{-}mset\ (clauses_{NOT}\ S))$
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
$I \models sext\ set\text{-}mset\ (clauses_{NOT}\ S) - \{C\}$ **and**
$I \subseteq J$ **and**
tot: $total\text{-}over\text{-}m\ J\ (set\text{-}mset\ (clauses_{NOT}\ S))$ **and**
cons: *consistent-interp J*
then have $J \models s\ set\text{-}mset\ (clauses_{NOT}\ S) - \{C\}$
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 order-refl*
set-mset-minus-replicate-mset(*1*))
ultimately have $J \models sm\ (clauses_{NOT}\ S)$ **by** (*metis insert-Diff-single true-clss-insert*)
}
then have H: $I \models sext\ set\text{-}mset\ (clauses_{NOT}\ S) - \{C\} \Longrightarrow I \models sextm\ (clauses_{NOT}\ S)$
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*

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-mm* (*clauses*$_{NOT}$ *S*) $\subseteq$ *A* **and**
    atms-trail-S: *atm-of* '(*lits-of-l* (*trail S*)) $\subseteq$ *A*
  shows *atm-of* ' (*lits-of-l* (*trail T*)) $\subseteq$ *A* $\wedge$ *atms-of-mm* (*clauses*$_{NOT}$ *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-mm* (*clauses*$_{NOT}$ *U*) $\subseteq$ *A*
    using *cdcl*$_{NOT}$*-atms-of-ms-clauses-decreasing*[*OF cdcl*$_{NOT}$] *IH n-d* ‹*inv T*› **by** *fast*
  moreover
    have *atm-of* '(*lits-of-l* (*trail U*)) $\subseteq$ *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*$_{NOT}$ *S*) (*get-all-ann-decomposition* (*trail S*))
  shows
    all-decomposition-implies-m (*clauses*$_{NOT}$ *T*) (*get-all-ann-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*$\models$*sextm clauses*$_{NOT}$ *S* $\longleftrightarrow$ *I*$\models$*sextm clauses*$_{NOT}$ *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* $\longleftrightarrow$ (*finite A* $\wedge$ *inv S* $\wedge$ *atms-of-mm* (*clauses*$_{NOT}$ *S*) $\subseteq$ *atms-of-ms A*
  $\wedge$ *atm-of* ' *lits-of-l* (*trail S*) $\subseteq$ *atms-of-ms A* $\wedge$ *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 $\equiv$ *learn S T* $\vee$ *forget*$_{NOT}$ *S T*

lemma *rtranclp-learn-or-forget-cdcl*$_{NOT}$:
  learn-or-forget** *S T* $\Longrightarrow$ *cdcl*$_{NOT}$** *S T*

using *rtranclp-mono*[*of learn-or-forget* $cdcl_{NOT}$] **by** (*blast intro*: $cdcl_{NOT}$*.c-learn* $cdcl_{NOT}$*.c-forget*$_{NOT}$)

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** *presburger*
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**
  $\neg$*learn* $(f\ i)$ $(f\ (Suc\ i))$ $\wedge$ $\neg$*forget*$_{NOT}$ $(f\ i)$ $(f\ (Suc\ i))$ **and**
  $\forall\, k{<}i.$ *learn-or-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-or-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
def $g \equiv \lambda n.\ f\ (n + Suc\ i)$
have *dpll-bj* $(f\ i)$ $(g\ 0)$
  using ‹$\neg$ *learn* $(f\ i)$ $(f\ (Suc\ i))$ $\wedge$ $\neg$ *forget*$_{NOT}$ $(f\ i)$ $(f\ (Suc\ i))$› *cdcl*$_{NOT}$ *cdcl*$_{NOT}$*.cases*
  g-def **by** *auto*
{
  fix j
  assume $j \le i$
  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*
      ‹$\forall\, k{<}i.$ *learn* $(f\ k)$ $(f\ (Suc\ k))$ $\vee$ *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))
      $- \mu_C$ (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))
    $- \mu_C$ (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 $\bigwedge i.$ *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: $\bigwedge i.\ i{\ge}j \Longrightarrow$ *learn-or-forget* $(g\ i)$ $(g\ (Suc\ i))$
  using *IH* **unfolding** μ[*symmetric*] **by** *presburger*
show *?thesis*
  proof
    {
      fix k
      assume $k \ge j + Suc\ i$

```
            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: ∃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 i0 where ¬ learn (f i0) (f (Suc i0)) ∧ ¬forget_NOT (f i0) (f (Suc i0))
            using j by auto
          then have {i. i≤i0 ∧ ¬ learn (f i) (f (Suc i)) ∧ ¬forget_NOT (f i) (f (Suc i))} ≠ {}
            by auto
          let ?I = {i. i≤i0 ∧ ¬ 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
      have dpll-bj (f i) (f (Suc i))
        using ‹¬ learn (f i) (f (Suc i)) ∧ ¬ forget_NOT (f i) (f (Suc i))› cdcl_NOT cdcl_NOT.cases
        by blast
      {
        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 show False
        using learn-or-forget-dpll-μC[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: $\bigwedge f\ j.\ \neg\ (\forall\ i{\geq}j.\ \textit{learn-or-forget}\ (f\ i)\ (f\ (\textit{Suc}\ i)))$
shows $\textit{wf}\ \{(T,\ S).\ \textit{cdcl}_{NOT}\ S\ T \wedge \textit{cdcl}_{NOT}\textit{-NOT-all-inv}\ A\ S\}$
(**is** $\textit{wf}\ \{(T,\ S).\ \textit{cdcl}_{NOT}\ S\ T \wedge \textit{?inv}\ S\}$)
unfolding *wf-iff-no-infinite-down-chain*
proof (*rule ccontr*)
assume $\neg\ \neg\ (\exists f.\ \forall\ i.\ (f\ (\textit{Suc}\ i),\ f\ i) \in \{(T,\ S).\ \textit{cdcl}_{NOT}\ S\ T \wedge \textit{?inv}\ S\})$
then obtain f **where**
$\forall\ i.\ \textit{cdcl}_{NOT}\ (f\ i)\ (f\ (\textit{Suc}\ i)) \wedge \textit{?inv}\ (f\ i)$
by *fast*
then have $\exists\ j.\ \forall\ i{\geq}j.\ \textit{learn-or-forget}\ (f\ i)\ (f\ (\textit{Suc}\ i))$
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*:
$\textit{cdcl}_{NOT}{}^{++}\ S\ T \wedge \textit{cdcl}_{NOT}\textit{-NOT-all-inv}\ A\ S \longleftrightarrow (\lambda S\ T.\ \textit{cdcl}_{NOT}\ S\ T \wedge \textit{cdcl}_{NOT}\textit{-NOT-all-inv}\ A\ S)^{++}\ S\ T$
(**is** $\textit{?A} \wedge \textit{?I} \longleftrightarrow \textit{?B}$)
proof
assume $\textit{?A} \wedge \textit{?I}$
then have *?A* **and** *?I* **by** *blast+*
then show *?B*
apply *induction*
apply (*simp add*: *tranclp.r-into-trancl*)
by (*subst tranclp.simps*) (*auto intro*: *cdcl*$_{NOT}$*-NOT-all-inv tranclp-into-rtranclp*)
next
assume *?B*
then have *?A* **by** *induction auto*
moreover have *?I* **using** ‹*?B*› *tranclpD* **by** *fastforce*
ultimately show $\textit{?A} \wedge \textit{?I}$ **by** *blast*
qed

lemma *wf-tranclp-cdcl*$_{NOT}$*-no-learn-and-forget-infinite-chain*:
assumes
no-infinite-lf: $\bigwedge f\ j.\ \neg\ (\forall\ i{\geq}j.\ \textit{learn-or-forget}\ (f\ i)\ (f\ (\textit{Suc}\ i)))$
shows $\textit{wf}\ \{(T,\ S).\ \textit{cdcl}_{NOT}{}^{++}\ S\ T \wedge \textit{cdcl}_{NOT}\textit{-NOT-all-inv}\ A\ S\}$
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*$_{NOT}$ *S*) (*get-all-ann-decomposition* (*trail S*))
shows *unsatisfiable* (*set-mset* (*clauses*$_{NOT}$ *S*))
$\vee$ (*trail S* $\models$*asm clauses*$_{NOT}$ *S* $\wedge$ *satisfiable* (*set-mset* (*clauses*$_{NOT}$ *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*$_{NOT}$ *S*) (*get-all-ann-decomposition* (*trail S*))
shows *unsatisfiable* (*set-mset* (*clauses*$_{NOT}$ *T*))
∨ (*trail T* ⊨*asm clauses*$_{NOT}$ *T* ∧ *satisfiable* (*set-mset* (*clauses*$_{NOT}$ *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*$_{NOT}$ *T*) (*get-all-ann-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*

Termination

To prove termination we need to restrict learn and forget. Otherwise we could forget and relearn the exact same clause over and over. A first idea is to forbid removing clauses that can be used to backjump. This does not change the rules of the calculus. A second idea is to "merge" backjump and learn: that way, though closer to implementation, needs a change of the rules, since the backjump-rule learns the clause used to backjump.

Restricting learn and forget

locale *conflict-driven-clause-learning-learning-before-backjump-only-distinct-learnt* =
dpll-state trail clauses$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$ +
conflict-driven-clause-learning trail clauses$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$
inv backjump-conds propagate-conds
$\lambda C\ S.$ *distinct-mset C* ∧ ¬*tautology C* ∧ *learn-restrictions C S* ∧
(∃ *F K d F' C' L*. *trail S* = *F'* @ *Decided K* # *F* ∧ *C* = *C'* + {#*L*#} ∧ *F* ⊨*as CNot C'*
∧ *C'* + {#*L*#} ∉# *clauses*$_{NOT}$ *S*)
$\lambda C\ S.$ ¬(∃ *F' F K d L*. *trail S* = *F'* @ *Decided K* # *F* ∧ *F* ⊨*as CNot* (*remove1-mset L C*))
∧ *forget-restrictions C S*
for
trail :: *'st* ⇒ (*'v*, *unit*) *ann-lits* **and**
clauses$_{NOT}$ :: *'st* ⇒ *'v clauses* **and**
prepend-trail :: (*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒*'st* **and**
add-cls$_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
remove-cls$_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
inv :: *'st* ⇒ *bool* **and**
backjump-conds :: *'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool* **and**
propagate-conds :: (*'v*, *unit*) *ann-lit* ⇒ *'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:
$\bigwedge C\ F\ K\ F'\ C'\ L\ T.$ $clauses_{NOT}$ $S \models pm$ $C \Longrightarrow$
atms-of $C \subseteq$ *atms-of-mm* ($clauses_{NOT}$ S) $\cup$ *atm-of* ' (*lits-of-l* (*trail* S)) $\Longrightarrow$
distinct-mset $C \Longrightarrow$
$\neg$ *tautology* $C \Longrightarrow$
learn-restrictions C $S \Longrightarrow$
trail $S = F'$ @ *Decided* K # $F \Longrightarrow$
$C = C' + \{\#L\#\} \Longrightarrow$
$F \models as$ *CNot* $C' \Longrightarrow$
$C' + \{\#L\#\} \notin\#$ $clauses_{NOT}$ $S \Longrightarrow$
$T \sim$ *add-cls*$_{NOT}$ C $S \Longrightarrow$
P S T **and**
forgetting: $\bigwedge C\ T.$ *removeAll-mset* C ($clauses_{NOT}$ S) $\models pm$ $C \Longrightarrow$
$C \in\#$ $clauses_{NOT}$ $S \Longrightarrow$
$\neg(\exists F'\ F\ K\ L.$ *trail* $S = F'$ @ *Decided* 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* ($clauses_{NOT}$ T $-$ $clauses_{NOT}$ S)
$\subseteq$ *simple-clss* (*atms-of-mm* ($clauses_{NOT}$ S) $\cup$ *atm-of* ' *lits-of-l* (*trail* S))
proof
fix C **assume** C: $C \in$ *set-mset* ($clauses_{NOT}$ T $-$ $clauses_{NOT}$ 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-mm* ($clauses_{NOT}$ S) $\cup$ *atm-of* ' *lits-of-l* (*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-mm* ($clauses_{NOT}$ S) $\cup$ *atm-of* ' *lits-of-l* (*trail* S))
by *auto*
ultimately show $C \in$ *simple-clss* (*atms-of-mm* ($clauses_{NOT}$ S) $\cup$ *atm-of* ' *lits-of-l* (*trail* S))
using *simple-clss-mono* **by** (*metis* (*no-types*) *insert-subset mk-disjoint-insert*)
qed

definition *conflicting-bj-clss* $S \equiv$

$\{C+\{\#L\#\} \mid C\ L.\ C+\{\#L\#\} \in\#\ clauses_{NOT}\ S \wedge distinct\text{-}mset\ (C+\{\#L\#\})$
$\wedge \neg tautology\ (C+\{\#L\#\})$
$\wedge (\exists F'\ K\ F.\ trail\ S = F' @ Decided\ K \# F \wedge F \models as\ CNot\ C)\}$

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-remove-cls$_{NOT}$′*[*simp*]:
$T \sim$ *remove-cls$_{NOT}$ C S* $\Longrightarrow$ *conflicting-bj-clss T* = *conflicting-bj-clss S* − $\{C\}$
unfolding *conflicting-bj-clss-def* **by** *fastforce*

lemma *conflicting-bj-clss-add-cls$_{NOT}$-state-eq*:
assumes
T: $T \sim$ *add-cls$_{NOT}$ C′ S* **and**
n-d: *no-dup* (*trail S*)
shows *conflicting-bj-clss T*
= *conflicting-bj-clss S*
$\cup$ (*if* $\exists C\ L.\ C' = C + \{\#L\#\} \wedge distinct\text{-}mset\ (C+\{\#L\#\}) \wedge \neg tautology\ (C+\{\#L\#\})$
$\wedge (\exists F'\ K\ d\ F.\ trail\ S = F' @ Decided\ K \# F \wedge F \models as\ CNot\ C)$
then $\{C'\}$ *else* $\{\}$)
proof −
def $P \equiv \lambda C\ L\ T.\ distinct\text{-}mset\ (C + \{\#L\#\}) \wedge \neg\ tautology\ (C + \{\#L\#\}) \wedge$
$(\exists F'\ K\ F.\ trail\ T = F' @ Decided\ K \# F \wedge F \models as\ CNot\ C)$
have *conf*: $\bigwedge T.\ conflicting\text{-}bj\text{-}clss\ T = \{C + \{\#L\#\} \mid C\ L.\ C + \{\#L\#\} \in\#\ clauses_{NOT}\ T \wedge P\ C\ L\ T\}$
unfolding *conflicting-bj-clss-def P-def* **by** *auto*
have *P-S-T*: $\bigwedge C\ L.\ P\ C\ L\ T = P\ C\ L\ S$
using *T n-d* **unfolding** *P-def* **by** *auto*
have *P*: *conflicting-bj-clss T* = $\{C + \{\#L\#\} \mid C\ L.\ C + \{\#L\#\} \in\#\ clauses_{NOT}\ S \wedge P\ C\ L\ T\} \cup$
$\{C + \{\#L\#\} \mid C\ L.\ C + \{\#L\#\} \in\# \{\#C'\#\} \wedge P\ C\ L\ T\}$
using *T n-d* **unfolding** *conf* **by** *auto*
moreover have $\{C + \{\#L\#\} \mid C\ L.\ C + \{\#L\#\} \in\#\ clauses_{NOT}\ S \wedge P\ C\ L\ T\}$ = *conflicting-bj-clss S*
using *T n-d* **unfolding** *P-def conflicting-bj-clss-def* **by** *auto*
moreover have $\{C + \{\#L\#\} \mid C\ L.\ C + \{\#L\#\} \in\# \{\#C'\#\} \wedge P\ C\ L\ T\}$ =
(*if* $\exists C\ L.\ C' = C + \{\#L\#\} \wedge P\ C\ L\ S$ *then* $\{C'\}$ *else* $\{\}$)
using *n-d T* **by** (*force simp*: *P-S-T*)
ultimately show *?thesis* **unfolding** *P-def* **by** *presburger*
qed

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* $\exists C\ L.\ C' = C + \{\#L\#\} \wedge distinct\text{-}mset\ (C+\{\#L\#\}) \wedge \neg tautology\ (C+\{\#L\#\})$
$\wedge (\exists F'\ K\ d\ F.\ trail\ S = F' @ Decided\ K \# F \wedge F \models as\ CNot\ C)$
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$_{NOT}$ 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* ($clauses_{NOT}$ *S*)))

lemma *remove1-mset-single-add-if*:
remove1-mset L (*C* + {#*L'*#}) = (*if L* = *L' then C else remove1-mset L C* + {#*L'*#})
by (*auto simp*: *multiset-eq-iff*)

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** (*elim* $forget_{NOT}E$)
apply *rule*
 apply (*subst conflicting-bj-clss-remove-cls*$_{NOT}$*'*[*of T*], *simp*)
 apply (*fastforce simp*: *conflicting-bj-clss-def remove1-mset-single-add-if split*: *if-splits*)
apply *fastforce*
done

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* ($clauses_{NOT}$ *S*)) > *0*
using $forget_{NOT}$ **by** (*elim* $forget_{NOT}E$) (*auto simp*: *size-mset-removeAll-mset-le-iff card-gt-0-iff*)
then have *card* (*set-mset* ($clauses_{NOT}$ *T*)) < *card* (*set-mset* ($clauses_{NOT}$ *S*))
using $forget_{NOT}$ **by** (*elim* $forget_{NOT}E$) (*auto simp*: *size-mset-removeAll-mset-le-iff*)
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* $\vee$ *Q a*) $\wedge$ *S a*} = (*if S b then* {*b*} *else* {}) $\cup$ {*a*. *Q a* $\wedge$ *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-mm* ($clauses_{NOT}$ *S*) $\cup$ *atm-of* ' *lits-of-l* (*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-mm* ($clauses_{NOT}$ *T*) $\cup$ *atm-of* ' *lits-of-l* (*trail T*))
= (*atms-of-mm* ($clauses_{NOT}$ *S*) $\cup$ *atm-of* ' *lits-of-l* (*trail S*))

using *learnST n-d* **by** (*elim* $learn_{NOT}E$) *auto*

then have *card* (*atms-of-mm* ($clauses_{NOT}$ *T*) ∪ *atm-of* ‘ *lits-of-l* (*trail T*))
= *card* (*atms-of-mm* ($clauses_{NOT}$ *S*) ∪ *atm-of* ‘ *lits-of-l* (*trail S*))
by (*auto intro*!: *card-mono*)
then have *3*: (*3*::*nat*) ^ *card* (*atms-of-mm* ($clauses_{NOT}$ *T*) ∪ *atm-of* ‘ *lits-of-l* (*trail T*))
= *3* ^ *card* (*atms-of-mm* ($clauses_{NOT}$ *S*) ∪ *atm-of* ‘ *lits-of-l* (*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′* @ *Decided K* # *F* **and**
C: *C* = *C′* + {#*L*#} **and**
F: *F* ⊨*as CNot C′* **and**
C-S:*C′* + {#*L*#} ∉# $clauses_{NOT}$ *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-mm* ($clauses_{NOT}$ *T*) ∪ *atm-of* ‘ *lits-of-l* (*trail T*))
by *auto*
have *1*:*atms-of-ms* (*conflicting-bj-clss T*) ⊆ *atms-of-mm* ($clauses_{NOT}$ *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-mm* ($clauses_{NOT}$ *T*) ∪ *atm-of* ‘ *lits-of-l* (*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-mm* ($clauses_{NOT}$ *T*) ∪ *atm-of* ‘ *lits-of-l* (*trail T*))
≥ *card* (*conflicting-bj-clss T*)
by (*meson Nat.le-trans simple-clss-card simple-clss-finite card-mono fin′*)
have *atms-of-mm* ($clauses_{NOT}$ *T*) ∪ *atm-of* ‘ *lits-of-l* (*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-l* (*trail S*) ⊆ *atms-of-ms A* and in the clauses *atms-of-mm* ($clauses_{NOT}$ *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* ($clauses_{NOT}$ *T*)))
lemma $cdcl_{NOT}$*-decreasing-measure*:
assumes
$cdcl_{NOT}$ *S T* **and**
inv: *inv S* **and**
atm-clss: *atms-of-mm* ($clauses_{NOT}$ *S*) ⊆ *atms-of-ms A* **and**
atm-lits: *atm-of ‘ lits-of-l* (*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 n-d*, *of atms-of-ms A*] *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-mm* ($clauses_{NOT}$ *S*) ⊆ *atms-of-ms A* ∧ *atm-of ‘ lits-of-l* (*trail S*) ⊆ *atms-of-ms A*
∧ *no-dup* (*trail S*)
∧ *inv S*)

$\wedge$ *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 clause 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 clause 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*$_{NOT}$ *T*))

lemma *cdcl*$_{NOT}$*-decreasing-measure'*:
assumes
cdcl$_{NOT}$ *S T* **and**
inv: *inv S* **and**
atms-clss: *atms-of-mm* (*clauses*$_{NOT}$ *S*) ⊆ *atms-of-ms A* **and**
atms-trail: *atm-of* ‘ *lits-of-l* (*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*$_{NOT}$ *T* = *clauses*$_{NOT}$ *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-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*)) − $\mu_C{}'$ *A S*)
∗ (*1* + *3* ^ *card* (*atms-of-ms A*))
+ *conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) *S*

by *linarith*
then have $((2 + \mathit{card}\ (\mathit{atms\text{-}of\text{-}ms}\ A))\ \widehat{}\ (1 + \mathit{card}\ (\mathit{atms\text{-}of\text{-}ms}\ A)) - \mu_C{}'\ A\ T)$
$* (1 + 3\ \widehat{}\ \mathit{card}\ (\mathit{atms\text{-}of\text{-}ms}\ A)) * 2$
$+ \mathit{conflicting\text{-}bj\text{-}clss\text{-}yet}\ (\mathit{card}\ (\mathit{atms\text{-}of\text{-}ms}\ A))\ T * 2$
$< ((2 + \mathit{card}\ (\mathit{atms\text{-}of\text{-}ms}\ A))\ \widehat{}\ (1 + \mathit{card}\ (\mathit{atms\text{-}of\text{-}ms}\ A)) - \mu_C{}'\ A\ S)$
$* (1 + 3\ \widehat{}\ \mathit{card}\ (\mathit{atms\text{-}of\text{-}ms}\ A)) * 2$
$+ \mathit{conflicting\text{-}bj\text{-}clss\text{-}yet}\ (\mathit{card}\ (\mathit{atms\text{-}of\text{-}ms}\ A))\ S * 2$
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*)
using *C′*
by (*metis* (*no-types*, *hide-lams*) *Un-subset-iff simple-clss-mono*
contra-subsetD dist distinct-mset-not-tautology-implies-in-simple-clss
dual-order.trans atms-C atms-clss atms-trail tauto)
moreover have *conflicting-bj-clss S* ⊆ *simple-clss* (*atms-of-ms A*)
proof
fix *x* :: *′v clause*
assume *x* ∈ *conflicting-bj-clss S*
then have $x \in\# \mathit{clauses}_{NOT}\ S \wedge \mathit{distinct\text{-}mset}\ x \wedge \neg\ \mathit{tautology}\ x$
unfolding *conflicting-bj-clss-def* **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
set-rev-mp)
qed
ultimately show *?thesis*
by *auto*
qed
then have *card* (*insert C* (*conflicting-bj-clss S*)) ≤ *3* ^ (*card* (*atms-of-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)
moreover have [*simp*]: $\mathit{conflicting\text{-}bj\text{-}clss}\ (\mathit{add\text{-}cls}_{NOT}\ C\ S) = \mathit{conflicting\text{-}bj\text{-}clss}\ S \cup \{C\}$
using *dist tauto F-C* **by** ($\mathit{subst}\ \mathit{conflicting\text{-}bj\text{-}clss\text{-}add\text{-}cls}_{NOT}[\mathit{OF}\ n\text{-}d]$) (*force simp*: *C′ tr-S n-d*)
ultimately have [*simp*]: *conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) *S*
$= \mathit{Suc}\ (\mathit{conflicting\text{-}bj\text{-}clss\text{-}yet}\ (\mathit{card}\ (\mathit{atms\text{-}of\text{-}ms}\ A))\ (\mathit{add\text{-}cls}_{NOT}\ C\ S))$
by *simp*
have *1*: $\mathit{clauses}_{NOT}\ T = \mathit{clauses}_{NOT}\ (\mathit{add\text{-}cls}_{NOT}\ C\ S)$ **using** *T* **by** *auto*
have *2*: *conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) *T*
$= \mathit{conflicting\text{-}bj\text{-}clss\text{-}yet}\ (\mathit{card}\ (\mathit{atms\text{-}of\text{-}ms}\ A))\ (\mathit{add\text{-}cls}_{NOT}\ C\ S)$
using *T* **unfolding** *conflicting-bj-clss-def* **by** *auto*
have *3*: $\mu_C{}'\ A\ T = \mu_C{}'\ A\ (\mathit{add\text{-}cls}_{NOT}\ C\ S)$
using *T* **unfolding** $\mu_C{}'$*-def* **by** *auto*
have $((2 + \mathit{card}\ (\mathit{atms\text{-}of\text{-}ms}\ A))\ \widehat{}\ (1 + \mathit{card}\ (\mathit{atms\text{-}of\text{-}ms}\ A)) - \mu_C{}'\ A\ (\mathit{add\text{-}cls}_{NOT}\ C\ S))$
$* (1 + 3\ \widehat{}\ \mathit{card}\ (\mathit{atms\text{-}of\text{-}ms}\ A)) * 2$
$= ((2 + \mathit{card}\ (\mathit{atms\text{-}of\text{-}ms}\ A))\ \widehat{}\ (1 + \mathit{card}\ (\mathit{atms\text{-}of\text{-}ms}\ A)) - \mu_C{}'\ A\ S)$
$* (1 + 3\ \widehat{}\ \mathit{card}\ (\mathit{atms\text{-}of\text{-}ms}\ A)) * 2$

using *n-d* **unfolding** μ_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*$_{NOT}$ (*add-cls*$_{NOT}$ *C S*)))
$<$ *conflicting-bj-clss-yet* (*card* (*atms-of-ms A*)) *S* $*$ *2*
$+$ *card* (*set-mset* (*clauses*$_{NOT}$ *S*))
by (*simp add*: *C' C-new n-d*)
ultimately show *?case* **unfolding** μ_{CDCL}*'-def 1 2 3* **by** *presburger*
next
case (*forget*$_{NOT}$ *C T*) **note** *T* = *this*(*4*)
have [*simp*]: μ_C*' A* (*remove-cls*$_{NOT}$ *C S*) = μ_C*' A S*
unfolding μ_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*$_{NOT}$ *T*)) $<$ *card* (*set-mset* (*clauses*$_{NOT}$ *S*))
by (*metis T card-Diff1-less clauses-remove-cls*$_{NOT}$ *finite-set-mset forget*$_{NOT}$*.hyps*(*2*)
order-refl set-mset-minus-replicate-mset(*1*) *state-eq*$_{NOT}$*-clauses*)
ultimately show *?case* **unfolding** μ_{CDCL}*'-def*
using *T* ‹μ_C*' A* (*remove-cls*$_{NOT}$ *C S*) = μ_C*' A S*› **by** (*metis* (*no-types*) *add-le-cancel-left*
μ_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-mm (*clauses*$_{NOT}$ *S*) $\subseteq$ *A* **and**
atm-of '(*lits-of-l* (*trail S*)) $\subseteq$ *A* **and**
n-d: *no-dup* (*trail S*) **and**
fin-A[*simp*]: *finite A*
shows *set-mset* (*clauses*$_{NOT}$ *T*) $\subseteq$ *set-mset* (*clauses*$_{NOT}$ *S*) $\cup$ *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* $\subseteq$ *A*
using *atms-C atms-clss-S atms-trail-S* **by** *fast*
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-mm (*clauses*$_{NOT}$ *S*) ⊆ *A* **and**
atm-of ‘(*lits-of-l* (*trail S*)) ⊆ *A* **and**
n-d: *no-dup* (*trail S*) **and**
finite: *finite A*
shows *set-mset* (*clauses*$_{NOT}$ *T*) ⊆ *set-mset* (*clauses*$_{NOT}$ *S*) ∪ *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-mm* (*clauses*$_{NOT}$ *T*) ⊆ *A* **and** *atm-of* ‘ *lits-of-l* (*trail T*) ⊆ *A*
using *rtranclp-cdcl*$_{NOT}$*-trail-clauses-bound*[*OF st*] *inv atms-clss-S atms-trail-S n-d* **by** *auto*
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*$_{NOT}$ *U*) ⊆ *set-mset* (*clauses*$_{NOT}$ *T*) ∪ *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-mm (*clauses*$_{NOT}$ *S*) ⊆ *A* **and**
atm-of ‘(*lits-of-l* (*trail S*)) ⊆ *A* **and**
n-d: *no-dup* (*trail S*) **and**
finite: *finite A*
shows *card* (*set-mset* (*clauses*$_{NOT}$ *T*)) ≤ *card* (*set-mset* (*clauses*$_{NOT}$ *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-mm (*clauses*$_{NOT}$ *S*) ⊆ *A* **and**
atm-of ‘(*lits-of-l* (*trail S*)) ⊆ *A* **and**
n-d: *no-dup* (*trail S*) **and**
finite: *finite A*
shows *card* {*C*|*C*. *C* ∈# *clauses*$_{NOT}$ *T* ∧ (*tautology C* ∨ ¬*distinct-mset C*)}
≤ *card* {*C*|*C*. *C*∈# *clauses*$_{NOT}$ *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**
    NA: *atms-of-mm* (*clauses$_{NOT}$ S*) $\subseteq$ *A* **and**
    MA: *atm-of* ` (*lits-of-l* (*trail S*)) $\subseteq$ *A* **and**
    n-d: *no-dup* (*trail S*) **and**
    finite: *finite A*
  shows *card* (*set-mset* (*clauses$_{NOT}$ T*))
  $\leq$ *card* {*C*. *C* $\in$# *clauses$_{NOT}$ S* $\wedge$ (*tautology C* $\vee$ $\neg$*distinct-mset C*)} + *3* ^ (*card A*)
    (**is** *card ?T* $\leq$ *card ?S* + -)
  using *rtranclp-cdcl$_{NOT}$-clauses-bound*[*OF assms*] *finite*
proof −
  have $\bigwedge$*x*. *x* $\in$# *clauses$_{NOT}$ T* $\Longrightarrow$ $\neg$ *tautology x* $\Longrightarrow$ *distinct-mset x* $\Longrightarrow$ *x* $\in$ *simple-clss A*
    using *rtranclp-cdcl$_{NOT}$-clauses-bound*[*OF assms*] **by** (*metis* (*no-types*, *hide-lams*) *Un-iff NA*
      atms-of-atms-of-ms-mono simple-clss-mono contra-subsetD subset-trans
      distinct-mset-not-tautology-implies-in-simple-clss)
  then have *set-mset* (*clauses$_{NOT}$ T*) $\subseteq$ *?S* $\cup$ *simple-clss A*
    using *rtranclp-cdcl$_{NOT}$-clauses-bound*[*OF assms*] **by** *auto*
  then have *card*(*set-mset* (*clauses$_{NOT}$ T*)) $\leq$ *card* (*?S* $\cup$ *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 clause set* $\Rightarrow$ *'st* $\Rightarrow$ *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* $\in$# *clauses$_{NOT}$ S* $\wedge$ (*tautology C* $\vee$ $\neg$*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-mm (*clauses$_{NOT}$ S*) $\subseteq$ *atms-of-ms A* **and**
    atm-of `(*lits-of-l* (*trail S*)) $\subseteq$ *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 *μ_{CDCL}' A U* $\leq$ *μ_{CDCL}'-bound A S*
proof −
  have ((*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*)) − *μ_C' A U*)
    $\leq$ (*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*)) − *μ_C' A U*)
      $*$ (*1* + *3* ^ *card* (*atms-of-ms A*)) $*$ *2*
    $\leq$ (*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)) $T * 2 \leq 2 * 3$ ^ *card* (*atms-of-ms* A)
by *linarith*
moreover have *card* (*set-mset* (*clauses*$_{NOT}$ U))
$\leq$ *card* $\{C.\ C \in\#$ *clauses*$_{NOT}$ $S \wedge$ (*tautology* $C \vee \neg$*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 μ_{CDCL}*'-def* μ_{CDCL}*'-bound-def* **by** *linarith*
qed

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

lemma *rtranclp-*μ_{CDCL}*'-bound-decreasing*:
assumes
cdcl$_{NOT}$** S T **and**
inv S **and**
atms-of-mm (*clauses*$_{NOT}$ S) $\subseteq$ *atms-of-ms* A **and**
atm-of ‘(*lits-of-l* (*trail* S)) $\subseteq$ *atms-of-ms* A **and**
n-d: *no-dup* (*trail* S) **and**
finite[*simp*]: *finite* (*atms-of-ms* A)
shows μ_{CDCL}*'-bound* A $T \leq \mu_{CDCL}$*'-bound* A S
proof −
have $\{C.\ C \in\#$ *clauses*$_{NOT}$ $T \wedge$ (*tautology* $C \vee \neg$ *distinct-mset* C)$\}$
$\subseteq \{C.\ C \in\#$ *clauses*$_{NOT}$ $S \wedge$ (*tautology* $C \vee \neg$ *distinct-mset* C)$\}$ (**is** *?T* $\subseteq$ *?S*)
proof (*rule Set.subsetI*)
fix C **assume** $C \in$ *?T*
then have *C-T*: $C \in\#$ *clauses*$_{NOT}$ T **and** *t-d*: *tautology* $C \vee \neg$ *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*$_{NOT}$ $T \wedge$ (*tautology* $C \vee \neg$ *distinct-mset* C)$\} \leq$
card $\{C.\ C \in\#$ *clauses*$_{NOT}$ $S \wedge$ (*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*

1.2.5 CDCL with restarts

Definition

locale *restart-ops* =
  fixes
    $cdcl_{NOT}$ :: $'st \Rightarrow 'st \Rightarrow bool$ **and**
    restart :: $'st \Rightarrow 'st \Rightarrow bool$
begin
inductive $cdcl_{NOT}$*-raw-restart* :: $'st \Rightarrow 'st \Rightarrow bool$ **where**
$cdcl_{NOT}$ S $T \Longrightarrow cdcl_{NOT}$*-raw-restart* S T |
restart S $T \Longrightarrow cdcl_{NOT}$*-raw-restart* S T

end

locale *conflict-driven-clause-learning-with-restarts* =
  conflict-driven-clause-learning trail $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
    inv backjump-conds propagate-conds learn-cond forget-cond
  for
    trail :: $'st \Rightarrow ('v, unit)$ *ann-lits* **and**
    $clauses_{NOT}$ :: $'st \Rightarrow 'v$ *clauses* **and**
    prepend-trail :: $('v, unit)$ *ann-lit* $\Rightarrow 'st \Rightarrow 'st$ **and**
    tl-trail :: $'st \Rightarrow 'st$ **and**
    $add\text{-}cls_{NOT}$ :: $'v$ *clause* $\Rightarrow 'st \Rightarrow 'st$ **and**
    $remove\text{-}cls_{NOT}$ :: $'v$ *clause* $\Rightarrow 'st \Rightarrow 'st$ **and**
    inv :: $'st \Rightarrow bool$ **and**
    backjump-conds :: $'v$ *clause* $\Rightarrow 'v$ *clause* $\Rightarrow 'v$ *literal* $\Rightarrow 'st \Rightarrow 'st \Rightarrow bool$ **and**
    propagate-conds :: $('v, unit)$ *ann-lit* $\Rightarrow 'st \Rightarrow bool$ **and**
    learn-cond forget-cond :: $'v$ *clause* $\Rightarrow 'st \Rightarrow bool$
begin

lemma $cdcl_{NOT}$*-iff-*$cdcl_{NOT}$*-raw-restart-no-restarts*:
  $cdcl_{NOT}$ S $T \longleftrightarrow$ *restart-ops*.$cdcl_{NOT}$*-raw-restart* $cdcl_{NOT}$ (λ- -. *False*) S T
  (**is** *?C* S $T \longleftrightarrow$ *?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

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**
    $\mu :: 'bound \Rightarrow 'st \Rightarrow nat$ **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: $\bigwedge A\ S\ T.\ cdcl_{NOT}\text{-}inv\ S \Longrightarrow bound\text{-}inv\ A\ S \Longrightarrow cdcl_{NOT}\ S\ T \Longrightarrow bound\text{-}inv\ A\ T$ **and**
    $cdcl_{NOT}$*-measure*: $\bigwedge A\ S\ T.\ cdcl_{NOT}\text{-}inv\ S \Longrightarrow bound\text{-}inv\ A\ S \Longrightarrow cdcl_{NOT}\ S\ T \Longrightarrow \mu\ A\ T < \mu\ A\ S$ **and**
    measure-bound2: $\bigwedge A\ T\ U.\ cdcl_{NOT}\text{-}inv\ T \Longrightarrow bound\text{-}inv\ A\ T \Longrightarrow cdcl_{NOT}^{**}\ T\ U$
      $\Longrightarrow \mu\ A\ U \leq \mu\text{-}bound\ A\ T$ **and**
    measure-bound4: $\bigwedge A\ T\ U.\ cdcl_{NOT}\text{-}inv\ T \Longrightarrow bound\text{-}inv\ A\ T \Longrightarrow cdcl_{NOT}^{**}\ T\ U$
      $\Longrightarrow \mu\text{-}bound\ A\ U \leq \mu\text{-}bound\ A\ T$ **and**
    $cdcl_{NOT}$*-restart-inv*: $\bigwedge A\ U\ V.\ cdcl_{NOT}\text{-}inv\ U \Longrightarrow restart\ U\ V \Longrightarrow bound\text{-}inv\ A\ U \Longrightarrow bound\text{-}inv\ A\ V$
      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
    $(cdcl_{NOT}\widehat{\ \ }n)\ S\ T$ **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
    $(cdcl_{NOT}\widehat{\ \ }n)\ S\ T$ **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}$$\widehat{}\widehat{}$(*Suc n*)) *S T* **and**
bound-inv A S
cdcl$_{NOT}$*-inv S*
shows μ *A T* $<$ μ *A S* $-$ *n*
using *assms*
proof (*induction n arbitrary*: *T*)
case *0*
then show *?case* **using** *cdcl*$_{NOT}$*-measure* **by** *auto*
next
case (*Suc n*) **note** *IH* = *this*(*1*)[*OF - this*(*3*) *this*(*4*)] **and** *S-T* = *this*(*2*) **and** *b-inv* = *this*(*3*) **and** *c-inv* = *this*(*4*)
obtain *U* :: *'st* **where** *S-U*: (*cdcl*$_{NOT}$$\widehat{}\widehat{}$(*Suc n*)) *S U* **and** *U-T*: *cdcl*$_{NOT}$ *U T* **using** *S-T* **by** *auto*
then have μ *A U* $<$ μ *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 μ *A T* $<$ μ *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* $\wedge$ *cdcl*$_{NOT}$*-inv S* $\wedge$ *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 μ *A T* $\leq$ μ *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 $\neg(cdcl_{NOT}\ \widehat{}\widehat{}\ m)\ S\ T$
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: $(cdcl_{NOT}\widehat{}\widehat{}m)\ S\ T \Longrightarrow m \geq f\ n \Longrightarrow restart\ T\ U$
$\Longrightarrow cdcl_{NOT}\text{-}restart\ (S,\ n)\ (U,\ Suc\ n)$ |
restart-full: *full1* $cdcl_{NOT}$ $S\ T \Longrightarrow cdcl_{NOT}\text{-}restart\ (S,\ n)\ (T,\ Suc\ n)$

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* $\Longrightarrow$ $cdcl_{NOT}$*-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* +
dpll-state trail $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
for
trail :: *'st* ⇒ (*'v*, *unit*) *ann-lits* **and**
$clauses_{NOT}$ :: *'st* ⇒ *'v clauses* **and**
prepend-trail :: (*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒*'st* **and**
$add\text{-}cls_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
$remove\text{-}cls_{NOT}$ :: *'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* $T \Longrightarrow$ *bound-inv* A T
$\Longrightarrow$ $cdcl_{NOT}$*-restart* (T, n) (V, *Suc* n) $\Longrightarrow \mu\ A\ V \leq \mu$*-bound* A T **and**
$cdcl_{NOT}$*-raw-restart-*μ*-bound*:
$cdcl_{NOT}$*-restart* (T, a) (V, b) $\Longrightarrow$ $cdcl_{NOT}$*-inv* $T \Longrightarrow$ *bound-inv* A T

$\Longrightarrow$ *μ-bound A V* $\leq$ *μ-bound A T*
begin

lemma *rtranclp-cdcl*$_{NOT}$*-raw-restart-μ-bound*:
  cdcl$_{NOT}$*-restart*** *(T, a) (V, b)* $\Longrightarrow$ *cdcl*$_{NOT}$*-inv T* $\Longrightarrow$ *bound-inv A T*
    $\Longrightarrow$ *μ-bound A V* $\leq$ *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)* $\Longrightarrow$ *cdcl*$_{NOT}$*-inv T* $\Longrightarrow$ *bound-inv A T*
    $\Longrightarrow$ *μ A V* $\leq$ *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)* $\Longrightarrow$ *cdcl*$_{NOT}$*-inv T* $\Longrightarrow$ *bound-inv A T*
    $\Longrightarrow$ *μ A V* $\leq$ *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* $\wedge$ *cdcl*$_{NOT}$*-inv (fst S)*} (**is** *wf ?A*)
proof (*rule ccontr*)
  assume $\neg$ *?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 (λi. 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*$_{NOT}$ $\widehat{\ }\widehat{\ }$ *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*$_{NOT}$ $\widehat{\ }\widehat{\ }$ *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 $\mu\ A\ S - m' = 0$
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 $(\lambda S\ T.\ cdcl_{NOT}\ S\ T \wedge cdcl_{NOT}\text{-}inv\ S \wedge bound\text{-}inv\ A\ S)^{**}\ S\ T \longleftrightarrow cdcl_{NOT}{}^{**}\ S\ T$
(**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* $(\lambda S\ T.\ cdcl_{NOT}\ S\ T \wedge cdcl_{NOT}\text{-}inv\ S \wedge bound\text{-}inv\ A\ S)\ T\ U$
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

1.2.6 Merging backjump and learning

locale $cdcl_{NOT}$*-merge-bj-learn-ops* =
decide-ops trail $clauses_{NOT}$ *prepend-trail tl-trail add-*cls_{NOT} *remove-*cls_{NOT} +

forget-ops trail clauses$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$ *forget-cond* +
propagate-ops trail clauses$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$ *propagate-conds*
for
trail :: *'st* ⇒ (*'v*, *unit*) *ann-lits* **and**
clauses$_{NOT}$ :: *'st* ⇒ *'v clauses* **and**
prepend-trail :: (*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒*'st* **and**
add-cls$_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
remove-cls$_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
propagate-conds :: (*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *bool* **and**
forget-cond :: *'v clause* ⇒ *'st* ⇒ *bool* +
fixes *backjump-l-cond* :: *'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool*
begin

We have a new backjump that combines the backjumping on the trail and the learning of the used clause (called C'' below)

inductive *backjump-l* **where**
backjump-l: *trail S* = *F'* @ *Decided K* # *F*
⟹ *no-dup* (*trail S*)
⟹ *T* ∼ *prepend-trail* (*Propagated L* ()) (*reduce-trail-to*$_{NOT}$ *F* (*add-cls*$_{NOT}$ *C'' S*))
⟹ *C* ∈# *clauses*$_{NOT}$ *S*
⟹ *trail S* ⊨*as CNot C*
⟹ *undefined-lit F L*
⟹ *atm-of L* ∈ *atms-of-mm* (*clauses*$_{NOT}$ *S*) ∪ *atm-of* ‘ (*lits-of-l* (*trail S*))
⟹ *clauses*$_{NOT}$ *S* ⊨*pm C'* + {#*L*#}
⟹ *C''* = *C'* + {#*L*#}
⟹ *F* ⊨*as CNot C'*
⟹ *backjump-l-cond C C' L S T*
⟹ *backjump-l S T*

Avoid (meaningless) simplification in the theorem generated by *inductive-cases*:

declare *reduce-trail-to*$_{NOT}$*-length-ne*[*simp del*] *Set.Un-iff*[*simp del*] *Set.insert-iff*[*simp del*]
inductive-cases *backjump-lE*: *backjump-l S T*
thm *backjump-lE*
declare *reduce-trail-to*$_{NOT}$*-length-ne*[*simp*] *Set.Un-iff*[*simp*] *Set.insert-iff*[*simp*]

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*$_{NOT}$ *prepend-trail tl-trail add-cls*$_{NOT}$ *remove-cls*$_{NOT}$
propagate-conds forget-cond

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

abbreviation *backjump-conds* :: $'v$ *clause* $\Rightarrow\ 'v$ *clause* $\Rightarrow\ 'v$ *literal* $\Rightarrow\ 'st \Rightarrow\ 'st \Rightarrow$ *bool*
where
backjump-conds $\equiv \lambda C\ C'\ L'\ S\ T.$ *distinct-mset* $(C' + \{\#L'\#\}) \wedge \neg$*tautology* $(C' + \{\#L'\#\})$

Without additional knowledge on *backjump-l-cond*, it is impossible to have the same invariant.

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

by (*elim backjump-lE*) *metis*
interpret *backjumping-ops trail clauses$_{NOT}$ prepend-trail tl-trail add-cls$_{NOT}$ remove-cls$_{NOT}$ backjump-conds*
by *unfold-locales*
have $\exists$ *T. backjump S T*
apply *rule*
apply (*rule backjump.intros*)
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 tr-S* **apply** *simp*
using *cls-S-C'* **apply** *simp*
using *F-C'* **apply** *simp*
using *dist not-tauto cond* **apply** *simp*
done
}
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$_{NOT}$ prepend-trail tl-trail add-cls$_{NOT}$ remove-cls$_{NOT}$ propagate-conds forget-cond backjump-l-cond inv
for
trail :: *'st* ⇒ (*'v*, *unit*) *ann-lits* **and**
clauses$_{NOT}$ :: *'st* ⇒ *'v clauses* **and**
prepend-trail :: (*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒*'st* **and**
add-cls$_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
remove-cls$_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
propagate-conds :: (*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *bool* **and**
forget-cond :: *'v clause* ⇒ *'st* ⇒ *bool* **and**
backjump-l-cond :: *'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool* **and**
inv :: *'st* ⇒ *bool*
begin

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

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

backjump-l-cond :: *'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool* **and**
propagate-conds :: (*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *bool* **and**
forget-cond :: *'v clause* ⇒ *'st* ⇒ *bool* **and**
inv :: *'st* ⇒ *bool* +
assumes
dpll-merge-bj-inv: $\bigwedge S\ T$. *dpll-bj S T* ⟹ *inv S* ⟹ *inv T* **and**
learn-inv: $\bigwedge S\ T$. *learn S T* ⟹ *inv S* ⟹ *inv T*
begin

sublocale
conflict-driven-clause-learning trail $clauses_{NOT}$ *prepend-trail tl-trail* $add\text{-}cls_{NOT}$ $remove\text{-}cls_{NOT}$
inv backjump-conds propagate-conds
λC -. *distinct-mset C* ∧ ¬*tautology C*
forget-cond
apply *unfold-locales*
using $cdcl_{NOT}$*-merged-bj-learn-forget*$_{NOT}$ *cdcl-merged-inv learn-inv*
by (*auto simp add*: $cdcl_{NOT}$*.simps dpll-merge-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 ∃ *C' L D*. *learn S* ($add\text{-}cls_{NOT}$ *D S*)
∧ *D* = (*C'* + {#*L*#})
∧ *backjump* ($add\text{-}cls_{NOT}$ *D S*) *T*
∧ *atms-of* (*C'* + {#*L*#}) ⊆ *atms-of-mm* ($clauses_{NOT}$ *S*) ∪ *atm-of* ‘ (*lits-of-l* (*trail S*))
proof −
obtain *C F' K F L l C' D* **where**
tr-S: *trail S* = *F'* @ *Decided K* # *F* **and**
T: *T* ∼ *prepend-trail* (*Propagated L l*) ($reduce\text{-}trail\text{-}to_{NOT}$ *F* ($add\text{-}cls_{NOT}$ *D S*)) **and**
C-cls-S: *C* ∈# $clauses_{NOT}$ *S* **and**
tr-S-CNot-C: *trail S* ⊨*as CNot C* **and**
undef: *undefined-lit F L* **and**
atm-L: *atm-of L* ∈ *atms-of-mm* ($clauses_{NOT}$ *S*) ∪ *atm-of* ‘ (*lits-of-l* (*trail S*)) **and**
clss-C: $clauses_{NOT}$ *S* ⊨*pm D* **and**
D: *D* = *C'* + {#*L*#}
F ⊨*as CNot C'* **and**
distinct: *distinct-mset D* **and**
not-tauto: ¬ *tautology D*
using *bt inv* **by** (*elim backjump-lE*) *simp*
have *atms-C'*: *atms-of C'* ⊆ *atm-of* ‘ (*lits-of-l F*)
by (*metis D*(*2*) *atms-of-def image-subsetI true-annots-CNot-all-atms-defined*)
then have *atms-of* (*C'* + {#*L*#}) ⊆ *atms-of-mm* ($clauses_{NOT}$ *S*) ∪ *atm-of* ‘ (*lits-of-l* (*trail S*))
using *atm-L tr-S* **by** *auto*
moreover have *learn*: *learn S* ($add\text{-}cls_{NOT}$ *D S*)
apply (*rule learn.intros*)
apply (*rule clss-C*)
using *atms-C' atm-L D* **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* ($add\text{-}cls_{NOT}$ *D S*) *T*
apply (*rule backjump.intros*)
using ‹*F* ⊨*as CNot C'*› *C-cls-S tr-S-CNot-C undef T distinct not-tauto n-d D*
by (*auto simp*: *tr-S* $state\text{-}eq_{NOT}$*-def simp del*: $state\text{-}simp_{NOT}$)
ultimately show *?thesis* **using** *D* **by** *blast*

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 clause* **and** L :: *'v literal* **and** D :: *'v clause* **where**
f3: *learn* S (*add-cls*$_{NOT}$ D S) $\wedge$
backjump (*add-cls*$_{NOT}$ D S) T $\wedge$
atms-of (C' + {#L#}) $\subseteq$ *atms-of-mm* (*clauses*$_{NOT}$ S) $\cup$ *atm-of* ‘ *lits-of-l* (*trail* S) **and**
D: D = C' + {#L#}
using *n-d backjump-l-learn-backjump*[*OF bt inv*] **by** *blast*
then have *f4*: *cdcl*$_{NOT}$ S (*add-cls*$_{NOT}$ D S)
using *n-d c-learn* **by** *blast*
have *cdcl*$_{NOT}$ (*add-cls*$_{NOT}$ D 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*
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› *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 clause 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 clause 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*$_{NOT}$ *T*))

lemma *cdcl*$_{NOT}$*-decreasing-measure'*:
assumes
cdcl$_{NOT}$*-merged-bj-learn S T* **and**
inv: *inv S* **and**
atm-clss: *atms-of-mm* (*clauses*$_{NOT}$ *S*) $\subseteq$ *atms-of-ms A* **and**
atm-trail: *atm-of ‘ lits-of-l* (*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*$_{NOT}$ *S* $=$ *clauses*$_{NOT}$ *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*$_{NOT}$ *S* $=$ *clauses*$_{NOT}$ *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*$_{NOT}$ *T*)) $<$ *card* (*set-mset* (*clauses*$_{NOT}$ *S*))
using ‹*forget*$_{NOT}$ *S T*› **by** (*metis card-Diff1-less clauses-remove-cls*$_{NOT}$ *finite-set-mset*
forget$_{NOT}$*.cases linear set-mset-minus-replicate-mset*(*1*) *state-eq*$_{NOT}$*-def*)

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 $\mu_{CDCL}{}'$*-merged-def* $\mu_C{}'$*-def* **by** *simp*
next
  case ($cdcl_{NOT}$*-merged-bj-learn-backjump-l T*) **note** *bj-l* = *this*(*1*)
  obtain *C′ L D* **where**
    learn: *learn S* (*add-cls*$_{NOT}$ *D S*) **and**
    bj: *backjump* (*add-cls*$_{NOT}$ *D S*) *T* **and**
    atms-C: *atms-of* (*C′* + {#*L*#}) ⊆ *atms-of-mm* (*clauses*$_{NOT}$ *S*) ∪ *atm-of* ‘ (*lits-of-l* (*trail S*)) **and**
    D: *D* = *C′* + {#*L*#}
    using *bj-l inv backjump-l-learn-backjump* [*of S*] *n-d atm-clss atm-trail* **by** *blast*
  have *card-T-S*: *card* (*set-mset* (*clauses*$_{NOT}$ *T*)) ≤ *1*+ *card* (*set-mset* (*clauses*$_{NOT}$ *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*$_{NOT}$ *D S*)))
    apply (*rule dpll-bj-trail-mes-decreasing-prop*)
        using *bj bj-backjump* **apply** *blast*
      using $cdcl_{NOT}$*.c-learn* $cdcl_{NOT}$*-inv inv learn* **apply** *blast*
     using *atms-C atm-clss atm-trail D* **apply** (*simp add*: *n-d*) **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** $\mu_{CDCL}{}'$*-merged-def* $\mu_C{}'$*-def* **by** *linarith*
qed

lemma *wf*-$cdcl_{NOT}$*-merged-bj-learn*:
  assumes
    fin-A: *finite A*
  shows *wf* {(*T*, *S*).
    (*inv S* ∧ *atms-of-mm* (*clauses*$_{NOT}$ *S*) ⊆ *atms-of-ms A* ∧ *atm-of* ‘ *lits-of-l* (*trail S*) ⊆ *atms-of-ms A*
    ∧ *no-dup* (*trail S*))
    ∧ $cdcl_{NOT}$*-merged-bj-learn S T*}
  apply (*rule wfP-if-measure*[*of - -* $\mu_{CDCL}{}'$*-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-mm* (*clauses*$_{NOT}$ *S*) ⊆ *atms-of-ms A* **and**
atm-trail: *atm-of* ' *lits-of-l* (*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-mm* (*clauses*$_{NOT}$ *S*) ⊆ *atms-of-ms A* ∧ *atm-of* ' *lits-of-l* (*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-mm* (*clauses*$_{NOT}$ *T*) ⊆ *atms-of-ms A*
using *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-l* (*trail T*))⊆ *atms-of-ms A*
using *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 *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-mm* (*clauses*$_{NOT}$ *S*) ⊆ *atms-of-ms A* ∧ *atm-of* ' *lits-of-l* (*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 clause set* **and** *S T* :: *'st*
assumes

n-s: *no-step* $cdcl_{NOT}$*-merged-bj-learn S* **and**
atms-S: *atms-of-mm* ($clauses_{NOT}$ *S*) ⊆ *atms-of-ms A* **and**
atms-trail: *atm-of* ' *lits-of-l* (*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* ($clauses_{NOT}$ *S*) (*get-all-ann-decomposition* (*trail S*))
shows *unsatisfiable* (*set-mset* ($clauses_{NOT}$ *S*))
∨ (*trail S* ⊨*asm* $clauses_{NOT}$ *S* ∧ *satisfiable* (*set-mset* ($clauses_{NOT}$ *S*)))
proof −
let *?N* = *set-mset* ($clauses_{NOT}$ *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-l ?M* ∧ *atm-of P* ∉ *atm-of* ' *I*}
let *?O* = {*unmark L* |*L*. *is-decided 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-l ?M*)
using *tot atms-of-s-def* **unfolding** *total-over-m-def total-over-set-def*
by (*fastforce simp*: *image-iff*)
have {*P* |*P*. *P* ∈ *lits-of-l ?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*: *lits-of-def elim*!: *is-decided-ex-Decided*)

have *atms-N-M*: *atms-of-ms ?N* ⊆ *atm-of* ' *lits-of-l ?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-l ?M*
by *auto*
have *undefined-lit ?M* (*Pos l*)
using *l-M* **by** (*metis Decided-Propagated-in-iff-in-lits-of-l*
atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set literal.sel(*1*))
have $decide_{NOT}$ *S* (*prepend-trail* (*Decided* (*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 $?M \models as\ CNot\ C$
apply (*rule all-variables-defined-not-imply-cnot*)
using *atms-N-M* ‹$C \in ?N$› ‹$\neg\ ?M \models a\ C$› *atms-of-atms-of-ms-mono*[*OF* ‹$C \in ?N$›]
by (*auto dest*: *atms-of-atms-of-ms-mono*)
have $\exists\, l \in set\ ?M.\ is\text{-}decided\ l$
proof (*rule ccontr*)
let $?O = \{unmark\ L\ |L.\ is\text{-}decided\ L \wedge L \in set\ ?M \wedge atm\text{-}of\ (lit\text{-}of\ L) \notin atms\text{-}of\text{-}ms\ ?N\}$
have ϑ[*iff*]: $\bigwedge I.\ total\text{-}over\text{-}m\ I\ (?N \cup ?O \cup unmark\text{-}l\ ?M)$
$\longleftrightarrow total\text{-}over\text{-}m\ I\ (?N \cup unmark\text{-}l\ ?M)$
unfolding *total-over-set-def total-over-m-def atms-of-ms-def* **by** *blast*
assume $\neg\ ?thesis$
then have [*simp*]:$\{unmark\ L\ |L.\ is\text{-}decided\ L \wedge L \in set\ ?M\}$
$= \{unmark\ L\ |L.\ is\text{-}decided\ L \wedge L \in set\ ?M \wedge atm\text{-}of\ (lit\text{-}of\ L) \notin atms\text{-}of\text{-}ms\ ?N\}$
by *auto*
then have $?N \cup ?O \models ps\ unmark\text{-}l\ ?M$
using *all-decomposition-implies-propagated-lits-are-implied*[*OF decomp*] **by** *auto*

then have $?I \models s\ unmark\text{-}l\ ?M$
using *cons-I' I'-N tot-I'* ‹$?I \models s\ ?N \cup ?O$› **unfolding** ϑ *true-clss-clss-def* **by** *blast*
then have $lits\text{-}of\text{-}l\ ?M \subseteq ?I$
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*) *ann-lits* **where**
M-K: $?M = F'\ @\ Decided\ K\ \#\ F$ **and**
nm: $\forall f \in set\ F'.\ \neg is\text{-}decided\ f$
unfolding *is-decided-def* **by** (*metis* (*full-types*) *old.unit.exhaust*)
let $?K = Decided\ K$::(*'v, unit*) *ann-lit*
have $?K \in set\ ?M$
unfolding *M-K* **by** *auto*
let $?C = image\text{-}mset\ lit\text{-}of\ \{\#L \in \#mset\ ?M.\ is\text{-}decided\ L \wedge L \neq ?K\#\}$ :: *'v clause*
let $?C' = set\text{-}mset\ (image\text{-}mset\ (\lambda L::'v\ literal.\ \{\#L\#\})\ (?C + unmark\ ?K))$
have $?N \cup \{unmark\ L\ |L.\ is\text{-}decided\ L \wedge L \in set\ ?M\} \models ps\ unmark\text{-}l\ ?M$
using *all-decomposition-implies-propagated-lits-are-implied*[*OF decomp*] **.**
moreover have C': $?C' = \{unmark\ L\ |L.\ is\text{-}decided\ L \wedge L \in set\ ?M\}$
unfolding *M-K* **apply** *standard*
apply *force*
by *auto*
ultimately have *N-C-M*: $?N \cup ?C' \models ps\ unmark\text{-}l\ ?M$
by *auto*
have *N-M-False*: $?N \cup (\lambda L.\ unmark\ L)$ ‘ $(set\ ?M) \models ps\ \{\{\#\}\}$
using *M* ‹$?M \models as\ CNot\ C$› ‹$C \in ?N$› **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-l ?M = ?N ∪ unmark-l ?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 {a ∈ set (trail S). is-decided a ∧ a ≠ Decided K} =
       set (trail S) ∩ {L. is-decided L ∧ L ≠ Decided K}
       by auto
      then have tot': total-over-set I
         (atm-of ` lit-of ` (set ?M ∩ {L. is-decided L ∧ L ≠ Decided K}))
        using tot by (auto simp add: atms-of-uminus-lit-atm-of-lit-of)
      { fix x :: ('v, unit) ann-lit
        assume
          a3: lit-of x ∉ I and
          a1: x ∈ set ?M and
          a4: is-decided x and
          a5: x ≠ Decided 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-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-decided L ∧ L ≠ Decided 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 clause set* **and** *S T* :: *'st*
assumes
full: *full cdcl$_{NOT}$-merged-bj-learn S T* **and**
atms-S: *atms-of-mm* (*clauses$_{NOT}$ S*) $\subseteq$ *atms-of-ms A* **and**
atms-trail: *atm-of ‘ lits-of-l* (*trail S*) $\subseteq$ *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$_{NOT}$ S*) (*get-all-ann-decomposition* (*trail S*))
shows *unsatisfiable* (*set-mset* (*clauses$_{NOT}$ T*))
$\vee$ (*trail T* $\models$*asm clauses$_{NOT}$ T* $\wedge$ *satisfiable* (*set-mset* (*clauses$_{NOT}$ 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-mm* (*clauses$_{NOT}$ T*) $\subseteq$ *atms-of-ms A* **and** *atm-of ‘ lits-of-l* (*trail T*) $\subseteq$ *atms-of-ms A*
using *rtranclp-cdcl$_{NOT}$-trail-clauses-bound*[*OF st inv n-d atms-S atms-trail*] **by** *blast+*
moreover have *no-dup* (*trail T*)
using *rtranclp-cdcl$_{NOT}$-no-dup inv n-d st* **by** *blast*
moreover have *inv T*
using *rtranclp-cdcl$_{NOT}$-inv inv st* **by** *blast*
moreover have *all-decomposition-implies-m* (*clauses$_{NOT}$ T*) (*get-all-ann-decomposition* (*trail T*))
using *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

1.2.7 Instantiations

In this section, we instantiate the previous locales to ensure that the assumption are not contradictory.

locale *cdcl$_{NOT}$-with-backtrack-and-restarts* =
conflict-driven-clause-learning-learning-before-backjump-only-distinct-learnt
trail clauses$_{NOT}$ prepend-trail tl-trail add-cls$_{NOT}$ remove-cls$_{NOT}$
inv backjump-conds propagate-conds learn-restrictions forget-restrictions
for
trail :: *'st* ⇒ (*'v*, *unit*) *ann-lits* **and**
clauses$_{NOT}$ :: *'st* ⇒ *'v clauses* **and**
prepend-trail :: (*'v*, *unit*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒*'st* **and**
add-cls$_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
remove-cls$_{NOT}$ :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
inv :: *'st* ⇒ *bool* **and**
backjump-conds :: *'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool* **and**
propagate-conds :: (*'v*, *unit*) *ann-lit* ⇒ *'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: $\bigwedge S\ T.\ inv\ S \Longrightarrow T \sim reduce\text{-}trail\text{-}to_{NOT}\ ([]::'a\ list)\ S \Longrightarrow inv\ T$
begin

lemma *bound-inv-inv*:
assumes
inv S **and**
n-d: *no-dup* (*trail S*) **and**
atms-clss-S-A: *atms-of-mm* ($clauses_{NOT}$ *S*) ⊆ *atms-of-ms A* **and**
atms-trail-S-A: *atm-of* ' *lits-of-l* (*trail S*) ⊆ *atms-of-ms A* **and**
finite A **and**
$cdcl_{NOT}$: $cdcl_{NOT}$ *S T*
shows
atms-of-mm ($clauses_{NOT}$ *T*) ⊆ *atms-of-ms A* **and**
atm-of ' *lits-of-l* (*trail T*) ⊆ *atms-of-ms A* **and**
finite A
proof −
have $cdcl_{NOT}$ *S T*
using ⟨*inv S*⟩ $cdcl_{NOT}$ **by** *linarith*
then have *atms-of-mm* ($clauses_{NOT}$ *T*) ⊆ *atms-of-mm* ($clauses_{NOT}$ *S*) ∪ *atm-of* ' *lits-of-l* (*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-mm* ($clauses_{NOT}$ *T*) ⊆ *atms-of-ms A*
using *atms-clss-S-A atms-trail-S-A* **by** *blast*
next
show *atm-of* ' *lits-of-l* (*trail T*) ⊆ *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* $\lambda S\ T.\ T \sim reduce\text{-}trail\text{-}to_{NOT}\ ([]::'a\ list)\ S$ $cdcl_{NOT}$ *f*
$\lambda A\ S.$ *atms-of-mm* ($clauses_{NOT}$ *S*) ⊆ *atms-of-ms A* ∧ *atm-of* ' *lits-of-l* (*trail S*) ⊆ *atms-of-ms A* ∧
finite A
μ_{CDCL}' $\lambda S.$ *inv S* ∧ *no-dup* (*trail S*)
μ_{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-*$cdcl_{NOT}$*-*μ_{CDCL}'*-bound*; *simp*)
apply (*rule rtranclp-*μ_{CDCL}'*-bound-decreasing*; *simp*)
apply *auto*[]
apply *auto*[]
using $cdcl_{NOT}$*-inv* $cdcl_{NOT}$*-no-dup* **apply** *blast*
using *inv-restart* **apply** *auto*[]
done

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-mm ($clauses_{NOT}$ *T*) ⊆ *atms-of-ms A*
atm-of ' *lits-of-l* (*trail T*) ⊆ *atms-of-ms A*
finite A
shows $\mu_{CDCL}{}'$ *A V* ≤ $\mu_{CDCL}{}'$*-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-*$cdcl_{NOT}$*-*$\mu_{CDCL}{}'$*-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-*$cdcl_{NOT}$*-*$\mu_{CDCL}{}'$*-bound*)
using *full 2* **unfolding** *full1-def* **by** *force+*
qed

lemma $cdcl_{NOT}$*-with-restart-*$\mu_{CDCL}{}'$*-bound-le-*$\mu_{CDCL}{}'$*-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-mm ($clauses_{NOT}$ *T*) ⊆ *atms-of-ms A*
atm-of ' *lits-of-l* (*trail T*) ⊆ *atms-of-ms A*
finite A
shows $\mu_{CDCL}{}'$*-bound A V* ≤ $\mu_{CDCL}{}'$*-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 $\mu_{CDCL}{}'$*-bound A T* ≤ $\mu_{CDCL}{}'$*-bound A S*
apply (*rule rtranclp-*$\mu_{CDCL}{}'$*-bound-decreasing*)
using ⟨($cdcl_{NOT}$ ^^ *m*) *S T*⟩ **apply** (*fastforce dest*: *relpowp-imp-rtranclp*)
using *1* **by** *auto*
then show *?case* **using** *U* **unfolding** $\mu_{CDCL}{}'$*-bound-def* **by** *auto*
next
case (*2 S T n*) **note** *full* = *this*(*2*)
show *?case*
apply (*rule rtranclp-*$\mu_{CDCL}{}'$*-bound-decreasing*)
using *full 2* **unfolding** *full1-def* **by** *force+*
qed

sublocale $cdcl_{NOT}$*-increasing-restarts* - - - - - -
f
λ*S T*. *T* ∼ *reduce-trail-to*$_{NOT}$ ([]::'*a list*) *S*
λ*A S*. *atms-of-mm* ($clauses_{NOT}$ *S*) ⊆ *atms-of-ms A*
∧ *atm-of* ' *lits-of-l* (*trail S*) ⊆ *atms-of-ms A* ∧ *finite A*
$\mu_{CDCL}{}'$ $cdcl_{NOT}$
λ*S*. *inv S* ∧ *no-dup* (*trail S*)
$\mu_{CDCL}{}'$*-bound*
apply *unfold-locales*

using $cdcl_{NOT}$*-with-restart-*μ_{CDCL}*'-le-*μ_{CDCL}*'-bound* **apply** *simp*
using $cdcl_{NOT}$*-with-restart-*μ_{CDCL}*'-bound-le-*μ_{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 ($clauses_{NOT}$ (*fst* S)) (*get-all-ann-decomposition* (*trail* (*fst* S)))
shows
all-decomposition-implies-m ($clauses_{NOT}$ (*fst* T)) (*get-all-ann-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 ($clauses_{NOT}$ (*fst* S)) (*get-all-ann-decomposition* (*trail* (*fst* S)))
shows
all-decomposition-implies-m ($clauses_{NOT}$ (*fst* T)) (*get-all-ann-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* $clauses_{NOT}$ (*fst* S) $\longleftrightarrow$ I $\models$*sextm* $clauses_{NOT}$ (*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*:
fixes S T :: *'st* $\times$ *nat*

assumes
st: $cdcl_{NOT}$*-restart*** *S T* **and**
n-d: *no-dup* (*trail* (*fst S*)) **and**
inv: *inv* (*fst S*)
shows $I \models$*sextm* $clauses_{NOT}$ (*fst S*) $\longleftrightarrow$ $I \models$*sextm* $clauses_{NOT}$ (*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 clause set* **and** *S T* :: *'st*
assumes
full: *full* $cdcl_{NOT}$*-restart* (*S*, *n*) (*T*, *m*) **and**
atms-S: *atms-of-mm* ($clauses_{NOT}$ *S*) $\subseteq$ *atms-of-ms A* **and**
atms-trail: *atm-of* ‘ *lits-of-l* (*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* ($clauses_{NOT}$ *S*) (*get-all-ann-decomposition* (*trail S*))
shows *unsatisfiable* (*set-mset* ($clauses_{NOT}$ *S*))
$\lor$ (*lits-of-l* (*trail T*) $\models$*sextm* $clauses_{NOT}$ *S* $\land$ *satisfiable* (*set-mset* ($clauses_{NOT}$ *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-mm* ($clauses_{NOT}$ *T*) $\subseteq$ *atms-of-ms A*
atm-of ‘ *lits-of-l* (*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* ($clauses_{NOT}$ *T*) (*get-all-ann-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* ($clauses_{NOT}$ *T*))
$\lor$ (*trail T* $\models$*asm* $clauses_{NOT}$ *T* $\land$ *satisfiable* (*set-mset* ($clauses_{NOT}$ *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* $clauses_{NOT}$ *S* $\longleftrightarrow$ $I \models$*sextm* $clauses_{NOT}$ *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-l* (*trail T*))
using *inv-T*(*1*) *distinct-consistent-interp* **by** *blast*
consider
(*unsat*) *unsatisfiable* (*set-mset* ($clauses_{NOT}$ *T*))
| (*sat*) *trail T* $\models$*asm* $clauses_{NOT}$ *T* **and** *satisfiable* (*set-mset* ($clauses_{NOT}$ *T*))

using T **by** *blast*
then show *?thesis*
proof *cases*
case *unsat*
then have *unsatisfiable* (*set-mset* ($clauses_{NOT}$ 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-l* (*trail* T) $\models$*sextm* $clauses_{NOT}$ 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* ($clauses_{NOT}$ 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

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

locale *cdcl*$_{NOT}$*-merge-bj-learn-with-backtrack-restarts* =
cdcl$_{NOT}$*-merge-bj-learn trail* $clauses_{NOT}$ *prepend-trail tl-trail* *add-cls*$_{NOT}$ *remove-cls*$_{NOT}$
$\lambda C\ C'\ L'\ S\ T.$ *distinct-mset* (C' + {#L'#}) $\wedge$ *backjump-l-cond* $C\ C'\ L'\ S\ T$
propagate-conds forget-conds inv
for
trail :: $'st \Rightarrow ('v,\ unit)$ *ann-lits* **and**
$clauses_{NOT}$ :: $'st \Rightarrow\ 'v$ *clauses* **and**
prepend-trail :: $('v,\ unit)$ *ann-lit* $\Rightarrow\ 'st \Rightarrow\ 'st$ **and**
tl-trail :: $'st \Rightarrow 'st$ **and**
add-cls$_{NOT}$ :: $'v$ *clause* $\Rightarrow\ 'st \Rightarrow\ 'st$ **and**
remove-cls$_{NOT}$ :: $'v$ *clause* $\Rightarrow\ 'st \Rightarrow\ 'st$ **and**
propagate-conds :: $('v,\ unit)$ *ann-lit* $\Rightarrow\ 'st \Rightarrow$ *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* $\Rightarrow\ 'st \Rightarrow\ 'st \Rightarrow$ *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

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-mm* ($clauses_{NOT}$ S) $\subseteq$ *atms-of-ms* A **and**
x $\in$# $clauses_{NOT}$ S **and** *finite* A
shows $x \in$ *simple-clss* (*atms-of-ms* A) $\vee$ x $\in$# *not-simplified-cls* ($clauses_{NOT}$ 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 subsetCE*)
then show *?thesis* **by** *blast*
next
case *n-simp*
then have $x \in\#$ *not-simplified-cls* ($clauses_{NOT}$ *S*)
using ‹$x \in\#$ $clauses_{NOT}$ *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-mm* ($clauses_{NOT}$ *S*) $\subseteq$ *atms-of-ms A* **and**
atms-trail: *atm-of* ‘(*lits-of-l* (*trail S*)) $\subseteq$ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
fin-A[*simp*]: *finite A*
shows *set-mset* ($clauses_{NOT}$ *T*) $\subseteq$ *set-mset* (*not-simplified-cls* ($clauses_{NOT}$ *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 *bj inv* $cdcl_{NOT}$*-merged-bj-learn.simps n-d* **by** *blast+*
have *atm-of* ‘(*lits-of-l* (*trail T*)) $\subseteq$ *atms-of-ms A*
using *rtranclp-*$cdcl_{NOT}$*-trail-clauses-bound*[*OF* ‹$cdcl_{NOT}{}^{**}$ *S T*›] *inv atms-trail atms-clss n-d* **by** *auto*
have *atms-of-mm* ($clauses_{NOT}$ *T*) $\subseteq$ *atms-of-ms A*
using *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 *rtranclp-*$cdcl_{NOT}$*-no-dup*[*OF* ‹$cdcl_{NOT}{}^{**}$ *S T*› *inv n-d*] **by** *fast*

obtain *F' K F L l C' C D* **where**
tr-S: *trail S* = *F'* @ *Decided K* # *F* **and**
T: *T* $\sim$ *prepend-trail* (*Propagated L l*) (*reduce-trail-to*$_{NOT}$ *F* (*add-cls*$_{NOT}$ *D S*)) **and**
C $\in\#$ $clauses_{NOT}$ *S* **and**

trail S $\models$*as CNot C* **and**
undef: *undefined-lit F L* **and**
$clauses_{NOT}$ *S* $\models$*pm* $C' + \{\#L\#\}$ **and**
F $\models$*as CNot* C' **and**
D: $D = C' + \{\#L\#\}$ **and**
dist: *distinct-mset* $(C' + \{\#L\#\})$ **and**
tauto: $\neg$ *tautology* $(C' + \{\#L\#\})$ **and**
backjump-l-cond C C' L S T
using ‹*backjump-l S T*› **apply** (*elim backjump-lE*) **by** *auto*

have *atms-of* $C' \subseteq$ *atm-of* ‘ (*lits-of-l F*)
using ‹*F* $\models$*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* $(C'+\{\#L\#\}) \subseteq$ *atms-of-ms A*
using *T* ‹*atm-of* ‘ *lits-of-l* (*trail T*) $\subseteq$ *atms-of-ms A*› *tr-S undef n-d* **by** *auto*
then have *simple-clss* (*atms-of* $(C' + \{\#L\#\})$) $\subseteq$ *simple-clss* (*atms-of-ms A*)
apply − **by** (*rule simple-clss-mono*) (*simp-all*)
then have $C' + \{\#L\#\} \in$ *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 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* ($clauses_{NOT}$ *T*)) $\subseteq\#$ (*not-simplified-cls* ($clauses_{NOT}$ *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* ($clauses_{NOT}$ *T*)) $\subseteq\#$ (*not-simplified-cls* ($clauses_{NOT}$ *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-mm ($clauses_{NOT}$ *S*) $\subseteq$ *atms-of-ms A* **and**
atm-of ‘(*lits-of-l* (*trail S*)) $\subseteq$ *atms-of-ms A* **and**
n-d: *no-dup* (*trail S*) **and**
finite[*simp*]: *finite A*
shows *set-mset* ($clauses_{NOT}$ *T*) $\subseteq$ *set-mset* (*not-simplified-cls* ($clauses_{NOT}$ *S*))
$\cup$ *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-mm* ($clauses_{NOT}$ *T*) ⊆ *atms-of-ms A* **and**
atm-of ' *lits-of-l* (*trail T*) ⊆ *atms-of-ms A*
using *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 *rtranclp-*$cdcl_{NOT}$*-no-dup*[*OF* ‹$cdcl_{NOT}^{**}$ *S T*› *inv n-d*] **by** *fast*
ultimately have *set-mset* ($clauses_{NOT}$ *U*)
⊆ *set-mset* (*not-simplified-cls* ($clauses_{NOT}$ *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* ($clauses_{NOT}$ *T*))
⊆ *set-mset* (*not-simplified-cls* ($clauses_{NOT}$ *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*($clauses_{NOT}$ *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-mm ($clauses_{NOT}$ *S*) ⊆ *atms-of-ms A* **and**
atm-of '(*lits-of-l* (*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* ($clauses_{NOT}$ *T*) ⊆ *set-mset* (*not-simplified-cls*($clauses_{NOT}$ *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*($clauses_{NOT}$ *S*))
∪ *simple-clss* (*atms-of-ms A*))
≤ *card* (*set-mset* (*not-simplified-cls*($clauses_{NOT}$ *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* ($clauses_{NOT}$ *T*))
≤ *card* (*set-mset* (*not-simplified-cls*($clauses_{NOT}$ *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*
≤ (*2* + *card* (*atms-of-ms A*)) ^ (*1* + *card* (*atms-of-ms A*)) ∗ *2*
by *auto*
ultimately show *?thesis* **unfolding** $\mu_{CDCL}{}'$*-merged-def* **by** *auto*
qed

sublocale $cdcl_{NOT}$*-increasing-restarts-ops* λ*S T*. *T* ∼ *reduce-trail-to*$_{NOT}$ ([]::'*a list*) *S*

cdcl$_{NOT}$*-merged-bj-learn f*
λA S. *atms-of-mm* (*clauses*$_{NOT}$ S) $\subseteq$ *atms-of-ms* A
$\wedge$ *atm-of* ‘ *lits-of-l* (*trail* S) $\subseteq$ *atms-of-ms* A $\wedge$ *finite* A
μ_{CDCL}′*-merged*
λS. *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* *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 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-mm (*clauses*$_{NOT}$ (*fst* T)) $\subseteq$ *atms-of-ms* A **and**
atm-of ‘ *lits-of-l* (*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-mm* (*clauses*$_{NOT}$ T) $\subseteq$ *atms-of-ms* A
using *rtranclp-cdcl*$_{NOT}$*-trail-clauses-bound*[*OF st*′′] *inv atms-clss atms-trail n-d*
by *simp*
then have *atms-of-mm* (*clauses*$_{NOT}$ U) $\subseteq$ *atms-of-ms* A
using U **by** *simp*
have *not-simplified-cls* (*clauses*$_{NOT}$ U) $\subseteq\#$ *not-simplified-cls* (*clauses*$_{NOT}$ T)
using ⟨U $\sim$ *reduce-trail-to*$_{NOT}$ [] T⟩ **by** *auto*

moreover have *not-simplified-cls* (*clauses*$_{NOT}$ *T*) ⊆# *not-simplified-cls* (*clauses*$_{NOT}$ *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*$_{NOT}$ *U*) ⊆# *not-simplified-cls* (*clauses*$_{NOT}$ *S*)
by *auto*

have (*set-mset* (*clauses*$_{NOT}$ *U*))
⊆ *set-mset* (*not-simplified-cls* (*clauses*$_{NOT}$ *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-mm* (*clauses*$_{NOT}$ *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*$_{NOT}$ *U*)) ≤ *card* (*set-mset* (*not-simplified-cls* (*clauses*$_{NOT}$ *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*$_{NOT}$ *U*)) ∪ *simple-clss* (*atms-of-ms A*)
⊆ *set-mset* (*not-simplified-cls* (*clauses*$_{NOT}$ *S*)) ∪ *simple-clss* (*atms-of-ms A*)
using *U-S* **by** *auto*
then have *f2*:
card (*set-mset* (*not-simplified-cls* (*clauses*$_{NOT}$ *U*)) ∪ *simple-clss* (*atms-of-ms A*))
≤ *card* (*set-mset* (*not-simplified-cls* (*clauses*$_{NOT}$ *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*$_{NOT}$ *S*))
∪ *simple-clss* (*atms-of-ms A*))
≤ *card* (*set-mset* (*not-simplified-cls* (*clauses*$_{NOT}$ *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*$_{NOT}$ *U*))
≤ *card* (*set-mset* (*not-simplified-cls* (*clauses*$_{NOT}$ *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*$_{NOT}$ *T*) ⊆# *not-simplified-cls* (*clauses*$_{NOT}$ *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* ($clauses_{NOT}$ *U*) ⊆# *not-simplified-cls* ($clauses_{NOT}$ *T*)
using ‹*U* ∼ *reduce-trail-to*$_{NOT}$ [] *T*› **by** *auto*
moreover have *not-simplified-cls* ($clauses_{NOT}$ *T*) ⊆# *not-simplified-cls* ($clauses_{NOT}$ *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* ($clauses_{NOT}$ *U*) ⊆# *not-simplified-cls* ($clauses_{NOT}$ *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-mm* ($clauses_{NOT}$ *S*) ⊆ *atms-of-ms A*
∧ *atm-of* ‘ *lits-of-l* (*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*($clauses_{NOT}$ *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* $clauses_{NOT}$ (*fst S*) ⟷ *I* ⊨*sextm* $clauses_{NOT}$ (*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 *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* ⊨*sextm* $clauses_{NOT}$ *S* ⟷ *I* ⊨*sextm* $clauses_{NOT}$ *T*
using *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 $I \models sextm\ clauses_{NOT}\ T \longleftrightarrow I \models sextm\ clauses_{NOT}\ U$
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 $I \models sextm\ clauses_{NOT}\ (fst\ S) \longleftrightarrow I \models sextm\ clauses_{NOT}\ (fst\ T)$
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 $I \models sextm\ clauses_{NOT}\ (fst\ T) \longleftrightarrow I \models sextm\ clauses_{NOT}\ (fst\ U)$
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*$_{NOT}$ (*fst S*))
(*get-all-ann-decomposition* (*trail* (*fst S*)))
shows *all-decomposition-implies-m* (*clauses*$_{NOT}$ (*fst T*))
(*get-all-ann-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 *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*$_{NOT}$ (*fst S*))
(*get-all-ann-decomposition* (*trail* (*fst S*)))

shows *all-decomposition-implies-m* (*clauses*$_{NOT}$ (*fst T*))
(*get-all-ann-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*$_{NOT}$ (*fst S*))
(*get-all-ann-decomposition* (*trail* (*fst S*))) **and**
atms-cls: *atms-of-mm* (*clauses*$_{NOT}$ (*fst S*)) ⊆ *atms-of-ms A* **and**
atms-trail: *atm-of* ‘ *lits-of-l* (*trail* (*fst S*)) ⊆ *atms-of-ms A* **and**
fin: *finite A*
shows *unsatisfiable* (*set-mset* (*clauses*$_{NOT}$ (*fst S*)))
∨ *lits-of-l* (*trail* (*fst T*)) ⊨*sextm clauses*$_{NOT}$ (*fst S*) ∧ *satisfiable* (*set-mset* (*clauses*$_{NOT}$ (*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-mm* (*clauses*$_{NOT}$ (*fst T*)) ⊆ *atms-of-ms A* **and**
atms-trail-T: *atm-of* ‘ *lits-of-l* (*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*$_{NOT}$ (*fst T*))
(*get-all-ann-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*$_{NOT}$ (*fst T*)))
∨ *trail* (*fst T*) ⊨*asm clauses*$_{NOT}$ (*fst T*) ∧ *satisfiable* (*set-mset* (*clauses*$_{NOT}$ (*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*$_{NOT}$ (*fst T*)))
| (*sat*) *trail* (*fst T*) ⊨*asm clauses*$_{NOT}$ (*fst T*) **and** *satisfiable* (*set-mset* (*clauses*$_{NOT}$ (*fst T*)))
by *auto*
then show *unsatisfiable* (*set-mset* (*clauses*$_{NOT}$ (*fst S*)))
∨ *lits-of-l* (*trail* (*fst T*)) ⊨*sextm clauses*$_{NOT}$ (*fst S*) ∧ *satisfiable* (*set-mset* (*clauses*$_{NOT}$ (*fst S*)))

proof *cases*
case *unsat*
then have *unsatisfiable* (*set-mset* ($clauses_{NOT}$ (*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-l* (*trail* (*fst T*)) $\models$*sextm* $clauses_{NOT}$ (*fst T*)
using *true-clss-imp-true-cls-ext* **by** (*auto simp*: *true-annots-true-cls*)
then have *lits-of-l* (*trail* (*fst T*)) $\models$*sextm* $clauses_{NOT}$ (*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* ($clauses_{NOT}$ (*fst S*)))
using *consistent-true-clss-ext-satisfiable distinct-consistent-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* = [] $clauses_{NOT}$ *S* = *N* **and**
full: *full cdcl*$_{NOT}$*-restart* (*S*, *0*) *T* **and**
inv: *inv S*
shows *unsatisfiable* (*set-mset N*)
∨ *lits-of-l* (*trail* (*fst T*)) $\models$*sextm N* ∧ *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

1.3 DPLL as an instance of NOT

1.3.1 DPLL with simple backtrack

We are using a concrete couple instead of an abstract state.

locale *dpll-with-backtrack*
begin
inductive *backtrack* :: (*'v*, *unit*) *ann-lits* × *'v clauses*
⇒ (*'v*, *unit*) *ann-lits* × *'v clauses* ⇒ *bool* **where**
backtrack-split (*fst S*) = (*M'*, *L* # *M*) ⟹ *is-decided L* ⟹ *D* ∈# *snd S*
⟹ *fst S* $\models$*as CNot D* ⟹ *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*) *ann-lits*
assumes
backtrack: *backtrack* (*M*, *N*) (*M'*, *N'*) **and**
no-dup: (*no-dup* ∘ *fst*) (*M*, *N*) **and**
decomp: *all-decomposition-implies-m N* (*get-all-ann-decomposition M*)

shows
  $\exists\, C\ F'\ K\ F\ L\ l\ C'.$
    $M = F' @ \mathit{Decided}\ K \# F \wedge$
    $M' = \mathit{Propagated}\ L\ l \# F \wedge N = N' \wedge C \in\# N \wedge F' @ \mathit{Decided}\ K \# F \models\mathit{as}\ \mathit{CNot}\ C \wedge$
    $\mathit{undefined\text{-}lit}\ F\ L \wedge \mathit{atm\text{-}of}\ L \in \mathit{atms\text{-}of\text{-}mm}\ N \cup \mathit{atm\text{-}of}\ `\ \mathit{lits\text{-}of\text{-}l}\ (F' @ \mathit{Decided}\ K \# F) \wedge$
    $N \models\mathit{pm}\ C' + \{\#L\#\} \wedge F \models\mathit{as}\ \mathit{CNot}\ C'$

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-decided L **and**
    D ∈# *snd ?S* **and**
    M ⊨*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*∈#*mset M*. *is-decided K* ∧ *K*≠*L*#} :: ′*v clause*
  let *?C′* = *set-mset* (*image-mset single* (*?C*+{#*?K*#}))
  obtain *K* **where** *L*: *L* = *Decided K* **using** ‹*is-decided L*› **by** (*cases L*) *auto*

  have *M*: *M* = *F′* @ *Decided K* # *F*
    using *b-sp* **by** (*metis L backtrack-split-list-eq fst-conv snd-conv*)
  moreover have *F′* @ *Decided K* # *F* ⊨*as CNot D*
    using ‹*M*⊨*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*) ∈ *atms-of-mm N* ∪ *atm-of* ‘ *lits-of-l* (*F′* @ *Decided K* # *F*)
    by *auto*
  moreover
    have *set-mset N* ∪ *?C′* ⊨*ps* {{#}}
      proof −
        have *A*: *set-mset N* ∪ *?C′* ∪ *unmark-l M* =
          set-mset N ∪ *unmark-l M*
          unfolding *M L* **by** *auto*
        have *set-mset N* ∪ {{#*lit-of L*#} |*L*. *is-decided L* ∧ *L* ∈ *set M*}
            ⊨*ps unmark-l M*
          using *all-decomposition-implies-propagated-lits-are-implied*[*OF decomp*] **.**
        moreover have *C′*: *?C′* = {{#*lit-of L*#} |*L*. *is-decided L* ∧ *L* ∈ *set M*}
          unfolding *M L* **apply** *standard*
            apply *force*
          using *IntI* **by** *auto*
        ultimately have *N-C-M*: *set-mset N* ∪ *?C′* ⊨*ps unmark-l M*
          by *auto*
        have *set-mset N* ∪ (λ*L*. {#*lit-of L*#}) ‘ (*set M*) ⊨*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* ⊨ *D*
              using *I-N-M* ‹*D* ∈# *snd ?S*› **unfolding** *true-clss-def* **by** *auto*
            moreover have *I* ⊨*s CNot D*
              using ‹*M* ⊨*as CNot D*› **unfolding** *M* **by** (*metis 1*(*3*) ‹*M* ⊨*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* ∪ {*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 {*a* ∈ *set M*. *is-decided a* ∧ *a* ≠ *Decided K*} =
set M ∩ {*L*. *is-decided L* ∧ *L* ≠ *Decided K*}
by *auto*
then have
tI: *total-over-set I* (*atm-of* ‘ *lit-of* ‘ (*set M* ∩ {*L*. *is-decided L* ∧ *L* ≠ *Decided K*}))
using *tot* **by** (*auto simp add*: *L atms-of-uminus-lit-atm-of-lit-of*)

then have *H*: $\bigwedge x$.
lit-of x ∉ *I* ⟹ *x* ∈ *set M* ⟹*is-decided x*
⟹ *x* ≠ *Decided K* ⟹ −*lit-of x* ∈ *I*
proof −
fix *x* :: (*'v*, *unit*) *ann-lit*
assume *a1*: *x* ≠ *Decided K*
assume *a2*: *is-decided x*
assume *a3*: *x* ∈ *set M*
assume *a4*: *lit-of x* ∉ *I*
have *atm-of* (*lit-of x*) ∈ *atm-of* ‘ *lit-of* ‘
(*set M* ∩ {*m*. *is-decided m* ∧ *m* ≠ *Decided K*})
using *a3 a2 a1* **by** *blast*
then have *Pos* (*atm-of* (*lit-of x*)) ∈ *I* ∨ *Neg* (*atm-of* (*lit-of x*)) ∈ *I*
using *tI* **unfolding** *total-over-set-def* **by** *blast*
then show − *lit-of x* ∈ *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 ¬$I \models s$ *?C'*
using ‹*set-mset N* ∪ *?C'* ⊨*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*
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'* ∩ {*K*. *is-decided K* ∧ *K* ≠ *L*} = {}
using *backtrack-split-fst-not-decided*[*of - M*] *b-sp* **by** *auto*
then have *F* ⊨*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* ∈# *snd ?S*› *L* **by** *force*

qed

lemma *backtrack-is-backjump'*:
fixes $M\ M'$ :: *('v, unit) ann-lits*
assumes
backtrack: *backtrack S T* **and**
no-dup: *(no-dup* ∘ *fst) S* **and**
decomp: *all-decomposition-implies-m (snd S) (get-all-ann-decomposition (fst S))*
shows
∃ *C F' K F L l C'*.
fst S = F' @ Decided K # F ∧
T = (Propagated L l # F, snd S) ∧ *C* ∈# *snd S* ∧ *fst S* ⊨*as CNot C*
∧ *undefined-lit F L* ∧ *atm-of L* ∈ *atms-of-mm (snd S)* ∪ *atm-of ‘ lits-of-l (fst S)* ∧
snd S ⊨*pm C' +* {#*L*#} ∧ *F* ⊨*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 $\lambda L\ (M, N).\ (L \# M, N)\ \lambda(M, N).\ (tl\ M, N)$
$\lambda C\ (M, N).\ (M, \{\#C\#\} + N)\ \lambda C\ (M, N).\ (M, removeAll\text{-}mset\ C\ N)$
by *unfold-locales (auto simp: ac-simps)*

sublocale *backjumping-ops*
fst snd $\lambda L\ (M, N).\ (L \# M, N)\ \lambda(M, N).\ (tl\ M, N)$
$\lambda C\ (M, N).\ (M, \{\#C\#\} + N)\ \lambda C\ (M, N).\ (M, removeAll\text{-}mset\ C\ N)\ \lambda\text{- - - } S\ T.\ backtrack\ S\ T$
by *unfold-locales*
thm *reduce-trail-to*$_{NOT}$*-clauses*

lemma *reduce-trail-to*$_{NOT}$:
reduce-trail-to$_{NOT}$ *F S =*
(if length (fst S) ≥ *length F*
then drop (length (fst S) − length F) (fst S)
else [],
snd S) (**is** *?R = ?C*)
proof −
have *?R = (fst ?R, snd ?R)*
by *(cases reduce-trail-to*$_{NOT}$ *F S) auto*
also have *(fst ?R, snd ?R) = ?C*
by *(auto simp: trail-reduce-trail-to*$_{NOT}$*-drop)*
finally show *?thesis* **.**
qed

lemma *backtrack-is-backjump''*:
fixes $M\ M'$ :: *('v, unit) ann-lits*
assumes
backtrack: *backtrack S T* **and**
no-dup: *(no-dup* ∘ *fst) S* **and**
decomp: *all-decomposition-implies-m (snd S) (get-all-ann-decomposition (fst S))*
shows *backjump S T*
proof −
obtain *C F' K F L l C'* **where**
1: *fst S = F' @ Decided K # F* **and**
2: *T = (Propagated L l # F, snd S)* **and**
3: *C* ∈# *snd S* **and**
4: *fst S* ⊨*as CNot C* **and**
5: *undefined-lit F L* **and**

6: *atm-of L* ∈ *atms-of-mm* (*snd S*) ∪ *atm-of* ' *lits-of-l* (*fst S*) **and**
7: *snd S* ⊨*pm C'* + {#*L*#} **and**
8: *F* ⊨*as CNot C'*
using *backtrack-is-backjump'*[*OF assms*] **by** *force*
show *?thesis*
apply (*cases S*)
using *backjump.intros*[*OF 1 - - 4 5 - - 8*, *of T*] *2 backtrack 1 5 3 6 7*
by (*auto simp*: *state-eq*$_{NOT}$*-def trail-reduce-trail-to*$_{NOT}$*-drop*
reduce-trail-to$_{NOT}$ *simp del*: *state-simp*$_{NOT}$)
qed

lemma *can-do-bt-step*:
assumes
M: *fst S* = *F'* @ *Decided K* # *F* **and**
C ∈# *snd S* **and**
C: *fst S* ⊨*as CNot C*
shows ¬ *no-step backtrack S*
proof −
obtain *L G' G* **where**
backtrack-split (*fst S*) = (*G'*, *L* # *G*)
unfolding *M* **by** (*induction F' rule*: *ann-lit-list-induct*) *auto*
moreover then have *is-decided L*
by (*metis backtrack-split-snd-hd-decided list.distinct*(*1*) *list.sel*(*1*) *snd-conv*)
ultimately show *?thesis*
using *backtrack.intros*[*of S G' L G C*] ‹*C* ∈# *snd S*› *C* **unfolding** *M* **by** *auto*
qed

end

sublocale *dpll-with-backtrack* ⊆ *dpll-with-backjumping-ops*
fst snd λ*L* (*M*, *N*). (*L* # *M*, *N*)
λ(*M*, *N*). (*tl M*, *N*) λ*C* (*M*, *N*). (*M*, {#*C*#} + *N*) λ*C* (*M*, *N*). (*M*, *removeAll-mset C N*)
λ(*M*, *N*). *no-dup M* ∧ *all-decomposition-implies-m N* (*get-all-ann-decomposition M*)
λ- - - *S T*. *backtrack S T*
λ- -. *True*
apply *unfold-locales*
by (*metis* (*mono-tags*, *lifting*) *case-prod-beta comp-def dpll-with-backtrack.backtrack-is-backjump''*
dpll-with-backtrack.can-do-bt-step)

sublocale *dpll-with-backtrack* ⊆ *dpll-with-backjumping*
fst snd λ*L* (*M*, *N*). (*L* # *M*, *N*)
λ(*M*, *N*). (*tl M*, *N*) λ*C* (*M*, *N*). (*M*, {#*C*#} + *N*) λ*C* (*M*, *N*). (*M*, *removeAll-mset C N*)
λ(*M*, *N*). *no-dup M* ∧ *all-decomposition-implies-m N* (*get-all-ann-decomposition M*)
λ- - - *S T*. *backtrack S T*
λ- -. *True*
apply *unfold-locales*
using *dpll-bj-no-dup dpll-bj-all-decomposition-implies-inv* **apply** *fastforce*
done

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

corollary *full-dpll-final-state-conclusive*:
fixes $M\ M' :: ('v,\ unit)$ *ann-lits*
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 $M\ M' :: ('v,\ unit)$ *ann-lits*
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 *distinct-consistent-interp satisfiable-carac′ true-annots-true-cls* **by** *blast*
then show *?thesis*
using *full-dpll-final-state-conclusive*[*OF full*] **by** *auto*
qed

interpretation *conflict-driven-clause-learning-ops*
fst snd $\lambda L\ (M,\ N).\ (L\ \#\ M,\ N)$
$\lambda(M,\ N).\ (tl\ M,\ N)\ \lambda C\ (M,\ N).\ (M,\ \{\#C\#\} + N)\ \lambda C\ (M,\ N).\ (M,\ removeAll\text{-}mset\ C\ N)$
$\lambda(M,\ N).$ *no-dup* $M \wedge$ *all-decomposition-implies-m* N (*get-all-ann-decomposition* M)
λ- - - $S\ T$. *backtrack* $S\ T$
λ- -. *True* λ- -. *False* λ- -. *False*
by *unfold-locales*

interpretation *conflict-driven-clause-learning*
fst snd $\lambda L\ (M,\ N).\ (L\ \#\ M,\ N)$
$\lambda(M,\ N).\ (tl\ M,\ N)\ \lambda C\ (M,\ N).\ (M,\ \{\#C\#\} + N)\ \lambda C\ (M,\ N).\ (M,\ removeAll\text{-}mset\ C\ N)$
$\lambda(M,\ N).$ *no-dup* $M \wedge$ *all-decomposition-implies-m* N (*get-all-ann-decomposition* M)
λ- - - $S\ T$. *backtrack* $S\ T$
λ- -. *True* λ- -. *False* λ- -. *False*
apply *unfold-locales*
using $cdcl_{NOT}$*-all-decomposition-implies* $cdcl_{NOT}$*-no-dup* **by** *fastforce*

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

1.3.2 Adding restarts

This was mainly a test whether it was possible to instantiate the assumption of the locale.

locale *dpll-withbacktrack-and-restarts* =

dpll-with-backtrack +
fixes f :: *nat* ⇒ *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 $\lambda L\ (M,\ N).\ (L\ \#\ M,\ N)\ \lambda(M,\ N).\ (tl\ M,\ N)$
$\lambda C\ (M,\ N).\ (M,\ \{\#C\#\} + N)\ \lambda C\ (M,\ N).\ (M,$ *removeAll-mset* $C\ N)\ f\ \lambda(\text{-},\ N)\ S.\ S = ([],\ N)$
$\lambda A\ (M,\ N).$ *atms-of-mm* $N \subseteq$ *atms-of-ms* $A \land$ *atm-of* ‘ *lits-of-l* $M \subseteq$ *atms-of-ms* $A \land$ *finite* A
$\land$ *all-decomposition-implies-m* N (*get-all-ann-decomposition* M)
$\lambda A\ T.$ (*2+card* (*atms-of-ms* A)) ^ (*1+card* (*atms-of-ms* A))
$-\ \mu_C$ (*1+card* (*atms-of-ms* A)) (*2+card* (*atms-of-ms* A)) (*trail-weight* T) *dpll-bj*
$\lambda(M,\ N).$ *no-dup* $M \land$ *all-decomposition-implies-m* N (*get-all-ann-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 id-apply 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

1.4 Weidenbach's DPLL

1.4.1 Rules

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

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

inductive $dpll_W$ :: $'v$ $dpll_W$*-state* ⇒ $'v$ $dpll_W$*-state* ⇒ *bool* **where**
propagate: $C + \{\#L\#\} \in\#$ *clauses* $S \Longrightarrow$ *trail* $S \models as$ *CNot* $C \Longrightarrow$ *undefined-lit* (*trail* S) L
$\Longrightarrow dpll_W\ S$ (*Propagated* L () # *trail* S, *clauses* S) |
decided: *undefined-lit* (*trail* S) $L \Longrightarrow$ *atm-of* $L \in$ *atms-of-mm* (*clauses* S)
$\Longrightarrow dpll_W\ S$ (*Decided* L # *trail* S, *clauses* S) |
backtrack: *backtrack-split* (*trail* S) = (M', L # M) $\Longrightarrow$ *is-decided* $L \Longrightarrow D \in\#$ *clauses* S
$\Longrightarrow$ *trail* $S \models as$ *CNot* $D \Longrightarrow dpll_W\ S$ (*Propagated* (− (*lit-of* L)) () # M, *clauses* S)

1.4.2 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-l* (*trail* S))
and *no-dup* (*trail* S)
shows *consistent-interp* (*lits-of-l* (*trail* S'))
using *assms*
proof (*induct rule*: $dpll_W$*.induct*)
case (*backtrack* S M' L M D) **note** *extracted* = *this*(*1*) **and** *decided* = *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-l* M) $\subseteq$ *lits-of-l* (*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-l* M))
using *consistent-interp-subset cons* **by** *blast*
moreover
have *lit-of* $L \notin$ *lits-of-l* 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) $\notin$ (λm. *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* $L \notin$ *lits-of-l* 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-l* (*trail* S)) $\subseteq$ *atms-of-mm* (*clauses* S)
shows *atm-of* ‘ (*lits-of-l* (*trail* S')) $\subseteq$ *atms-of-mm* (*clauses* S')
using *assms*
proof (*induct rule*: $dpll_W$*.induct*)
case (*backtrack* S M' L M D)
then have *atm-of* (*lit-of* L) $\in$ *atms-of-mm* (*clauses* S)
using *backtrack-split-list-eq*[*of trail* S, *symmetric*] **by** *auto*
moreover
have *atm-of* ‘ *lits-of-l* (*trail* S) $\subseteq$ *atms-of-mm* (*clauses* S)
using *backtrack*(*5*) **by** *simp*
then have $\bigwedge xb$. $xb \in$ *set* $M \Longrightarrow$ *atm-of* (*lit-of* xb) $\in$ *atms-of-mm* (*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*

theorem 2.8.2 page 73 of Weidenbach's book

lemma $dpll_W$*-propagate-is-conclusion*:
assumes $dpll_W$ *S S'*
and *all-decomposition-implies-m* (*clauses S*) (*get-all-ann-decomposition* (*trail S*))
and *atm-of* ' *lits-of-l* (*trail S*) ⊆ *atms-of-mm* (*clauses S*)
shows *all-decomposition-implies-m* (*clauses S'*) (*get-all-ann-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-ann-decomposition* (*trail S*) = []
then have *?case* **by** *blast*
}
moreover {
assume *n*: *get-all-ann-decomposition* (*trail S*) ≠ []
have *1*: ⋀*a b*. (*a*, *b*) ∈ *set* (*tl* (*get-all-ann-decomposition* (*trail S*)))
⟹ (*unmark-l a* ∪ *set-mset* (*clauses S*)) ⊨*ps unmark-l 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-ann-decomposition* (*trail S*)) = (*a*, *c*)
⟹ (*unmark-l a* ∪ *set-mset* (*clauses S*)) ⊨*ps* (*unmark-l c*)
by (*metis IH all-decomposition-implies-cons-pair all-decomposition-implies-single list.collapse n*)
moreover have *3*: ⋀*a c*. *hd* (*get-all-ann-decomposition* (*trail S*)) = (*a*, *c*)
⟹ (*unmark-l a* ∪ *set-mset* (*clauses S*)) ⊨*p* {#*L*#}
proof −
fix *a c*
assume *h*: *hd* (*get-all-ann-decomposition* (*trail S*)) = (*a*, *c*)
have *h'*: *trail S* = *c* @ *a* **using** *get-all-ann-decomposition-decomp h* **by** *blast*
have *I*: *set* (*map* (λ*a*. {#*lit-of a*#}) *a*) ∪ *set-mset* (*clauses S*)
∪ *unmark-l 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-l 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 sup.coboundedI2*)
using *inS atms-of-atms-of-ms-mono atms-incl* **by** (*fastforce simp*: *h'*)

then have *unmark-l a* ∪ *set-mset* (*clauses S*) ⊨*ps CNot C*

using *true-clss-clss-left-right*[*OF* - *I*] *h 2* **by** *auto*
then show *unmark-l a* ∪ *set-mset* (*clauses S*) ⊨*p* {#*L*#}
by (*metis* (*no-types*) *Un-insert-right inS insertI1 mk-disjoint-insert inS*
true-clss-cls-in true-clss-cls-plus-CNot)
qed
ultimately have *?case*
by (*cases hd* (*get-all-ann-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** *decided* = *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-decided l*
using *extracted backtrack-split-fst-not-decided*[*of* - *trail S*] **by** *simp*
have *n*: *get-all-ann-decomposition* (*trail S*) ≠ [] **by** *auto*
then have *all-decomposition-implies-m* (*clauses S*) ((*L* # *M*, *M′*)
tl (*get-all-ann-decomposition* (*trail S*)))
by (*metis* (*no-types*) *IH extracted get-all-ann-decomposition-backtrack-split list.exhaust-sel*)
then have *1*: *unmark-l* (*L* # *M*) ∪ *set-mset* (*clauses S*) ⊨*ps*(λ*a*.{#*lit-of a*#}) ‘ *set M′*
by *simp*
moreover
have *unmark-l* (*L* # *M*) ∪ *unmark-l 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* (*L* # *M*) ∪ *set-mset* (*clauses S*) ∪ *unmark-l M′*
⊨*ps CNot D*
by (*metis* (*no-types*, *lifting*) *Un-assoc Un-left-commute true-clss-clss-union-l-r*)
ultimately
have *set* (*map* (λ*a*. {#*lit-of a*#}) (*L* # *M*)) ∪ *set-mset* (*clauses S*) ⊨*ps CNot D*
using *true-clss-clss-left-right* **by** *fastforce*
then have *set* (*map* (λ*a*. {#*lit-of a*#}) (*L* # *M*)) ∪ *set-mset* (*clauses S*) ⊨*p* {#}
by (*metis* (*mono-tags*, *lifting*) *D Un-def mem-Collect-eq*
true-clss-clss-contradiction-true-clss-cls-false)
then have *IL*: *unmark-l 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-ann-decomposition*
(*fst* (*Propagated* (− *lit-of L*) *P* # *M*, *clauses S*)))
let *?M′* = *Propagated* (− *lit-of L*) *P* # *M*
let *?hd* = *hd* (*get-all-ann-decomposition ?M′*)
let *?tl* = *tl* (*get-all-ann-decomposition ?M′*)
have *x* = *?hd* ∨ *x* ∈ *set ?tl*
using *x*
by (*cases get-all-ann-decomposition ?M′*)
auto
moreover {
assume *x′*: *x* ∈ *set ?tl*
have *L′*: *Decided* (*lit-of L*) = *L* **using** *decided* **by** (*cases L*, *auto*)
have *x* ∈ *set* (*get-all-ann-decomposition* (*M′* @ *L* # *M*))
using *x′ get-all-ann-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* $(Ls, seen) \Rightarrow$ *unmark-l Ls* $\cup$ *set-mset* (*clauses S*)
$\models$*ps unmark-l seen*
using *decided IH* **by** (*cases L*) (*auto simp add*: *S all-decomposition-implies-def*)
}
moreover {
assume x': $x = ?hd$
have *tl*: *tl* (*get-all-ann-decomposition* (M' @ L # M)) $\neq$ []
proof −
have *f1*: $\bigwedge$*ms*. *length* (*get-all-ann-decomposition* (M' @ *ms*))
= *length* (*get-all-ann-decomposition ms*)
by (*simp add*: M' *get-all-ann-decomposition-remove-undecided-length*)
have *Suc* (*length* (*get-all-ann-decomposition M*)) $\neq$ *Suc 0*
by *blast*
then show *?thesis*
using *f1 decided* **by** (*metis* (*no-types*) *get-all-ann-decomposition.simps*(*1*) *length-tl list.sel*(*3*) *list.size*(*3*) *ann-lit.collapse*(*1*))
qed
obtain $M0'$ $M0$ **where**
L0: *hd* (*tl* (*get-all-ann-decomposition* (M' @ L # M))) = ($M0$, $M0'$)
by (*cases hd* (*tl* (*get-all-ann-decomposition* (M' @ L # M))))
have x'': x = ($M0$, *Propagated* (−*lit-of L*) *P* # $M0'$)
unfolding x' **using** *get-all-ann-decomposition-last-choice tl* M' *L0*
by (*metis decided ann-lit.collapse*(*1*))
obtain *l-get-all-ann-decomposition* **where**
get-all-ann-decomposition (*trail S*) = (L # M, M') # ($M0$, $M0'$) #
l-get-all-ann-decomposition
using *get-all-ann-decomposition-backtrack-split extracted* **by** (*metis* (*no-types*) *L0 S hd-Cons-tl n tl*)
then have $M = M0'$ @ $M0$ **using** *get-all-ann-decomposition-hd-hd* **by** *fastforce*
then have IL': *unmark-l M0* $\cup$ *set-mset* (*clauses S*)
$\cup$ *unmark-l* $M0'$ $\models$*ps* {{#− *lit-of L*#}}
using *IL* **by** (*simp add*: *Un-commute Un-left-commute image-Un*)
moreover have *H*: *unmark-l M0* $\cup$ *set-mset* (*clauses S*)
$\models$*ps unmark-l* $M0'$
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* $(Ls, seen) \Rightarrow$ *unmark-l Ls* $\cup$ *set-mset* (*clauses S*)
$\models$*ps unmark-l seen*
using *true-clss-clss-left-right* **unfolding** x'' **by** *auto*
}
ultimately show *case x of* $(Ls, seen) \Rightarrow$
unmark-l Ls $\cup$ *set-mset* (*snd* (*?M'*, *clauses S*))
$\models$*ps unmark-l seen*
unfolding *snd-conv* **by** *blast*
qed
qed

theorem 2.8.3 page 73 of Weidenbach's book

theorem $dpll_W$*-propagate-is-conclusion-of-decided*:
assumes $dpll_W$ S S'
and *all-decomposition-implies-m* (*clauses S*) (*get-all-ann-decomposition* (*trail S*))
and *atm-of* ' *lits-of-l* (*trail S*) $\subseteq$ *atms-of-mm* (*clauses S*)
shows *set-mset* (*clauses* S') $\cup$ {{#*lit-of L*#} |*L*. *is-decided L* $\wedge$ $L \in$ *set* (*trail* S')}
$\models$*ps* (λa. {#*lit-of a*#}) ' $\bigcup$(*set* ' *snd* ' *set* (*get-all-ann-decomposition* (*trail* S')))
using *all-decomposition-implies-trail-is-implied*[*OF* $dpll_W$*-propagate-is-conclusion*[*OF assms*]] **.**

theorem 2.8.4 page 73 of Weidenbach's book

lemma *only-propagated-vars-unsat*:
  assumes *decided*: $\forall x \in$ *set M*. $\neg$ *is-decided x*
  and *DN*: $D \in N$ **and** *D*: $M \models as$ *CNot D*
  and *inv*: *all-decomposition-implies N* (*get-all-ann-decomposition M*)
  and *atm-incl*: *atm-of* ‘ *lits-of-l M* $\subseteq$ *atms-of-ms N*
  shows *unsatisfiable N*
proof (*rule ccontr*)
  assume $\neg$ *unsatisfiable N*
  then obtain *I* **where**
    I: $I \models s$ *N* **and**
    cons: *consistent-interp I* **and**
    tot: *total-over-m I N*
    unfolding *satisfiable-def* **by** *auto*
  then have *I-D*: $I \models D$
    using *DN* **unfolding** *true-clss-def* **by** *auto*

  have *l0*: $\{\{\#$*lit-of L*$\#\}$ $|$*L*. *is-decided L* $\wedge$ $L \in$ *set M*$\}$ = $\{\}$ **using** *decided* **by** *auto*
  have *atms-of-ms* ($N \cup$ *unmark-l M*) = *atms-of-ms N*
    using *atm-incl* **unfolding** *atms-of-ms-def lits-of-def* **by** *auto*

  then have *total-over-m I* ($N \cup$ (λa. $\{\#$*lit-of a*$\#\}$) ‘ (*set M*))
    using *tot* **unfolding** *total-over-m-def* **by** *auto*
  then have $I \models s$ (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*: $I \models s$ *unmark-l M* **by** *auto*
  {
    fix *K*
    assume $K \in\#$ *D*
    then have $-K \in$ *lits-of-l M*
      by (*auto split*: *if-split-asm*
        intro: *allE*[*OF D*[*unfolded true-annots-def Ball-def*], *of* $\{\#-K\#\}$])
    then have $-K \in I$ **using** *IM true-clss-singleton-lit-of-implies-incl* **by** *fastforce*
  }
  then have $\neg$ $I \models D$ **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-ann-decomposition* (*trail S*))
  and *atm-incl*: *atm-of* ‘ *lits-of-l* (*trail S*) $\subseteq$ *atms-of-mm* (*clauses S*)
  and *consistent-interp* (*lits-of-l* (*trail S*))
  and *no-dup* (*trail S*)
  shows *all-decomposition-implies-m* (*clauses S'*) (*get-all-ann-decomposition* (*trail S'*))
  and *atm-of* ‘ *lits-of-l* (*trail S'*) $\subseteq$ *atms-of-mm* (*clauses S'*)
  and *clauses S* = *clauses S'*
  and *consistent-interp* (*lits-of-l* (*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-ann-decomposition* (*trail S*)) **and**
atm-of ‘ *lits-of-l* (*trail S*) ⊆ *atms-of-mm* (*clauses S*) **and**
clauses S = *clauses S* **and**
consistent-interp (*lits-of-l* (*trail S*)) **and**
no-dup (*trail S*) **using** *assms* **by** *auto*
next
case (*step S' S''*) **note** $dpll_W Star$ = *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-ann-decomposition* (*trail S*)) **and**
atm-incl: *atm-of* ‘ *lits-of-l* (*trail S*) ⊆ *atms-of-mm* (*clauses S*) **and**
cons: *consistent-interp* (*lits-of-l* (*trail S*)) **and**
no-dup (*trail S*)
ultimately have *decomp*: *all-decomposition-implies-m* (*clauses S'*)
(*get-all-ann-decomposition* (*trail S'*)) **and**
atm-incl': *atm-of* ‘ *lits-of-l* (*trail S'*) ⊆ *atms-of-mm* (*clauses S'*) **and**
snd: *clauses S* = *clauses S'* **and**
cons': *consistent-interp* (*lits-of-l* (*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-ann-decomposition* (*trail S''*))
using $dpll_W$*-propagate-is-conclusion*[*OF* $dpll_W$] *decomp atm-incl'* **by** *auto*
show *atm-of* ‘ *lits-of-l* (*trail S''*) ⊆ *atms-of-mm* (*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-l* (*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-ann-decomposition* (*trail S*))
∧ *atm-of* ‘ *lits-of-l* (*trail S*) ⊆ *atms-of-mm* (*clauses S*)
∧ *consistent-interp* (*lits-of-l* (*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-ann-decomposition* (*trail S*))
and *atm-of* ‘ *lits-of-l* (*trail S*) ⊆ *atms-of-mm* (*clauses S*)
and *consistent-interp* (*lits-of-l* (*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-mm N*
shows *rtranclp* $dpll_W$ ([], *N*) (*map Decided M*, *N*)
using *assms*
proof (*induct M*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons L M*)
then have *undefined-lit* (*map Decided M*) *L*
unfolding *defined-lit-def consistent-interp-def* **by** *auto*
moreover have *atm-of L* ∈ *atms-of-mm N* **using** *Cons.prems*(*3*) **by** *auto*
ultimately have $dpll_W$ (*map Decided M*, *N*) (*map Decided* (*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-mm 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-decided L*)
∧ (∃ *C* ∈# *clauses S*. *trail S* ⊨*as CNot C*)))

theorem 2.8.6 page 74 of Weidenbach's book

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-mm N*
shows $dpll_W^{**}$ ([], *N*) (*map Decided M*, *N*)
and *conclusive-*$dpll_W$*-state* (*map Decided M*, *N*)
proof −
show *rtranclp* $dpll_W$ ([], *N*) (*map Decided M*, *N*) **using** $dpll_W$*-can-do-step assms* **by** *auto*
have *map Decided M* ⊨*asm N* **using** *assms*(*1*) *true-annots-decided-true-cls* **by** *auto*
then show *conclusive-*$dpll_W$*-state* (*map Decided M*, *N*)
unfolding *conclusive-*$dpll_W$*-state-def* **by** *auto*
qed

theorem 2.8.5 page 73 of Weidenbach's book

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-l 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-mm 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-decided-or-true-lit true-cls-def by blast
          then show ?thesis
            by (metis Bex-def D atms-of-atms-of-ms-mono rev-subsetD)
        qed
      moreover {
        assume ∃ L. undefined-lit M L ∧ atm-of L ∈ atms-of-mm 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-decided 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-decided l
          then have False
            using backtrack[of (M, N) - - - D ] DN MD assms(2)
              backtrack-split-some-is-decided-then-snd-has-hd[OF l]
            by (metis backtrack-split-snd-hd-decided fst-conv list.distinct(1) list.sel(1) snd-conv)
        }
        ultimately have False by blast
      }
      ultimately show False by blast
     qed
qed
```

1.4.3 Termination

definition $dpll_W$*-mes M n* =
  map (*λl. if is-decided 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-l 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* (*λl. if is-decided l then 2 else 1*) (*rev* (*trail S*))
      @ *replicate* (*card vars* − *length* (*trail S*)) *3*
    = *map* (*λl. if is-decided 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* (*λl. if is-decided l then 2 else 1*) (*rev* (*trail S*))
      @ *replicate* (*card vars* − *length* (*trail S*)) *3*
    = *map* (*λl. if is-decided 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-decided 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

theorem 2.8.7 page 74 of Weidenbach’s book

lemma $dpll_W$*-card-decrease′*:
  assumes *dpll*: $dpll_W$ *S S′*
  and *atm-incl*: *atm-of ‘ lits-of-l* (*trail S*) ⊆ *atms-of-mm* (*clauses S*)
  and *no-dup*: *no-dup* (*trail S*)
  shows ($dpll_W$*-mes* (*trail S′*) (*card* (*atms-of-mm* (*clauses S′*))),
      $dpll_W$*-mes* (*trail S*) (*card* (*atms-of-mm* (*clauses S*)))) ∈ *lex* {(*a*, *b*). *a* < *b*}
proof −

have *finite* (*atms-of-mm* (*clauses S*)) **unfolding** *atms-of-ms-def* **by** *auto*
then have *1*: *length* (*trail S*) $\leq$ *card* (*atms-of-mm* (*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-l* (*trail S'*) $\subseteq$ *atms-of-mm* (*clauses S'*)
using *atm-incl dpll* $dpll_W$*-vars-in-snd-inv*[*OF dpll*] **by** *force*
have *finite* (*atms-of-mm* (*clauses S'*))
unfolding *atms-of-ms-def* **by** *auto*
then have *2*: *length* (*trail S'*) $\leq$ *card* (*atms-of-mm* (*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-mm* (*clauses S*))),
$dpll_W$*-mes* (*trail S*) (*card* (*atms-of-mm* (*clauses S*))))
$\in$ *lexn* $\{(a, b).\ a < b\}$ (*card* (*atms-of-mm* (*clauses S*)))
using $dpll_W$*-card-decrease*[*OF assms*(*1*), *of atms-of-mm* (*clauses S*)] **by** *blast*
then have ($dpll_W$*-mes* (*trail S'*) (*card* (*atms-of-mm* (*clauses S*))),
$dpll_W$*-mes* (*trail S*) (*card* (*atms-of-mm* (*clauses S*)))) $\in$ *lex* $\{(a, b).\ a < b\}$
unfolding *lex-def* **by** *auto*
then show ($dpll_W$*-mes* (*trail S'*) (*card* (*atms-of-mm* (*clauses S'*))),
$dpll_W$*-mes* (*trail S*) (*card* (*atms-of-mm* (*clauses S*)))) $\in$ *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-mm* (*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* $\wedge$ $dpll_W$ *S S'*$\}$
apply (*rule wf-wf-if-measure'*[*OF wf-lex-less*, *of - -*
$\lambda S.$ $dpll_W$*-mes* (*trail S*) (*card* (*atms-of-mm* (*clauses S*)))])
using $dpll_W$*-card-decrease'* **by** *fast*

lemma $dpll_W$*-tranclp-star-commute*:
$\{(S', S).$ $dpll_W$*-all-inv S* $\wedge$ $dpll_W$ *S S'*$\}^+$ = $\{(S', S).$ $dpll_W$*-all-inv S* $\wedge$ *tranclp* $dpll_W$ *S S'*$\}$
(**is** *?A* = *?B*)
proof
{ fix *S S'*
assume (*S*, *S'*) $\in$ *?A*
then have (*S*, *S'*) $\in$ *?B*
by (*induct rule*: *trancl.induct*, *auto*)
}
then show *?A* $\subseteq$ *?B* **by** *blast*
{ fix *S S'*
assume (*S*, *S'*) $\in$ *?B*
then have $dpll_W^{++}$ *S' S* **and** $dpll_W$*-all-inv S'* **by** *auto*
then have (*S*, *S'*) $\in$ *?A*
proof (*induct rule*: *tranclp.induct*)
case *r-into-trancl*
then show *?case* **by** (*simp-all add*: *r-into-trancl'*)

next
case (*tranclr-into-trancl* S S' S'')
then have $(S', S) \in \{a.\ case\ a\ of\ (S', S) \Rightarrow dpll_W\text{-}all\text{-}inv\ S \wedge dpll_W\ S\ S'\}^+$ **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 $(S'', S') \in \{(pa, p).\ dpll_W\text{-}all\text{-}inv\ p \wedge dpll_W\ p\ pa\}^+$
using ‹$dpll_W$*-all-inv* S'› *trancl-into-trancl.hyps*(*3*) **by** *blast*
then show *?case*
using ‹$(S', S) \in \{a.\ case\ a\ of\ (S', S) \Rightarrow dpll_W\text{-}all\text{-}inv\ S \wedge dpll_W\ S\ S'\}^+$› **by** *auto*
qed
}
then show *?B* ⊆ *?A* **by** *blast*
qed

lemma $dpll_W$*-wf-tranclp*: *wf* $\{(S', S).\ dpll_W\text{-}all\text{-}inv\ S \wedge {dpll_W}^{++}\ S\ S'\}$
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*

1.4.4 Final States

Proposition 2.8.1: final states are the normal forms of $dpll_W$

lemma $dpll_W$*-no-more-step-is-a-conclusive-state*:
assumes $\forall S'.\ \neg dpll_W\ S\ S'$
shows *conclusive-*$dpll_W$*-state* S
proof −
have *vars*: ∀ *s* ∈ *atms-of-mm* (*clauses S*). *s* ∈ *atm-of* ‘ *lits-of-l* (*trail S*)
proof (*rule ccontr*)
assume ¬ (∀ *s*∈*atms-of-mm* (*clauses S*). *s* ∈ *atm-of* ‘ *lits-of-l* (*trail S*))
then obtain *L* **where**
L-in-atms: *L* ∈ *atms-of-mm* (*clauses S*) **and**
L-notin-trail: *L* ∉ *atm-of* ‘ *lits-of-l* (*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 *Decided-Propagated-in-iff-in-lits-of-l* **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-decided L*) ∨ (∀ *C*∈#*clauses S*. ¬*trail S* ⊨*as CNot C*)
unfolding *conclusive-*$dpll_W$*-state-def* **by** *auto*
moreover {
assume ∃ *L*∈*set* (*trail S*). *is-decided L*
then obtain *L* M' *M* **where** *L*: *backtrack-split* (*trail S*) = (M', *L* # *M*)
using *backtrack-split-some-is-decided-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-l* (*trail S*)

using *vars* **unfolding** *atms-of-ms-def* **by** *auto*
then have *trail S* $\models$*as CNot D*
using *all-variables-defined-not-imply-cnot*[*of D*] ‹¬ *trail S* $\models$*a D*› **by** *auto*
moreover have *is-decided L*
using *L* **by** (*metis backtrack-split-snd-hd-decided list.distinct*(*1*) *list.sel*(*1*) *snd-conv*)
ultimately have *False*
using *assms*(*1*) $dpll_W$*.backtrack L* ‹*D* ∈# *clauses S*› ‹*trail S* $\models$*as CNot D*› **by** *blast*
}
moreover {
assume *tr*: ∀ *C*∈#*clauses S*. ¬*trail S* $\models$*as CNot C*
obtain *C* **where** *C-in-cls*: *C* ∈# *clauses S* **and** *trC*: ¬ *trail S* $\models$*a C*
using ‹¬ *trail S* $\models$*asm clauses S*› **unfolding** *true-annots-def* **by** *auto*
have ∀ *s*∈*atms-of-ms* {*C*}. *s* ∈ *atm-of* ‘ *lits-of-l* (*trail S*)
using *vars* ‹*C* ∈# *clauses S*› **unfolding** *atms-of-ms-def* **by** *auto*
then have *trail S* $\models$*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* $\models$*asm N* ⟷ *satisfiable* (*set-mset N*) (**is** *?A* ⟷ *?B*)
proof
let *?M'*= *lits-of-l M*
assume *?A*
then have *?M'* $\models$*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-decided L* ∃ *C* ∈# *N*. *M* $\models$*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* $\models$*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

1.4.5 Link with NOT's DPLL

interpretation $dpll_W$*-*$_{NOT}$: *dpll-with-backtrack* **.**

declare $dpll_W$*-*$_{NOT}$*.state-simp*$_{NOT}$[*simp del*]
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*
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*)
apply (*rule dpll*$_W$*-*$_{NOT}$*.bj-propagate*$_{NOT}$)
apply (*rule dpll*$_W$*-*$_{NOT}$*.propagate*$_{NOT}$*.propagate*$_{NOT}$; *simp?*)
apply *fastforce*
apply (*rule dpll*$_W$*-*$_{NOT}$*.bj-decide*$_{NOT}$)
apply (*rule dpll*$_W$*-*$_{NOT}$*.decide*$_{NOT}$*.decide*$_{NOT}$; *simp?*)
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*)
apply (*frule decided*; *simp*)

apply (*elim dpll*$_W$*-*$_{NOT}$*.propagate*$_{NOT}$*E*, *cases S*)
apply (*auto intro*!: *propagate*[*of* - - (-, -), *simplified*])[]
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

Level of literals and clauses

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

abbreviation *count-decided* :: $('v, 'm)$ *ann-lits* $\Rightarrow$ *nat* **where**
count-decided l $\equiv$ *length* (*filter is-decided l*)

abbreviation *get-level* :: $('v, 'm)$ *ann-lits* $\Rightarrow$ $'v$ *literal* $\Rightarrow$ *nat* **where**
get-level S L $\equiv$ *length* (*filter is-decided* (*dropWhile* ($\lambda S.$ *atm-of* (*lit-of S*) $\neq$ *atm-of L*) *S*))

lemma *get-level-uminus*: *get-level M* $(-L)$ = *get-level M L*
  by *auto*

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

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

In *get-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-l M*
  shows *get-level* (*M* @ *M'*) *L* = *get-level M' L*
  using *assms* **by** (*induct M rule*: *ann-lit-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-l M*
  shows *get-level* (*M* @ *M'*) *L* = *get-level M L* + *length* (*filter is-decided M'*)
  using *assms* **by** (*induct M' rule*: *ann-lit-list-induct*) (*auto simp*: *lits-of-def*)

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-decided*[*simp*]:
  assumes *atm-of L* $\notin$ *atm-of* ‘ *lits-of-l S*
  and $\forall s \in set\ S.\ \neg$*is-decided s*
  shows *get-level* (*M* @ *S*) *L* = *get-level M L*
  using *assms* **apply** (*induction S rule*: *ann-lit-list-induct*)
    apply *auto*[*2*]
  apply (*case-tac atm-of L* $\in$ *atm-of* ‘ *lits-of-l M*)
  apply (*auto simp*: *image-iff lits-of-def filter-empty-conv dest*: *set-dropWhileD*)
  done

lemma *get-level-skip-in-all-not-decided*:
fixes M :: $('a, 'b)$ *ann-lits* **and** L :: $'a$ *literal*
assumes $\forall m{\in}set\ M.\ \neg$ *is-decided m*
and *atm-of* $L \in$ *atm-of* ‘ *lits-of-l M*
shows *get-level M L* $= 0$
using *assms* **by** (*induction M rule*: *ann-lit-list-induct*) *auto*

lemma *get-level-skip-all-not-decided*[*simp*]:
fixes M
assumes $\forall m{\in}set\ M.\ \neg$ *is-decided m*
shows *get-level M L* $= 0$
using *assms* **by** (*auto simp*: *filter-empty-conv dest*: *set-dropWhileD*)

abbreviation *MMax M* $\equiv$ *Max* (*set-mset M*)

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

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

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

lemma *get-maximum-level-empty*[*simp*]:
get-maximum-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-level M L* $=$ *get-maximum-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-maximum-level [] $D = 0$
unfolding *get-maximum-level-def* **by** (*simp add*: *image-constant-conv*)

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

lemma *get-maximum-level-plus*:
get-maximum-level M $(D + D')$ $=$ *max* (*get-maximum-level M D*) (*get-maximum-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 $\exists L \in\#D.$ *get-level M L* $= n$
proof −
have *f*: *finite* (*insert* 0 (($\lambda L.$ *get-level M L*) ‘ *set-mset D*)) **by** *auto*
then have $n \in$ (($\lambda L.$ *get-level M L*) ‘ *set-mset D*)
  using *n max Max-in*[*OF f*] **unfolding** *get-maximum-level-def* **by** *simp*
then show $\exists L \in\# D.$ *get-level M L* $= n$ **by** *auto*

qed

lemma *get-maximum-level-skip-first*[*simp*]:
  assumes *atm-of L* $\notin$ *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-lit.sel*(*2*)
    multiset.map-cong0)

lemma *get-maximum-level-skip-beginning*:
  assumes *DH*: $\forall$ *x* $\in$ *atms-of D*. *x* $\notin$ *atm-of* ` *lits-of-l c*
  shows *get-maximum-level* (*c* @ *H*) *D* = *get-maximum-level H D*
proof −
  have (*get-level* (*c* @ *H*)) ` *set-mset D* = (*get-level H*) ` *set-mset D*
    apply (*rule image-cong*)
     apply *simp*
    using *DH* **unfolding** *atms-of-def* **by** *auto*
  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*
  unfolding *get-maximum-level-def* **by** (*simp add*: *image-constant-conv*)

lemma *get-maximum-level-skip-un-decided-not-present*:
  assumes
   $\forall$ *L*$\in$#*D*. *atm-of L* $\notin$ *atm-of* ` *lits-of-l M* **and**
   $\forall$ *m*$\in$*set M*. $\neg$ *is-decided m*
  shows *get-maximum-level* (*M* @ *aa*) *D* = *get-maximum-level aa D*
  using *assms* **unfolding** *get-maximum-level-def* **by** *simp*

lemma *get-maximum-level-union-mset*:
  get-maximum-level M (*A* #$\cup$ *B*) = *get-maximum-level M* (*A* + *B*)
  unfolding *get-maximum-level-def* **by** (*auto simp*: *image-Un*)

lemma *count-decided-rev*[*simp*]:
  count-decided (*rev M*) = *count-decided M*
  by (*auto simp*: *rev-filter*[*symmetric*])

lemma *count-decided-ge-get-level*[*simp*]:
  count-decided M $\geq$ *get-level M L*
  by (*induct M rule*: *ann-lit-list-induct*) (*auto simp add*: *le-max-iff-disj*)

lemma *count-decided-ge-get-maximum-level*:
  count-decided M $\geq$ *get-maximum-level M D*
  using *get-maximum-level-exists-lit-of-max-level* **unfolding** *Bex-def*
  by (*metis get-maximum-level-empty count-decided-ge-get-level le0*)

fun *get-all-mark-of-propagated* **where**
get-all-mark-of-propagated [] = [] |
get-all-mark-of-propagated (*Decided* - # *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-lit-list-induct*) *auto*

Properties about the levels

lemma *atm-lit-of-set-lits-of-l*:
  ($\lambda l.$ *atm-of* (*lit-of* l)) ‘ *set xs* = *atm-of* ‘ *lits-of-l xs*
  unfolding *lits-of-def* **by** *auto*

lemma *le-count-decided-decomp*:
  assumes *no-dup M*
  shows $i <$ *count-decided M* $\longleftrightarrow$ ($\exists$ *c K c'*. *M* = *c* @ *Decided K* # *c'* $\wedge$ *get-level M K* = *Suc i*)
    (**is** *?A* $\longleftrightarrow$ *?B*)
proof
  assume *?B*
  then obtain *c K c'* **where**
    M = *c* @ *Decided K* # *c'* **and** *get-level M K* = *Suc i*
    by *blast*
  then show *?A* **using** *count-decided-ge-get-level*[*of K M*] **by** *auto*
next
  assume *?A*
  then show *?B*
    using ‹*no-dup M*›
    proof (*induction M rule*: *ann-lit-list-induct*)
      case *Nil*
      then show *?case* **by** *simp*
    next
      case (*Decided L M*) **note** *IH* = *this*(*1*) **and** *i* = *this*(*2*) **and** *n-d* = *this*(*3*)
      then have *n-d-M*: *no-dup M* **by** *simp*
      show *?case*
        proof (*cases* $i <$ *count-decided M*)
          case *True*
          then obtain *c K c'* **where**
            M: *M* = *c* @ *Decided K* # *c'* **and** *lev-K*: *get-level M K* = *Suc i*
            using *IH n-d-M* **by** *blast*
          show *?thesis*
            apply (*rule exI*[*of - Decided L* # *c*])
            apply (*rule exI*[*of - K*])
            apply (*rule exI*[*of - c'*])
            using *lev-K n-d* **unfolding** *M* **by** *auto*
        next
          case *False*
          show *?thesis*
            apply (*rule exI*[*of -* []])
            apply (*rule exI*[*of - L*])
            apply (*rule exI*[*of - M*])
            using *False i* **by** *auto*
        qed
      next
        case (*Propagated L mark' M*) **note** *i* = *this*(*2*) **and** *n-d* = *this*(*3*) **and** *IH* = *this*(*1*)
        then obtain *c K c'* **where**
          M: *M* = *c* @ *Decided K* # *c'* **and** *lev-K*: *get-level M K* = *Suc i*
          by *auto*
        show *?case*
          apply (*rule exI*[*of - Propagated L mark'* # *c*])
          apply (*rule exI*[*of - K*])
          apply (*rule exI*[*of - c'*])

using *lev-K n-d* **unfolding** *M* **by** (*auto simp*: *atm-lit-of-set-lits-of-l*)
qed
qed

end
theory *CDCL-W*
imports *CDCL-Abstract-Clause-Representation List-More CDCL-W-Level Wellfounded-More*

begin

Chapter 2

Weidenbach's CDCL

The organisation of the development is the following:

- `CDCL_W.thy` contains the specification of the rules: the rules and the strategy are defined, and we proof the correctness of CDCL.
- `CDCL_W_Termination.thy` contains the proof of termination.
- `CDCL_W_Merge.thy` contains a variant of the calculus: some rules of the raw calculus are always applied together (like the rules analysing the conflict and then backtracking). We define an equivalent version of the calculus where these rules are applied together. This is useful for implementations.
- `CDCL_WNOT.thy` proves the inclusion of Weidenbach's version of CDCL in NOT's version. We use here the version defined in `CDCL_W_Merge.thy`. We need this, because NOT's backjump corresponds to multiple applications of three rules in Weidenbach's calculus. We show also the termination of the calculus without strategy.

We have some variants build on the top of Weidenbach's CDCL calculus:

- `CDCL_W_Incremental.thy` adds incrementality on the top of `CDCL_W.thy`. The way we are doing it is not compatible with `CDCL_W_Merge.thy` , because we add conflicts and the `CDCL_W_Merge.thy` cannot analyse conflicts added externally, because the conflict and analyse are merged.
- `CDCL_W_Restart.thy` adds restart. It is built on the top of `CDCL_W_Merge.thy`.

2.1 Weidenbach's CDCL with Multisets

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

2.1.1 The State

We will abstract the representation of clause and clauses via two locales. We here use multisets, contrary to `CDCL_W_Abstract_State.thy` where we assume only the existence of a conversion to the state.

locale $state_W$*-ops* =

fixes
trail :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* **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*, *'v clause*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒ *'st* **and**
add-learned-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
remove-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
update-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
abbreviation *hd-trail* :: *'st* ⇒ (*'v*, *'v clause*) *ann-lit* **where**
hd-trail S ≡ *hd* (*trail S*)

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

abbreviation *resolve-cls* **where**
resolve-cls L D' E ≡ *remove1-mset* (−*L*) *D'* #∪ *remove1-mset L E*

end

We are using an abstract state to abstract away the detail of the implementation: we do not need to know how the clauses are represented internally, we just need to know that they can be converted to multisets.

Weidenbach state is a five-tuple composed of:

1. the trail is a list of decided literals;
2. the initial set of clauses (that is not changed during the whole calculus);
3. the learned clauses (clauses can be added or remove);
4. the maximum level of the trail;
5. the conflicting clause (if any has been found so far).

There are two different clause representation: one for the conflicting clause (*'v CDCL-Abstract-Clause-Represent*
standing for conflicting clause) and one for the initial and learned clauses (*'v CDCL-Abstract-Clause-Representat*
standing for clause). The representation of the clauses annotating literals in the trail is slightly different: being able to convert it to *'v CDCL-Abstract-Clause-Representation.clause* is enough (needed for function *hd-trail* below).

There are several axioms to state the independance of the different fields of the state: for example, adding a clause to the learned clauses does not change the trail.

locale $state_W$ =
$state_W$*-ops*

— functions about the state:
— getter:
trail init-clss learned-clss backtrack-lvl conflicting
— setter:
cons-trail tl-trail add-learned-cls remove-cls update-backtrack-lvl
update-conflicting

— Some specific states:
init-state
restart-state
for
trail :: $'st \Rightarrow ('v, 'v\ clause)\ ann\text{-}lits$ **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 :: $('v, 'v\ clause)\ ann\text{-}lit \Rightarrow 'st \Rightarrow 'st$ **and**
tl-trail :: $'st \Rightarrow 'st$ **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 :: $'st \Rightarrow 'st$ +
assumes
trail-cons-trail[*simp*]:
$\bigwedge L\ st.\ trail\ (cons\text{-}trail\ L\ st) = L\ \#\ trail\ st$ **and**
trail-tl-trail[*simp*]: $\bigwedge st.\ trail\ (tl\text{-}trail\ st) = tl\ (trail\ st)$ **and**
trail-add-learned-cls[*simp*]:
$\bigwedge C\ st.\ trail\ (add\text{-}learned\text{-}cls\ C\ st) = trail\ st$ **and**
trail-remove-cls[*simp*]:
$\bigwedge C\ st.\ trail\ (remove\text{-}cls\ C\ st) = trail\ st$ **and**
trail-update-backtrack-lvl[*simp*]: $\bigwedge st\ C.\ trail\ (update\text{-}backtrack\text{-}lvl\ C\ st) = trail\ st$ **and**
trail-update-conflicting[*simp*]: $\bigwedge C\ st.\ trail\ (update\text{-}conflicting\ C\ st) = trail\ st$ **and**

init-clss-cons-trail[*simp*]:
$\bigwedge M\ st.\ init\text{-}clss\ (cons\text{-}trail\ M\ st) = init\text{-}clss\ st$
and
init-clss-tl-trail[*simp*]:
$\bigwedge st.\ init\text{-}clss\ (tl\text{-}trail\ st) = init\text{-}clss\ st$ **and**
init-clss-add-learned-cls[*simp*]:
$\bigwedge C\ st.\ init\text{-}clss\ (add\text{-}learned\text{-}cls\ C\ st) = init\text{-}clss\ st$ **and**
init-clss-remove-cls[*simp*]:
$\bigwedge C\ st.\ init\text{-}clss\ (remove\text{-}cls\ C\ st) = removeAll\text{-}mset\ C\ (init\text{-}clss\ st)$ **and**
init-clss-update-backtrack-lvl[*simp*]:
$\bigwedge st\ C.\ init\text{-}clss\ (update\text{-}backtrack\text{-}lvl\ C\ st) = init\text{-}clss\ st$ **and**
init-clss-update-conflicting[*simp*]:
$\bigwedge C\ st.\ init\text{-}clss\ (update\text{-}conflicting\ C\ st) = init\text{-}clss\ st$ **and**

learned-clss-cons-trail[*simp*]:
$\bigwedge M\ st.\ learned\text{-}clss\ (cons\text{-}trail\ M\ st) = learned\text{-}clss\ st$ **and**
learned-clss-tl-trail[*simp*]:
$\bigwedge st.\ learned\text{-}clss\ (tl\text{-}trail\ st) = learned\text{-}clss\ st$ **and**
learned-clss-add-learned-cls[*simp*]:

$\bigwedge C\ st.\ learned\text{-}clss\ (add\text{-}learned\text{-}cls\ C\ st) = \{\#C\#\} + learned\text{-}clss\ st$ **and**
learned-clss-remove-cls[*simp*]:
$\bigwedge C\ st.\ learned\text{-}clss\ (remove\text{-}cls\ C\ st) = removeAll\text{-}mset\ C\ (learned\text{-}clss\ st)$ **and**
learned-clss-update-backtrack-lvl[*simp*]:
$\bigwedge st\ C.\ learned\text{-}clss\ (update\text{-}backtrack\text{-}lvl\ C\ st) = learned\text{-}clss\ st$ **and**
learned-clss-update-conflicting[*simp*]:
$\bigwedge C\ st.\ learned\text{-}clss\ (update\text{-}conflicting\ C\ st) = learned\text{-}clss\ st$ **and**

backtrack-lvl-cons-trail[*simp*]:
$\bigwedge M\ st.\ backtrack\text{-}lvl\ (cons\text{-}trail\ M\ st) = backtrack\text{-}lvl\ st$ **and**
backtrack-lvl-tl-trail[*simp*]:
$\bigwedge st.\ backtrack\text{-}lvl\ (tl\text{-}trail\ st) = backtrack\text{-}lvl\ st$ **and**
backtrack-lvl-add-learned-cls[*simp*]:
$\bigwedge C\ st.\ backtrack\text{-}lvl\ (add\text{-}learned\text{-}cls\ C\ st) = backtrack\text{-}lvl\ st$ **and**
backtrack-lvl-remove-cls[*simp*]:
$\bigwedge C\ st.\ backtrack\text{-}lvl\ (remove\text{-}cls\ C\ st) = backtrack\text{-}lvl\ st$ **and**
backtrack-lvl-update-backtrack-lvl[*simp*]:
$\bigwedge st\ k.\ backtrack\text{-}lvl\ (update\text{-}backtrack\text{-}lvl\ k\ st) = k$ **and**
backtrack-lvl-update-conflicting[*simp*]:
$\bigwedge C\ st.\ backtrack\text{-}lvl\ (update\text{-}conflicting\ C\ st) = backtrack\text{-}lvl\ st$ **and**

conflicting-cons-trail[*simp*]:
$\bigwedge M\ st.\ conflicting\ (cons\text{-}trail\ M\ st) = conflicting\ st$ **and**
conflicting-tl-trail[*simp*]:
$\bigwedge st.\ conflicting\ (tl\text{-}trail\ st) = conflicting\ st$ **and**
conflicting-add-learned-cls[*simp*]:
$\bigwedge C\ st.\ conflicting\ (add\text{-}learned\text{-}cls\ C\ st) = conflicting\ st$
and
conflicting-remove-cls[*simp*]:
$\bigwedge C\ st.\ conflicting\ (remove\text{-}cls\ C\ st) = conflicting\ st$ **and**
conflicting-update-backtrack-lvl[*simp*]:
$\bigwedge st\ C.\ conflicting\ (update\text{-}backtrack\text{-}lvl\ C\ st) = conflicting\ st$ **and**
conflicting-update-conflicting[*simp*]:
$\bigwedge C\ st.\ conflicting\ (update\text{-}conflicting\ C\ st) = C$ **and**

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

trail-restart-state[*simp*]: $trail\ (restart\text{-}state\ S) = []$ **and**
init-clss-restart-state[*simp*]: $init\text{-}clss\ (restart\text{-}state\ S) = init\text{-}clss\ S$ **and**
learned-clss-restart-state[*intro*]:
$learned\text{-}clss\ (restart\text{-}state\ S) \subseteq\#\ learned\text{-}clss\ S$ **and**
backtrack-lvl-restart-state[*simp*]: $backtrack\text{-}lvl\ (restart\text{-}state\ S) = 0$ **and**
conflicting-restart-state[*simp*]: $conflicting\ (restart\text{-}state\ S) = None$
begin

lemma
shows
clauses-cons-trail[*simp*]:
$clauses\ (cons\text{-}trail\ M\ S) = clauses\ S$ **and**

clss-tl-trail[*simp*]: $clauses\ (tl\text{-}trail\ S) = clauses\ S$ **and**
clauses-add-learned-cls-unfolded:

clauses (*add-learned-cls U S*) = {#*U*#} + *learned-clss S* + *init-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*) = *removeAll-mset C* (*clauses S*) **and**
clauses-add-learned-cls[*simp*]:
clauses (*add-learned-cls C S*) = {#*C*#} + *clauses S* **and**
clauses-restart[*simp*]: *clauses* (*restart-state S*) ⊆# *clauses S* **and**
clauses-init-state[*simp*]: *clauses* (*init-state N*) = *N*
by (*auto simp*: *ac-simps replicate-mset-plus clauses-def intro*: *multiset-eqI*)

abbreviation *state* :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* × *'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 ∼ *T* ⟷ *state S* = *state T*

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

lemma *state-eq-sym*:
S ∼ *T* ⟷ *T* ∼ *S*
unfolding *state-eq-def* **by** *auto*

lemma *state-eq-trans*:
S ∼ *T* ⟹ *T* ∼ *U* ⟹ *S* ∼ *U*
unfolding *state-eq-def* **by** *auto*

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

lemma *state-eq-conflicting-None*:
S ∼ *T* ⟹ *conflicting T* = *None* ⟹ *conflicting S* = *None*
unfolding *state-eq-def clauses-def* **by** *auto*

We combine all simplification rules about *op* ∼ in a single list of theorems. While they are handy as simplification rule as long as we are working on the state, they also cause a *huge* slow-down in all other cases.

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
state-eq-conflicting-None

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

function *reduce-trail-to* :: *'a list* ⇒ *'st* ⇒ *'st* **where**
reduce-trail-to F S =
  (*if length* (*trail* S) = *length* F ∨ *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 = [] ⟹ *reduce-trail-to* F S = S **and**
    reduce-trail-to-eq-length[*simp*]: *length* (*trail* S) = *length* F ⟹ *reduce-trail-to* F S = S
  by (*auto simp*: *reduce-trail-to.simps*)

lemma *reduce-trail-to-length-ne*:
  length (*trail* S) ≠ *length* F ⟹ *trail* S ≠ [] ⟹
    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*, *'v clause*) *ann-lits* 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-trail*[*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-trail*[*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-trail*[*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-trail*[*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 *conflicting-reduce-trail-to*[*simp*]:
conflicting (*reduce-trail-to F S*) = *None* $\longleftrightarrow$ *conflicting S* = *None*
apply (*induction F S rule*: *reduce-trail-to.induct*)
by (*metis conflicting-update-trail map-option-is-None*)

lemma *trail-eq-reduce-trail-to-eq*:
trail S = *trail T* $\Longrightarrow$ *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' @ *Decided K* # *F* $\Longrightarrow$ (*trail* (*reduce-trail-to F S*)) = *F*
apply (*rule reduce-trail-to-skip-beginning*[*of - F' @ Decided K # []*])
by (*cases* F') (*auto simp add*:*tl-append reduce-trail-to-skip-beginning*)

lemma *reduce-trail-to-add-learned-cls*[*simp*]:
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-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 *reduce-trail-to-length*:
length M = *length M'* $\Longrightarrow$ *reduce-trail-to M S* = *reduce-trail-to M' S*
apply (*induction M S rule*: *reduce-trail-to.induct*)
by (*simp add*: *reduce-trail-to.simps*)

lemma *trail-reduce-trail-to-drop*:
trail (*reduce-trail-to 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.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** (*metis diff-is-0-eq drop-Cons' length-Cons nat-le-linear nat-less-le reduce-trail-to-eq-length trail-reduce-trail-to-length-le*)
apply (*subgoal-tac Suc* (*length list*) $-$ *length F* = *Suc* (*length list* $-$ *length F*))
by (*auto simp add*: *reduce-trail-to-length-ne*)

lemma *in-get-all-ann-decomposition-trail-update-trail*[*simp*]:
assumes *H*: (*L* # *M1*, *M2*) $\in$ *set* (*get-all-ann-decomposition* (*trail S*))
shows *trail* (*reduce-trail-to M1 S*) = *M1*
proof $-$
obtain *K* **where**
L: *L* = *Decided K*
using *H* **by** (*cases L*) (*auto dest*!: *in-get-all-ann-decomposition-decided-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*])
(*auto simp*: *tr-S L*)
qed

lemma *conflicting-cons-trail-conflicting*[*simp*]:
assumes *undefined-lit* (*trail S*) (*lit-of L*)
shows
conflicting (*cons-trail L S*) = *None* $\longleftrightarrow$ *conflicting S* = *None*
using *assms conflicting-cons-trail*[*of L S*] *map-option-is-None* **by** *fastforce+*

lemma *conflicting-add-learned-cls-conflicting*[*simp*]:
conflicting (*add-learned-cls C S*) = *None* $\longleftrightarrow$ *conflicting S* = *None*
by *fastforce+*

lemma *conflicting-update-backtracl-lvl*[*simp*]:
conflicting (*update-backtrack-lvl k S*) = *None* $\longleftrightarrow$ *conflicting S* = *None*
using *map-option-is-None conflicting-update-backtrack-lvl*[*of k S*] **by** *fastforce+*

end — end of $state_W$ locale

2.1.2 CDCL Rules

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

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

— get state:
init-state
restart-state
for
trail :: $'st \Rightarrow ('v, 'v\ clause)\ ann\text{-}lits$ **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 :: $('v, 'v\ clause)\ ann\text{-}lit \Rightarrow 'st \Rightarrow 'st$ **and**
tl-trail :: $'st \Rightarrow 'st$ **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* :: $'st \Rightarrow 'st \Rightarrow bool$ **for** S :: $'st$ **where**
propagate-rule: $conflicting\ S = None \Longrightarrow$
$E \in\#\ clauses\ S \Longrightarrow$
$L \in\#\ E \Longrightarrow$
$trail\ S \models as\ CNot\ (E - \{\#L\#\}) \Longrightarrow$
$undefined\text{-}lit\ (trail\ S)\ L \Longrightarrow$
$T \sim cons\text{-}trail\ (Propagated\ L\ E)\ S \Longrightarrow$
$propagate\ S\ T$

inductive-cases *propagateE*: *propagate S T*

inductive *conflict* :: $'st \Rightarrow 'st \Rightarrow bool$ **for** S :: $'st$ **where**
conflict-rule:
$conflicting\ S = None \Longrightarrow$
$D \in\#\ clauses\ S \Longrightarrow$
$trail\ S \models as\ CNot\ D \Longrightarrow$
$T \sim update\text{-}conflicting\ (Some\ D)\ S \Longrightarrow$
$conflict\ S\ T$

inductive-cases *conflictE*: *conflict S T*

inductive *backtrack* :: $'st \Rightarrow 'st \Rightarrow bool$ **for** S :: $'st$ **where**

backtrack-rule:
  conflicting S = Some D $\Longrightarrow$
  $L \in\# D \Longrightarrow$
  (*Decided K* # *M1*, *M2*) $\in$ *set* (*get-all-ann-decomposition* (*trail S*)) $\Longrightarrow$
  get-level (*trail S*) *L* = *backtrack-lvl S* $\Longrightarrow$
  get-level (*trail S*) *L* = *get-maximum-level* (*trail S*) *D* $\Longrightarrow$
  get-maximum-level (*trail S*) (*D* − {#*L*#}) $\equiv$ *i* $\Longrightarrow$
  get-level (*trail S*) *K* = *i* + *1* $\Longrightarrow$
  T $\sim$ *cons-trail* (*Propagated L D*)
      (*reduce-trail-to M1*
        (*add-learned-cls D*
          (*update-backtrack-lvl i*
            (*update-conflicting None S*)))) $\Longrightarrow$
  backtrack S T

inductive-cases *backtrackE*: *backtrack S T*
thm *backtrackE*

inductive *decide* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **for** *S* :: *'st* **where**
decide-rule:
  conflicting S = None $\Longrightarrow$
  undefined-lit (*trail S*) *L* $\Longrightarrow$
  atm-of L $\in$ *atms-of-mm* (*init-clss S*) $\Longrightarrow$
  T $\sim$ *cons-trail* (*Decided L*) (*incr-lvl S*) $\Longrightarrow$
  decide S T

inductive-cases *decideE*: *decide S T*

inductive *skip* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **for** *S* :: *'st* **where**
skip-rule:
  trail S = Propagated L C' # *M* $\Longrightarrow$
   conflicting S = Some E $\Longrightarrow$
   $-L \notin\# E \Longrightarrow$
   $E \neq \{\#\} \Longrightarrow$
   T $\sim$ *tl-trail S* $\Longrightarrow$
   skip S T

inductive-cases *skipE*: *skip S T*

get-maximum-level (*Propagated L* (*C* + {#*L*#}) # *M*) *D* = *k* $\vee$ *k* = *0* (that was in a previous version of the book) is equivalent to *get-maximum-level* (*Propagated L* (*C* + {#*L*#}) # *M*) *D* = *k*, when the structural invariants holds.

inductive *resolve* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **for** *S* :: *'st* **where**
resolve-rule: *trail S* $\neq$ [] $\Longrightarrow$
  hd-trail S = Propagated L E $\Longrightarrow$
  $L \in\# E \Longrightarrow$
  conflicting S = Some D' $\Longrightarrow$
  $-L \in\# D' \Longrightarrow$
  get-maximum-level (*trail S*) ((*remove1-mset* (−*L*) *D'*)) = *backtrack-lvl S* $\Longrightarrow$
  T $\sim$ *update-conflicting* (*Some* (*resolve-cls L D' E*))
    (*tl-trail S*) $\Longrightarrow$
  resolve S T

inductive-cases *resolveE*: *resolve S T*

inductive *restart* :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
restart: *state* $S = (M, N, U, k, None) \Longrightarrow \neg M \models asm\ clauses\ S$
$\Longrightarrow T \sim restart\text{-}state\ S$
$\Longrightarrow restart\ S\ T$

inductive-cases *restartE*: *restart S T*

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

inductive *forget* :: *'st* ⇒ *'st* ⇒ *bool* **where**
forget-rule:
conflicting $S = None \Longrightarrow$
$C \in\#$ *learned-clss* $S \Longrightarrow$
$\neg(trail\ S) \models asm\ clauses\ S \Longrightarrow$
$C \notin set\ (get\text{-}all\text{-}mark\text{-}of\text{-}propagated\ (trail\ S)) \Longrightarrow$
$C \notin\#$ *init-clss* $S \Longrightarrow$
$T \sim remove\text{-}cls\ C\ S \Longrightarrow$
forget S T

inductive-cases *forgetE*: *forget S T*

inductive $cdcl_W$*-rf* :: *'st* ⇒ *'st* ⇒ *bool* **for** *S* :: *'st* **where**
restart: *restart S T* ⟹ $cdcl_W$*-rf S T* |
forget: *forget S T* ⟹ $cdcl_W$*-rf S T*

inductive $cdcl_W$*-bj* :: *'st* ⇒ *'st* ⇒ *bool* **where**
skip: *skip S S'* ⟹ $cdcl_W$*-bj S S'* |
resolve: *resolve S S'* ⟹ $cdcl_W$*-bj S S'* |
backtrack: *backtrack S S'* ⟹ $cdcl_W$*-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: *decide S S'* ⟹ $cdcl_W$*-o S S'* |
bj: $cdcl_W$*-bj S S'* ⟹ $cdcl_W$*-o S S'*

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

lemma *rtranclp-propagate-is-rtranclp-*$cdcl_W$:
$propagate^{**}\ S\ S' \Longrightarrow cdcl_W^{**}\ S\ S'$
apply (*induction rule*: *rtranclp-induct*)
apply *simp*
apply (*frule propagate*)
using *rtranclp-trans*[*of* $cdcl_W$] **by** *blast*

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* ⟹ *P S T* **and**
conflict: $\bigwedge T$. *conflict S T* ⟹ *P S T* **and**
forget: $\bigwedge T$. *forget S T* ⟹ *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. *conflicting* S = *None* $\Longrightarrow$
$C \in\#$ *clauses* $S \Longrightarrow$
$L \in\#$ $C \Longrightarrow$
trail S $\models$*as CNot* (*remove1-mset* L C) $\Longrightarrow$
undefined-lit (*trail* S) $L \Longrightarrow$
$T \sim$ *cons-trail* (*Propagated* L C) $S \Longrightarrow$
P S T **and**
conflictH: $\bigwedge D$ T. *conflicting* S = *None* $\Longrightarrow$
$D \in\#$ *clauses* $S \Longrightarrow$
trail S $\models$*as CNot* $D \Longrightarrow$
$T \sim$ *update-conflicting* (*Some* D) $S \Longrightarrow$
P S T **and**
forgetH: $\bigwedge C$ T. *conflicting* S = *None* $\Longrightarrow$
$C \in\#$ *learned-clss* $S \Longrightarrow$
$\neg$(*trail* S) $\models$*asm clauses* $S \Longrightarrow$
$C \notin$ *set* (*get-all-mark-of-propagated* (*trail* S)) $\Longrightarrow$
$C \notin\#$ *init-clss* $S \Longrightarrow$
$T \sim$ *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-mm* (*init-clss* S) $\Longrightarrow$
$T \sim$ *cons-trail* (*Decided* L) (*incr-lvl* S) $\Longrightarrow$
$P\ S\ T$ **and**
skipH: $\bigwedge L\ C'\ M\ E\ T.$
trail $S =$ *Propagated* $L\ C'\ \#\ M \Longrightarrow$
conflicting $S =$ *Some* $E \Longrightarrow$
$-L \notin\#\ E \Longrightarrow E \neq \{\#\} \Longrightarrow$
$T \sim$ *tl-trail* $S \Longrightarrow$
$P\ S\ T$ **and**
resolveH: $\bigwedge L\ E\ M\ D\ T.$
trail $S =$ *Propagated* $L\ E\ \#\ M \Longrightarrow$
$L \in\#\ E \Longrightarrow$
hd-trail $S =$ *Propagated* $L\ E \Longrightarrow$
conflicting $S =$ *Some* $D \Longrightarrow$
$-L \in\#\ D \Longrightarrow$
get-maximum-level (*trail* S) ((*remove1-mset* $(-L)\ D$)) $=$ *backtrack-lvl* $S \Longrightarrow$
$T \sim$ *update-conflicting*
(*Some* (*resolve-cls* $L\ D\ E$)) (*tl-trail* S) $\Longrightarrow$
$P\ S\ T$ **and**
backtrackH: $\bigwedge L\ D\ K\ i\ M1\ M2\ T.$
conflicting $S =$ *Some* $D \Longrightarrow$
$L \in\#\ D \Longrightarrow$
(*Decided* $K\ \#\ M1$, $M2$) $\in$ *set* (*get-all-ann-decomposition* (*trail* S)) $\Longrightarrow$
get-level (*trail* S) $L =$ *backtrack-lvl* $S \Longrightarrow$
get-level (*trail* S) $L =$ *get-maximum-level* (*trail* S) $D \Longrightarrow$
get-maximum-level (*trail* S) (*remove1-mset* $L\ D$) $\equiv i \Longrightarrow$
get-level (*trail* S) $K = i{+}1 \Longrightarrow$
$T \sim$ *cons-trail* (*Propagated* $L\ D$)
(*reduce-trail-to* $M1$
(*add-learned-cls* D
(*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 (*auto elim*!: *propagateE intro*!: *propagateH*)
next
case (*conflict* S')
then show *?case*
by (*auto elim*!: *conflictE intro*!: *conflictH*)
next
case (*restart* S')
then show *?case*
by (*auto elim*!: *restartE intro*!: *restartH*)
next
case (*decide* T)
then show *?case*
by (*auto elim*!: *decideE intro*!: *decideH*)
next
case (*backtrack* S')

then show *?case* **by** (*auto elim*!: *backtrackE intro*!: *backtrackH*
simp del: *state-simp simp add*: *state-eq-def*)
next
case (*forget* S')
then show *?case* **by** (*auto elim*!: *forgetE intro*!: *forgetH*)
next
case (*skip* S')
then show *?case* **by** (*auto elim*!: *skipE intro*!: *skipH*)
next
case (*resolve* S')
then show *?case*
by (*cases trail S*) (*auto elim*!: *resolveE intro*!: *resolveH*)
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-mm* (*init-clss S*)
$\Longrightarrow$ *T* $\sim$ *cons-trail* (*Decided L*) (*incr-lvl S*)
$\Longrightarrow$ *P S T* **and**
skipH: $\bigwedge L\ C'\ M\ E\ T$.
trail S = Propagated L C' # M $\Longrightarrow$
conflicting S = Some E $\Longrightarrow$
$-L \notin\#\ E \Longrightarrow E \neq \{\#\} \Longrightarrow$
T $\sim$ *tl-trail S* $\Longrightarrow$
P S T **and**
resolveH: $\bigwedge L\ E\ M\ D\ T$.
trail S = Propagated L E # M $\Longrightarrow$
$L \in\#\ E \Longrightarrow$
hd-trail S = Propagated L E $\Longrightarrow$
conflicting S = Some D $\Longrightarrow$
$-L \in\#\ D \Longrightarrow$
get-maximum-level (*trail S*) ((*remove1-mset* ($-L$) *D*)) = *backtrack-lvl S* $\Longrightarrow$
T $\sim$ *update-conflicting*
(*Some* (*resolve-cls L D E*)) (*tl-trail S*) $\Longrightarrow$
P S T **and**
backtrackH: $\bigwedge L\ D\ K\ i\ M1\ M2\ T$.
conflicting S = Some D $\Longrightarrow$
$L \in\#\ D \Longrightarrow$
(*Decided K # M1*, *M2*) $\in$ *set* (*get-all-ann-decomposition* (*trail S*)) $\Longrightarrow$
get-level (*trail S*) *L = backtrack-lvl S* $\Longrightarrow$
get-level (*trail S*) *L = get-maximum-level* (*trail S*) *D* $\Longrightarrow$
get-maximum-level (*trail S*) (*remove1-mset L D*) $\equiv$ *i* $\Longrightarrow$
get-level (*trail S*) *K = i + 1* $\Longrightarrow$
T $\sim$ *cons-trail* (*Propagated L D*)
(*reduce-trail-to M1*
(*add-learned-cls D*
(*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 elim*: *decideE*)[*1*]
apply (*elim* $cdcl_W$*-bjE skipE resolveE backtrackE*)
apply (*frule skipH*; *simp*)

apply (*cases trail S*; *auto elim*!: *resolveE intro*!: *resolveH*)
apply (*frule backtrackH*; *simp*)
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*)

2.1.3 Structural Invariants

Properties of the trail

We here establish that:

- 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: (*Decided K* # *M1*, *M2*) $\in$ *set* (*get-all-ann-decomposition* (*trail S*)) **and**
no-dup: *no-dup* (*trail S*) **and**
bt-l: *backtrack-lvl S* = *length* (*filter is-decided* (*trail S*)) **and**
lev-K: *get-level* (*trail S*) $K = i + 1$
shows *atm-of L* $\notin$ *atm-of* ' *lits-of-l M1*
proof (*rule ccontr*)
let *?M* = *trail S*
assume *L-in-M1*: $\neg$*atm-of L* $\notin$ *atm-of* ' *lits-of-l M1*
obtain *c* **where**
Mc: *trail S* = *c* @ *M2* @ *Decided K* # *M1*
using *M1* **by** *blast*
have *atm-of L* $\notin$ *atm-of* ' *lits-of-l c* **and** *atm-of L* $\notin$ *atm-of* ' *lits-of-l M2* **and**
atm-of L $\neq$ *atm-of K* **and** *Kc*: *atm-of K* $\notin$ *atm-of* ' *lits-of-l c* **and**
KM2: *atm-of K* $\notin$ *atm-of* ' *lits-of-l M2*
using *L-in-M1 no-dup* **unfolding** *Mc lits-of-def* **by** *force+*
then have *g-M-eq-g-M1*: *get-level ?M L* = *get-level M1 L*

using *L-in-M1* **unfolding** *Mc* **by** *auto*
then have *get-level M1 L* $<$ *Suc i*
using *count-decided-ge-get-level*[*of L M1*] *KM2 lev-K Kc* **unfolding** *Mc*
by (*auto simp del*: *count-decided-ge-get-level*)
moreover have *Suc i* $\leq$ *backtrack-lvl S* **using** *bt-l KM2 lev-K Kc* **unfolding** *Mc* **by** (*simp add*: *Mc*)
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**
bt-lev: *backtrack-lvl S* = *count-decided* (*trail S*)
shows *no-dup* (*trail S'*)
using *assms*
proof (*induct rule*: $cdcl_W$*-all-induct*)
case (*backtrack L D K i M1 M2 T*) **note** *decomp* = *this*(*3*) **and** *L* = *this*(*4*) **and** *lev-K* = *this*(*7*) **and**
T = *this*(*8*) **and** *n-d* = *this*(*9*)
obtain *c* **where** *Mc*: *trail S* = *c* @ *M2* @ *Decided K* # *M1*
using *decomp* **by** *auto*
have *no-dup* (*M2* @ *Decided K* # *M1*)
using *Mc n-d* **by** *fastforce*
moreover have *atm-of L* $\notin$ *atm-of* ‘ *lits-of-l M1*
using *backtrack-lit-skiped*[*of L S K M1 M2 i*] *L decomp lev-K n-d bt-lev* **by** *fast*
moreover then have *undefined-lit M1 L*
by (*simp add*: *defined-lit-map lits-of-def image-image*)
ultimately show *?case* **using** *decomp T n-d* **by** (*simp add*: *lits-of-def image-image*)
qed (*auto simp*: *defined-lit-map*)

Item 1 page 81 of Weidenbach’s book

lemma $cdcl_W$*-consistent-inv-2*:
assumes
$cdcl_W$ S S' **and**
no-dup (*trail S*) **and**
backtrack-lvl S = *count-decided* (*trail S*)
shows *consistent-interp* (*lits-of-l* (*trail S'*))
using $cdcl_W$*-distinctinv-1*[*OF assms*] *distinct-consistent-interp* **by** *fast*

lemma $cdcl_W$*-o-bt*:
assumes
$cdcl_W$*-o S S'* **and**
backtrack-lvl S = *count-decided* (*trail S*) **and**
n-d[*simp*]: *no-dup* (*trail S*)
shows *backtrack-lvl S'* = *count-decided* (*trail S'*)
using *assms*
proof (*induct rule*: $cdcl_W$*-o-induct*)
case (*backtrack L D K i M1 M2 T*) **note** *decomp* = *this*(*3*) **and** *levK* = *this*(*7*) **and** *T* = *this*(*8*) **and**
level = *this*(*9*)
have [*simp*]: *trail* (*reduce-trail-to M1 S*) = *M1*
using *decomp* **by** *auto*
obtain *c* **where** *M*: *trail S* = *c* @ *M2* @ *Decided K* # *M1* **using** *decomp* **by** *auto*
moreover have *atm-of L* $\notin$ *atm-of* ‘ *lits-of-l M1*
using *backtrack-lit-skiped*[*of L S K M1 M2 i*] *backtrack*(*4*,*8*,*9*) *levK decomp*
by (*fastforce simp add*: *lits-of-def*)

moreover then have *undefined-lit M1 L*
  by (*simp add*: *defined-lit-map lits-of-def image-image*)
moreover
  have *atm-of K* $\notin$ *atm-of* ‘ *lits-of-l M1* **and** *atm-of K* $\notin$ *atm-of* ‘ *lits-of-l c*
    and *atm-of K* $\notin$ *atm-of* ‘ *lits-of-l M2*
    using *T n-d levK* **unfolding** *M* **by** (*auto simp*: *lits-of-def*)
ultimately show *?case*
  using *T levK* **unfolding** *M* **by** (*auto dest*!: *append-cons-eq-upt-length*)
qed *auto*

lemma *cdcl*$_W$*-rf-bt*:
assumes
  cdcl$_W$*-rf S S'* **and**
  backtrack-lvl S = *count-decided* (*trail S*)
shows *backtrack-lvl S'* = *count-decided* (*trail S'*)
using *assms* **by** (*induct rule*: *cdcl*$_W$*-rf.induct*) (*auto elim*: *restartE forgetE*)

Item 7 page 81 of Weidenbach's book

lemma *cdcl*$_W$*-bt*:
assumes
  cdcl$_W$ *S S'* **and**
  backtrack-lvl S = *count-decided* (*trail S*) **and**
  no-dup (*trail S*)
shows *backtrack-lvl S'* = *count-decided* (*trail S'*)
using *assms* **by** (*induct rule*: *cdcl*$_W$*.induct*) (*auto simp*: *cdcl*$_W$*-o-bt cdcl*$_W$*-rf-bt*
  elim: *conflictE propagateE*)

We write *1* + *count-decided* (*trail S*) instead of *backtrack-lvl S* to avoid non termination of rewriting.

definition *cdcl*$_W$*-M-level-inv* :: *'st* $\Rightarrow$ *bool* **where**
cdcl$_W$*-M-level-inv S* $\longleftrightarrow$
  consistent-interp (*lits-of-l* (*trail S*))
  $\wedge$ *no-dup* (*trail S*)
  $\wedge$ *backtrack-lvl S* = *count-decided* (*trail S*)

lemma *cdcl*$_W$*-M-level-inv-decomp*:
assumes *cdcl*$_W$*-M-level-inv S*
shows
  consistent-interp (*lits-of-l* (*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*
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* ≤ *backtrack-lvl S*
using *inv* **unfolding** *cdcl*$_W$*-M-level-inv-def*
by *simp*

lemma *backtrack-ex-decomp*:
assumes
M-l: *cdcl*$_W$*-M-level-inv S* **and**
i-S: *i* < *backtrack-lvl S*
shows ∃ *K M1 M2*. (*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail S*)) ∧
get-level (*trail S*) *K* = *Suc i*
proof −
let *?M* = *trail S*
have *i* < *count-decided* (*trail S*)
using *i-S M-l* **by** (*auto simp*: *cdcl*$_W$*-M-level-inv-def*)
then obtain *c K c'* **where** *tr-S*: *trail S* = *c* @ *Decided K* # *c'* **and**
lev-K: *get-level* (*trail S*) *K* = *Suc i*
using *le-count-decided-decomp*[*of trail S i*] *M-l* **by** (*auto simp*: *cdcl*$_W$*-M-level-inv-def*)
obtain *M1 M2* **where** (*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail S*))
using *Decided-cons-in-get-all-ann-decomposition-append-Decided-cons* **unfolding** *tr-S* **by** *fast*
then show *?thesis* **using** *lev-K* **by** *blast*
qed

Compatibility with *op* ∼

lemma *propagate-state-eq-compatible*:
assumes
propa: *propagate S T* **and**
SS': *S* ∼ *S'* **and**
TT': *T* ∼ *T'*
shows *propagate S' T'*
proof −
obtain *C L* **where**
conf: *conflicting S* = *None* **and**
C: *C* ∈# *clauses S* **and**
L: *L* ∈# *C* **and**
tr: *trail S* ⊨*as CNot* (*remove1-mset L C*) **and**
undef: *undefined-lit* (*trail S*) *L* **and**
T: *T* ∼ *cons-trail* (*Propagated L C*) *S*
using *propa* **by** (*elim propagateE*) *auto*

have C': $C \in\#$ *clauses* S'
using SS' C
by (*auto simp*: *state-eq-def clauses-def simp del*: *state-simp*)

show *?thesis*
apply (*rule propagate-rule*[*of* - C])
using *state-eq-sym*[*of* S S'] SS' *conf* C' L *tr undef* TT' T
by (*auto simp*: *state-eq-def simp del*: *state-simp*)
qed

lemma *conflict-state-eq-compatible*:
assumes
confl: *conflict* S T **and**
TT': $T \sim T'$ **and**
SS': $S \sim S'$
shows *conflict* S' T'
proof −
obtain D **where**
conf: *conflicting* S = *None* **and**
D: $D \in\#$ *clauses* S **and**
tr: *trail* S $\models$*as CNot* D **and**
T: $T \sim$ *update-conflicting* (*Some* D) S
using *confl* **by** (*elim conflictE*) *auto*

have D': $D \in\#$ *clauses* S'
using D SS' **by** *fastforce*

show *?thesis*
apply (*rule conflict-rule*[*of* - D])
using *state-eq-sym*[*of* S S'] SS' *conf* D' *tr* TT' T
by (*auto simp*: *state-eq-def simp del*: *state-simp*)
qed

lemma *backtrack-state-eq-compatible*:
assumes
bt: *backtrack* S T **and**
SS': $S \sim S'$ **and**
TT': $T \sim T'$ **and**
inv: $cdcl_W$*-M-level-inv* S
shows *backtrack* S' T'
proof −
obtain D L K i $M1$ $M2$ **where**
conf: *conflicting* S = *Some* D **and**
L: $L \in\#$ D **and**
decomp: (*Decided* K # $M1$, $M2$) $\in$ *set* (*get-all-ann-decomposition* (*trail* S)) **and**
lev: *get-level* (*trail* S) L = *backtrack-lvl* S **and**
max: *get-level* (*trail* S) L = *get-maximum-level* (*trail* S) D **and**
max-D: *get-maximum-level* (*trail* S) (*remove1-mset* L D) $\equiv i$ **and**
lev-K: *get-level* (*trail* S) K = *Suc* i **and**
T: $T \sim$ *cons-trail* (*Propagated* L D)
(*reduce-trail-to* $M1$
(*add-learned-cls* D
(*update-backtrack-lvl* i
(*update-conflicting None* S))))
using *bt inv* **by** (*elim backtrackE*) *metis*
have D': *conflicting* S' = *Some* D

using *SS'* *conf* **by** (*cases conflicting S'*) *auto*

have *T'*: $T' \sim$ *cons-trail* (*Propagated L D*)
(*reduce-trail-to M1* (*add-learned-cls D*
(*update-backtrack-lvl i* (*update-conflicting None S'*))))
using *TT'* **unfolding** *state-eq-def*
using *decomp D' inv SS' T* **by** (*auto simp add*: $cdcl_W$*-M-level-inv-def*)

show *?thesis*
apply (*rule backtrack-rule*[*of - D*])
apply (*rule D'*)
using *state-eq-sym*[*of S S'*] *TT' SS' D' conf L decomp lev max max-D T*
apply (*auto simp*: *state-eq-def simp del*: *state-simp*)[]
using *decomp SS' lev SS' max-D max T' lev-K* **by** (*auto simp*: *state-eq-def simp del*: *state-simp*)
qed

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*)
by (*rule decide-rule*) (*auto simp*: *state-eq-def clauses-def simp del*: *state-simp*)

lemma *skip-state-eq-compatible*:
assumes
skip: *skip S T* **and**
SS': $S \sim S'$ **and**
TT': $T \sim T'$
shows *skip S' T'*
proof −
obtain *L C' M E* **where**
tr: *trail S* = *Propagated L C'* # *M* **and**
raw: *conflicting S* = *Some E* **and**
L: $-L \notin\#$ *E* **and**
E: $E \neq \{\#\}$ **and**
T: $T \sim$ *tl-trail S*
using *skip* **by** (*elim skipE*) *simp*
obtain *E'* **where** *E'*: *conflicting S'* = *Some E'*
using *SS' raw* **by** (*cases conflicting S'*) (*auto simp*: *state-eq-def simp del*: *state-simp*)
show *?thesis*
apply (*rule skip-rule*)
using *tr raw L E T SS'* **apply** (*auto simp*: *simp del*:)[]
using *E'* **apply** *simp*
using *E' SS' L raw E* **apply** (*auto simp*: *state-eq-def simp del*: *state-simp*)[*2*]
using *T TT' SS'* **by** (*auto simp*: *state-eq-def simp del*: *state-simp*)
qed

lemma *resolve-state-eq-compatible*:
assumes
res: *resolve S T* **and**
TT': $T \sim T'$ **and**
SS': $S \sim S'$
shows *resolve S' T'*
proof −

obtain E D L **where**
tr: *trail* $S \neq []$ **and**
hd: *hd-trail* S = *Propagated* L E **and**
L: $L \in\# E$ **and**
raw: *conflicting* S = *Some* D **and**
LD: $-L \in\# D$ **and**
i: *get-maximum-level* (*trail* S) ((*remove1-mset* $(-L)$ D)) = *backtrack-lvl* S **and**
T: $T \sim$ *update-conflicting* (*Some* (*resolve-cls* L D E)) (*tl-trail* S)
using *assms* **by** (*elim resolveE*) *simp*

obtain D' **where**
D': *conflicting* S' = *Some* D'
using SS' *raw* **by** *fastforce*
have [*simp*]: $D = D'$
using D' SS' *raw state-simp*(5) **by** *fastforce*
have $T'T$: $T' \sim T$
using TT' *state-eq-sym* **by** *auto*
show *?thesis*
apply (*rule resolve-rule*)
using *tr* SS' **apply** *simp*
using *hd* SS' **apply** *simp*
using L **apply** *simp*
using D' **apply** *simp*
using D' SS' *raw* LD **apply** (*auto simp add*: *state-eq-def simp del*: *state-simp*)[]
using D' SS' *raw* LD **apply** (*auto simp add*: *state-eq-def simp del*: *state-simp*)[]
using *raw* SS' i **apply** (*auto simp add*: *state-eq-def simp del*: *state-simp*)[]
using T $T'T$ SS' **by** (*auto simp*: *state-eq-def simp del*: *state-simp*)
qed

lemma *forget-state-eq-compatible*:
assumes
forget: *forget* S T **and**
SS': $S \sim S'$ **and**
TT': $T \sim T'$
shows *forget* S' T'
proof –
obtain C **where**
conf: *conflicting* S = *None* **and**
C: $C \in\#$ *learned-clss* S **and**
tr: $\neg$(*trail* S) $\models$*asm clauses* S **and**
C1: $C \notin$ *set* (*get-all-mark-of-propagated* (*trail* S)) **and**
C2: $C \notin\#$ *init-clss* S **and**
T: $T \sim$ *remove-cls* C S
using *forget* **by** (*elim forgetE*) *simp*

show *?thesis*
apply (*rule forget-rule*)
using SS' *conf* **apply** *simp*
using C SS' **apply** *simp*
using SS' *tr* **apply** *simp*
using SS' *C1* **apply** *simp*
using SS' *C2* **apply** *simp*
using T TT' SS' **by** (*auto simp*: *state-eq-def simp del*: *state-simp*)
qed

lemma $cdcl_W$*-state-eq-compatible*:

assumes
cdcl$_W$ *S T* **and** ¬*restart S T* **and**
$S \sim S'$
$T \sim T'$ **and**
cdcl$_W$*-M-level-inv S*
shows *cdcl*$_W$ S' T'
using *assms* **by** (*meson backtrack backtrack-state-eq-compatible bj cdcl*$_W$*.simps cdcl*$_W$*-o-rule-cases cdcl*$_W$*-rf.cases conflict-state-eq-compatible decide decide-state-eq-compatible forget forget-state-eq-compatible propagate-state-eq-compatible resolve resolve-state-eq-compatible skip skip-state-eq-compatible state-eq-ref*)

lemma *cdcl*$_W$*-bj-state-eq-compatible*:
assumes
cdcl$_W$*-bj S T* **and** *cdcl*$_W$*-M-level-inv S*
$T \sim T'$
shows *cdcl*$_W$*-bj S* T'
using *assms* **by** (*meson backtrack backtrack-state-eq-compatible cdcl*$_W$*-bjE resolve resolve-state-eq-compatible skip skip-state-eq-compatible state-eq-ref*)

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*
unfolding *tranclp-unfold-end* **by** (*meson backtrack-state-eq-compatible cdcl*$_W$*-bj.simps resolve-state-eq-compatible rtranclp-unfold skip-state-eq-compatible*)
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*$_W$] *step.hyps*(*1*) *cdcl*$_W$*.other cdcl*$_W$*-o.bj* **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

Conservation of some Properties

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*) (*auto simp*: *inv 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*:
assumes
$cdcl_W$ S T **and**
inv: $cdcl_W$*-M-level-inv* S
shows *init-clss* S = *init-clss* T
using *assms* **by** (*induction rule*: $cdcl_W$*-all-induct*)
(*auto simp*: *inv* $cdcl_W$*-M-level-inv-decomp not-in-iff*)

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*

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.

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))

of Weidenbach's book for the inital state and some additional structural properties about the trail.

lemma $cdcl_W$*-learned-clause-S0-*$cdcl_W$[*simp*]:
$cdcl_W$*-learned-clause* (*init-state* N)
unfolding $cdcl_W$*-learned-clause-def* **by** *auto*

Item 4 page 81 of Weidenbach's book

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*)
case (*backtrack K i M1 M2 L D T*) **note** *decomp* = *this*(*3*) **and** *confl* = *this*(*1*) **and** *lev-K* = *this* (*7*) **and**
undef = *this*(*8*) **and** *T* = *this*(*9*)
show *?case*
using *decomp confl learned undef T lev-K* **unfolding** $cdcl_W$*-learned-clause-def*
by (*auto dest*!: *get-all-ann-decomposition-exists-prepend*
simp: *clauses-def lev-inv* $cdcl_W$*-M-level-inv-decomp dest*: *true-clss-clss-left-right*)
next
case (*resolve L C M D*) **note** *trail* = *this*(*1*) **and** *CL* = *this*(*2*) **and** *confl* = *this*(*4*) **and** *DL* = *this*(*5*)
and *lvl* = *this*(*6*) **and** *T* = *this*(*7*)
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*)
moreover have *remove1-mset* (− *L*) *D* + {#− *L*#} = *D*
using *DL* **by** (*auto simp*: *multiset-eq-iff*)
moreover have *remove1-mset L C* + {#*L*#} = *C*
using *CL* **by** (*auto simp*: *multiset-eq-iff*)
ultimately show *?case*
using *learned T*
by (*auto dest*: *mk-disjoint-insert*
simp add: $cdcl_W$*-learned-clause-def clauses-def*
intro!: *true-clss-cls-union-mset-true-clss-cls-or-not-true-clss-cls-or*[*of - - L*])
next
case (*restart T*)
then show *?case*
using *learned learned-clss-restart-state*[*of T*]
by (*auto*
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*)
next
case *conflict*
then show *?case* **using** *learned*
by (*fastforce simp*: $cdcl_W$*-learned-clause-def clauses-def*
true-clss-clss-in-imp-true-clss-cls)
next
case (*forget U*)
then show *?case* **using** *learned*
by (*auto simp*: $cdcl_W$*-learned-clause-def clauses-def split*: *if-split-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*)

No alien atom in the state

This invariant means that all the literals are in the set of clauses. These properties are implicit in Weidenbach's book.

definition *no-strange-atm* $S' \longleftrightarrow$ (
 ($\forall$ T. *conflicting* S' = *Some* T $\longrightarrow$ *atms-of* T $\subseteq$ *atms-of-mm* (*init-clss* S'))
 $\wedge$ ($\forall$ *L mark*. *Propagated L mark* $\in$ *set* (*trail* S')
 $\longrightarrow$ *atms-of mark* $\subseteq$ *atms-of-mm* (*init-clss* S'))
 $\wedge$ *atms-of-mm* (*learned-clss* S') $\subseteq$ *atms-of-mm* (*init-clss* S')
 $\wedge$ *atm-of* ' (*lits-of-l* (*trail* S')) $\subseteq$ *atms-of-mm* (*init-clss* S'))

lemma *no-strange-atm-decomp*:
 assumes *no-strange-atm S*
 shows *conflicting S* = *Some T* $\Longrightarrow$ *atms-of T* $\subseteq$ *atms-of-mm* (*init-clss S*)
 and ($\forall$ *L mark*. *Propagated L mark* $\in$ *set* (*trail S*)
 $\longrightarrow$ *atms-of mark* $\subseteq$ *atms-of-mm* (*init-clss S*))
 and *atms-of-mm* (*learned-clss S*) $\subseteq$ *atms-of-mm* (*init-clss S*)
 and *atm-of* ' (*lits-of-l* (*trail S*)) $\subseteq$ *atms-of-mm* (*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 *in-atms-of-implies-atm-of-on-atms-of-ms*:
 $C + \{\#L\#\} \in\# A \Longrightarrow x \in$ *atms-of* $C \Longrightarrow x \in$ *atms-of-mm* A
 using *multi-member-split* **by** *fastforce*

lemma *propagate-no-strange-atm-inv*:
 assumes
 propagate S T **and**
 alien: *no-strange-atm S*
 shows *no-strange-atm T*
 using *assms*(*1*)
proof (*induction*)
 case (*propagate-rule C L T*) **note** *confl* = *this*(*1*) **and** *C* = *this*(*2*) **and** *C-L* = *this*(*3*) **and**
 tr = *this*(*4*) **and** *undef* = *this*(*5*) **and** *T* = *this*(*6*)
 have *atm-CL*: *atms-of C* $\subseteq$ *atms-of-mm* (*init-clss S*)
 using *C alien* **unfolding** *no-strange-atm-def*
 by (*auto simp*: *clauses-def atms-of-ms-def*)
 show *?case*
 unfolding *no-strange-atm-def*
 proof (*intro conjI allI impI*, *goal-cases*)
 case *1*
 then show *?case*
 using *confl T undef* **by** *auto*
 next
 case (*2 L' mark'*)
 then show *?case*
 using *C-L T alien undef atm-CL* **unfolding** *no-strange-atm-def clauses-def* **by** (*auto 5 5*)
 next
 case (*3*)
 show *?case* **using** *T alien undef* **unfolding** *no-strange-atm-def* **by** *auto*

next
case (*4*)
show *?case*
using *T alien undef C-L atm-CL* **unfolding** *no-strange-atm-def* **by** (*auto simp*: *atms-of-def*)
qed
qed

lemma *in-atms-of-remove1-mset-in-atms-of*:
$x \in$ *atms-of* (*remove1-mset L C*) $\Longrightarrow x \in$ *atms-of C*
using *in-diffD* **unfolding** *atms-of-def* **by** *fastforce*

lemma $cdcl_W$*-no-strange-atm-explicit*:
assumes
$cdcl_W$ *S S'* **and**
lev: $cdcl_W$*-M-level-inv S* **and**
conf: $\forall$ *T. conflicting S = Some T* $\longrightarrow$ *atms-of T* $\subseteq$ *atms-of-mm* (*init-clss S*) **and**
decided: $\forall$ *L mark. Propagated L mark* $\in$ *set* (*trail S*)
$\longrightarrow$ *atms-of mark* $\subseteq$ *atms-of-mm* (*init-clss S*) **and**
learned: *atms-of-mm* (*learned-clss S*) $\subseteq$ *atms-of-mm* (*init-clss S*) **and**
trail: *atm-of* ' (*lits-of-l* (*trail S*)) $\subseteq$ *atms-of-mm* (*init-clss S*)
shows
($\forall$ *T. conflicting S' = Some T* $\longrightarrow$ *atms-of T* $\subseteq$ *atms-of-mm* (*init-clss S'*)) $\wedge$
($\forall$ *L mark. Propagated L mark* $\in$ *set* (*trail S'*)
$\longrightarrow$ *atms-of mark* $\subseteq$ *atms-of-mm* (*init-clss S'*)) $\wedge$
atms-of-mm (*learned-clss S'*) $\subseteq$ *atms-of-mm* (*init-clss S'*) $\wedge$
atm-of ' (*lits-of-l* (*trail S'*)) $\subseteq$ *atms-of-mm* (*init-clss S'*)
(**is** *?C S'* $\wedge$ *?M S'* $\wedge$ *?U S'* $\wedge$ *?V S'*)
using *assms*(*1*,*2*)
proof (*induct rule*: $cdcl_W$*-all-induct*)
case (*propagate C L T*) **note** *confl* = *this*(*1*) **and** *C-L* = *this*(*2*) **and** *tr* = *this*(*3*) **and** *undef* = *this*(*4*)
and *T* = *this*(*5*)
show *?case*
using *propagate-rule*[*OF propagate.hyps*(*1*−*3*) - *propagate.hyps*(*5*,*6*), *simplified*]
propagate.hyps(*4*) *propagate-no-strange-atm-inv*[*of S T*]
conf decided learned trail **unfolding** *no-strange-atm-def* **by** *presburger*
next
case (*decide L*)
then show *?case* **using** *learned decided conf trail* **unfolding** *clauses-def* **by** *auto*
next
case (*skip L C M D*)
then show *?case* **using** *learned decided conf trail* **by** *auto*
next
case (*conflict D T*) **note** *D-S* = *this*(*2*) **and** *T* = *this*(*4*)
have *D*: *atm-of* ' *set-mset D* $\subseteq \bigcup$(*atms-of* ' (*set-mset* (*clauses S*)))
using *D-S* **by** (*auto simp add*: *atms-of-def atms-of-ms-def*)
moreover {
fix *xa* :: *'v literal*
assume *a1*: *atm-of* ' *set-mset D* $\subseteq$ ($\bigcup$*x*$\in$*set-mset* (*init-clss S*). *atms-of x*)
$\cup$ ($\bigcup$*x*$\in$*set-mset* (*learned-clss S*). *atms-of x*)
assume *a2*:
($\bigcup$*x*$\in$*set-mset* (*learned-clss S*). *atms-of x*) $\subseteq$ ($\bigcup$*x*$\in$*set-mset* (*init-clss S*). *atms-of x*)
assume *xa* $\in\#$ *D*
then have *atm-of xa* $\in$ *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 $\exists$ *m*$\in$*set-mset* (*init-clss S*). *atm-of xa* $\in$ *atms-of m*

by *blast*
} **note** $H = this$
ultimately show *?case* **using** *conflict.prems T learned decided 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 decided conf trail* **by** *auto*
next
case (*forget C T*) **note** $confl = this(1)$ **and** $C = this(4)$ **and** $C\text{-}le = this(5)$ **and** $T = this(6)$
have *H*: $\bigwedge L$ *mark. Propagated L mark* $\in$ *set* (*trail S*) $\Longrightarrow$ *atms-of mark* $\subseteq$ *atms-of-mm* (*init-clss S*)
using *decided* **by** *simp*
show *?case* **unfolding** *clauses-def* **apply** (*intro conjI*)
using *conf confl T trail C* **unfolding** *clauses-def* **apply** (*auto dest*!: *H*)[]
using *T trail C C-le* **apply** (*auto dest*!: *H*)[]
using *T learned C-le atms-of-ms-remove-subset*[*of set-mset* (*learned-clss S*)] **apply** *auto*[]
using *T trail C-le* **apply** (*auto simp*: *clauses-def lits-of-def*)[]
done
next
case (*backtrack L D K i M1 M2 T*) **note** $confl = this(1)$ **and** $LD = this(2)$ **and** $decomp = this(3)$ **and**
$lev\text{-}K = this(7)$ **and** $T = this(8)$
have *?C T*
using *conf T decomp lev lev-K* **by** (*auto simp*: $cdcl_W$*-M-level-inv-decomp*)
moreover have *set M1* $\subseteq$ *set* (*trail S*)
using *decomp* **by** *auto*
then have *M*: *?M T*
using *decided conf confl T decomp lev lev-K*
by (*auto simp*: *image-subset-iff clauses-def* $cdcl_W$*-M-level-inv-decomp*)
moreover have *?U T*
using *learned decomp conf confl T lev lev-K* **unfolding** *clauses-def*
by (*auto simp*: $cdcl_W$*-M-level-inv-decomp*)
moreover have *?V T*
using *M conf confl trail T decomp lev LD lev-K*
by (*auto simp*: $cdcl_W$*-M-level-inv-decomp atms-of-def*
dest!: *get-all-ann-decomposition-exists-prepend*)
ultimately show *?case* **by** *blast*
next
case (*resolve L C M D T*) **note** $trail\text{-}S = this(1)$ **and** $confl = this(4)$ **and** $T = this(7)$
let *?T* = *update-conflicting* (*Some* (*resolve-cls L D C*)) (*tl-trail S*)
have *?C ?T*
using *confl trail-S conf decided* **by** (*auto dest*!: *in-atms-of-remove1-mset-in-atms-of*)
moreover have *?M ?T*
using *confl trail-S conf decided* **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** *simp*
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*)

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 also. Remark that we will show later that there cannot be duplicate *clause*.

definition *distinct-cdcl$_W$-state* (S ::$'st$)
$\longleftrightarrow$ (($\forall$ T. *conflicting* S = *Some* T $\longrightarrow$ *distinct-mset* T)
$\wedge$ *distinct-mset-mset* (*learned-clss* S)
$\wedge$ *distinct-mset-mset* (*init-clss* S)
$\wedge$ ($\forall$ L *mark*. (*Propagated* L *mark* $\in$ *set* (*trail* S) $\longrightarrow$ *distinct-mset mark*)))

lemma *distinct-cdcl$_W$-state-decomp*:
assumes *distinct-cdcl$_W$-state* (S ::$'st$)
shows
$\forall$ T. *conflicting* S = *Some* T $\longrightarrow$ *distinct-mset* T **and**
distinct-mset-mset (*learned-clss* S) **and**
distinct-mset-mset (*init-clss* S) **and**
$\forall$ L *mark*. (*Propagated* L *mark* $\in$ *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 ::$'st$) **and** *conflicting* S = *Some* T
shows *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**
lev-inv: *cdcl$_W$-M-level-inv* S **and**
distinct-cdcl$_W$-state S
shows *distinct-cdcl$_W$-state* S'
using *assms*(*1*,*2*,*2*,*3*)
proof (*induct rule*: *cdcl$_W$-all-induct*)
case (*backtrack* L D K i $M1$ $M2$)
then show *?case*
using *lev-inv* **unfolding** *distinct-cdcl$_W$-state-def*
by (*auto dest*: *get-all-ann-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*: *distinct-cdcl$_W$-state-def distinct-mset-set-def clauses-def*
dest!: *in-diffD*)

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+*

Conflicts and Annotations

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*, *'v clause*) *ann-lits*
assumes
inv: *cdcl$_W$-M-level-inv S* **and**
i: *get-maximum-level* (*trail S*) ((*remove1-mset L D*)) ≡ *i* **and**
decomp: (*Decided K* # *M1*, *M2*)
∈ *set* (*get-all-ann-decomposition* (*trail S*)) **and**
S-lvl: *backtrack-lvl S* = *get-maximum-level* (*trail S*) *D* **and**
S-confl: *conflicting S* = *Some D* **and**
lev-K: *get-level* (*trail S*) *K* = *Suc i* **and**
T: *T* ∼ *cons-trail* (*Propagated L D*)
(*reduce-trail-to M1*
(*add-learned-cls D*
(*update-backtrack-lvl i*
(*update-conflicting None S*)))) **and**
confl: ∀ *T. conflicting S* = *Some T* ⟶ *trail S* ⊨*as CNot T*
shows *atms-of* ((*remove1-mset L D*)) ⊆ *atm-of* ‘ *lits-of-l* (*tl* (*trail T*))
proof (*rule ccontr*)
let *?k* = *get-maximum-level* (*trail S*) *D*
let *?D* = *D*
let *?D'* = (*remove1-mset L D*)
have *trail S* ⊨*as CNot ?D* **using** *confl S-confl* **by** *auto*
then have *vars-of-D*: *atms-of ?D* ⊆ *atm-of* ‘ *lits-of-l* (*trail S*) **unfolding** *atms-of-def*
by (*meson image-subsetI true-annots-CNot-all-atms-defined*)

obtain *M0* **where** *M*: *trail S* = *M0* @ *M2* @ *Decided K* # *M1*
using *decomp* **by** *auto*

```
  have max: ?k = count-decided (M0 @ M2 @ Decided K # 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-l M1
    using T decomp inv by (auto simp: cdcl_W-M-level-inv-decomp)
  then have L'-in: L' ∈ atm-of ‘ lits-of-l (M0 @ M2 @ Decided K # [])
    using vars-of-D unfolding M by (auto dest: in-atms-of-remove1-mset-in-atms-of)
  then obtain L'' where
    L'' ∈# ?D' and
    L'': L' = atm-of L''
    using L' L'-notin-M1 unfolding atms-of-def by auto
  have atm-of K ∉ atm-of ‘ lits-of-l (M0 @ M2)
    using inv by (auto simp: cdcl_W-M-level-inv-def M lits-of-def)
  then have count-decided M1 = i
    using lev-K unfolding M by (auto simp: image-Un)
  then have lev-L'':
    get-level (trail S) L'' = get-level (M0 @ M2 @ Decided K # []) L'' + i
    using L'-notin-M1 L'' get-rev-level-skip-end[OF L'-in[unfolded L''], of M1] M by auto
  moreover
    consider
      (M0) L' ∈ atm-of ‘ lits-of-l M0 |
      (M2) L' ∈ atm-of ‘ lits-of-l M2 |
      (K) L' = atm-of K
      using inv L'-in unfolding L'' by (auto simp: cdcl_W-M-level-inv-def)
    then have get-level (M0 @ M2 @ Decided K # []) L'' ≥ Suc 0
      proof cases
        case M0
        then have L' ≠ atm-of K
          using inv ‹atm-of K ∉ atm-of ‘ lits-of-l (M0 @ M2)› unfolding L'' by auto
        then show ?thesis using M0 unfolding L'' by auto
      next
        case M2
        then have L' ∉ atm-of ‘ lits-of-l (M0 @ Decided K # [])
          using inv ‹atm-of K ∉ atm-of ‘ lits-of-l (M0 @ M2)› unfolding L''
          by (auto simp: M cdcl_W-M-level-inv-def atm-lit-of-set-lits-of-l)
        then show ?thesis using M2 unfolding L'' by (auto simp: image-Un)
      next
        case K
        then have L' ∉ atm-of ‘ lits-of-l (M0 @ M2)
          using inv unfolding L'' by (auto simp: cdcl_W-M-level-inv-def atm-lit-of-set-lits-of-l M)
        then show ?thesis using K unfolding L'' by (auto simp: image-Un)
      qed
  ultimately have get-level (trail S) L'' ≥ i + 1
    using lev-L'' unfolding M by simp
  then have get-maximum-level (trail S) ?D' ≥ i + 1
    using get-maximum-level-ge-get-level[OF ‹L'' ∈# ?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 ⊆ atm-of ‘ lits-of-l M' and
```

a3: $x \in$ *atms-of D*
shows $x \notin$ *atm-of ‘ lits-of-l M*
proof −
have *ff1*: $\bigwedge$*l ms. undefined-lit ms l* $\vee$ *atm-of l*
$\in$ *set* (*map* (λ*m. atm-of* (*lit-of* (*m* ::('*a*, '*b*) *ann-lit*))) *ms*)
by (*simp add*: *defined-lit-map*)
have *ff2*: $\bigwedge$*a. a* $\notin$ *atms-of D* $\vee$ *a* $\in$ *atm-of ‘ lits-of-l M'*
using *a2* **by** (*meson subsetCE*)
have *ff3*: $\bigwedge$*a. a* $\notin$ *set* (*map* (λ*m. atm-of* (*lit-of m*)) *M'*)
$\vee$ *a* $\notin$ *set* (*map* (λ*m. 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 show *x* $\notin$ *atm-of ‘ lits-of-l M*
using *ff3 ff2 ff1 a3* **by** (*metis* (*no-types*) *Decided-Propagated-in-iff-in-lits-of-l*)
qed

Item 5 page 81 of Weidenbach's book

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-ann-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-ann-decomposition* (*trail S'*))
using *assms*(*1*,*2*)
proof (*induct rule*: $cdcl_W$*-all-induct*)
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*(*7*)
let *?decomp* = *get-all-ann-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-ann-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-ann-decomposition M*)
= *insert* (*hd* (*get-all-ann-decomposition M*)) (*set* (*tl* (*get-all-ann-decomposition M*)))
by (*cases get-all-ann-decomposition M*) *auto*
show *?case*
using *decomp tr T* **unfolding** *all-decomposition-implies-def*
by (*cases hd* (*get-all-ann-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** $L = this(3)$ **and** $undef = this(5)$ **and** $T = this(6)$
obtain *a y* **where** *ay*: *hd* (*get-all-ann-decomposition* (*trail S*)) = (*a*, *y*)
by (*cases hd* (*get-all-ann-decomposition* (*trail S*)))
then have *M*: *trail S* = *y* @ *a* **using** *get-all-ann-decomposition-decomp* **by** *blast*
have *M'*: *set* (*get-all-ann-decomposition* (*trail S*))
= *insert* (*a*, *y*) (*set* (*tl* (*get-all-ann-decomposition* (*trail S*))))
using *ay* **by** (*cases get-all-ann-decomposition* (*trail S*)) *auto*
have *unmark-l a* ∪ *set-mset* (*init-clss S*) ⊨*ps unmark-l y*
using *decomp ay* **unfolding** *all-decomposition-implies-def*
by (*cases get-all-ann-decomposition* (*trail S*)) *fastforce+*
then have *a-Un-N-M*: *unmark-l a* ∪ *set-mset* (*init-clss S*)
⊨*ps unmark-l* (*trail S*)
unfolding *M* **by** (*auto simp add*: *all-in-true-clss-clss image-Un*)

have *unmark-l a* ∪ *set-mset* (*init-clss S*) ⊨*p* {#*L*#} (**is** *?I* ⊨*p* -)
proof (*rule true-clss-cls-plus-CNot*)
show *?I* ⊨*p remove1-mset L C* + {#*L*#}
apply (*rule true-clss-clss-in-imp-true-clss-cls*[*of* -
set-mset (*init-clss S*) ∪ *set-mset* (*learned-clss S*)])
using *learned propa L* **by** (*auto simp*: *clauses-def* $cdcl_W$*-learned-clause-def*
true-annot-CNot-diff)
next
have *unmark-l* (*trail S*) ⊨*ps CNot* (*remove1-mset L C*)
using ‹(*trail S*) ⊨*as CNot* (*remove1-mset L C*)› *true-annots-true-clss-clss*
by *blast*
then show *?I* ⊨*ps CNot* (*remove1-mset L 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-ann-decomposition* (*y* @ *a*)).
unmark-l Ls ∪ *set-mset* (*init-clss S*) ⊨*ps unmark-l seen* ⟹
(*aa*, *b*) ∈ *set* (*tl* (*get-all-ann-decomposition* (*y* @ *a*))) ⟹
unmark-l aa ∪ *set-mset* (*init-clss S*) ⊨*ps unmark-l b*
by (*metis* (*no-types*, *lifting*) *case-prod-conv get-all-ann-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-l a* ∪ *set-mset* (*init-clss S*) ⊨*ps unmark-l y*›
ay **by** *auto*
next
case (*backtrack L D K i M1 M2 T*) **note** $conf = this(1)$ **and** $LD = this(2)$ **and** $decomp' = this(3)$
and
$lev\text{-}L = this(4)$ **and** $lev\text{-}K = this(7)$ **and** $undef = this(8)$ **and** $T = this(9)$
let *?D* = *D*
let *?D'* = (*remove1-mset L D*)
have ∀ *l* ∈ *set M2*. ¬*is-decided l*
using *get-all-ann-decomposition-snd-not-decided decomp'* **by** *blast*
obtain *M0* **where** *M*: *trail S* = *M0* @ *M2* @ *Decided K* # *M1*
using *decomp'* **by** *auto*
show *?case* **unfolding** *all-decomposition-implies-def*
proof
fix *x*

assume *x* ∈ *set* (*get-all-ann-decomposition* (*trail T*))
then have *x*: *x* ∈ *set* (*get-all-ann-decomposition* (*Propagated L ?D # M1*))
  using *T decomp′ undef inv* **by** (*simp add*: *cdcl*$_W$*-M-level-inv-decomp*)
let *?m* = *get-all-ann-decomposition* (*Propagated L ?D # M1*)
let *?hd* = *hd ?m*
let *?tl* = *tl ?m*
consider
    (*hd*) *x* = *?hd*
  | (*tl*) *x* ∈ *set ?tl*
  using *x* **by** (*cases ?m*) *auto*
then show *case x of* (*Ls*, *seen*) ⇒ *unmark-l Ls* ∪ *set-mset* (*init-clss T*) ⊨*ps unmark-l seen*
  proof *cases*
    case *tl*
    then have *x* ∈ *set* (*get-all-ann-decomposition* (*trail S*))
      using *tl-get-all-ann-decomposition-skip-some*[*of x*] **by** (*simp add*: *list.set-sel*(*2*) *M*)
    then show *?thesis*
      using *decomp learned decomp confl alien inv T undef M*
      unfolding *all-decomposition-implies-def cdcl*$_W$*-M-level-inv-def*
      by *auto*
  next
    case *hd*
    obtain *M1′ M1″* **where** *M1*: *hd* (*get-all-ann-decomposition M1*) = (*M1′*, *M1″*)
      by (*cases hd* (*get-all-ann-decomposition M1*))
    then have *x′*: *x* = (*M1′*, *Propagated L ?D # M1″*)
      using ‹*x* = *?hd*› **by** *auto*
    have (*M1′*, *M1″*) ∈ *set* (*get-all-ann-decomposition* (*trail S*))
      using *M1*[*symmetric*] *hd-get-all-ann-decomposition-skip-some*[*OF M1*[*symmetric*],
        of M0 @ M2] **unfolding** *M* **by** *fastforce*
    then have *1*: *unmark-l M1′* ∪ *set-mset* (*init-clss S*) ⊨*ps unmark-l M1″*
      using *decomp* **unfolding** *all-decomposition-implies-def* **by** *auto*

    moreover
      have *vars-of-D*: *atms-of ?D′* ⊆ *atm-of* ‘ *lits-of-l M1*
        using *backtrack-atms-of-D-in-M1*[*of S D L i K M1 M2 T*] *backtrack.hyps 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-l* (*M0 @ M2 @ Decided K #* [])
        using *vars-of-D distinct-atms-of-incl-not-in-other*[*of*
          M0 @M2 @ Decided K # [] *M1*] **unfolding** *M* **by** *auto*
      have *trail S* ⊨*as CNot* (*remove1-mset L D*)
        using *conf confl LD* **unfolding** *M true-annots-true-cls-def-iff-negation-in-model*
        by (*auto dest*!: *Multiset.in-diffD*)
      then have *M1* ⊨*as CNot ?D′*
        using *vars-in-M1 true-annots-remove-if-notin-vars*[*of M0 @ M2 @ Decided K #* []
          M1 CNot ?D′] *conf confl* **unfolding** *M lits-of-def* **by** *simp*
      have *M1* = *M1″ @ M1′* **by** (*simp add*: *M1 get-all-ann-decomposition-decomp*)
      have *TT*: *unmark-l 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*]
        unfolding ‹*M1* = *M1″ @ M1′*› **by** (*auto simp add*: *inf-sup-aci*(*5*,*7*))
      have *init-clss S* ⊨*pm ?D′* + {#*L*#}
        using *conf learned confl LD* **unfolding** *cdcl*$_W$*-learned-clause-def* **by** *auto*
      then have *T′*: *unmark-l M1′* ∪ *set-mset* (*init-clss S*) ⊨*p ?D′* + {#*L*#} **by** *auto*
      have *atms-of* (*?D′* + {#*L*#}) ⊆ *atms-of-mm* (*clauses S*)
        using *alien conf LD* **unfolding** *no-strange-atm-def clauses-def* **by** *auto*
      then have *unmark-l M1′* ∪ *set-mset* (*init-clss S*) ⊨*p* {#*L*#}

using *true-clss-cls-plus-CNot*[*OF* T' *TT*] **by** *auto*

ultimately show *?thesis*
using T' *T decomp*$'$ *undef inv* **unfolding** x' **by** (*simp add*: $cdcl_W$*-M-level-inv-decomp*)
qed
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-ann-decomposition* (*trail S*)) **and**
confl: $\forall$ *T*. *conflicting S* = *Some T* $\longrightarrow$ *trail S* $\models$*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*)
case (*propagate C L T*) **note** *LC* = *this*(*3*) **and** *confl* = *this*(*4*) **and** *undef* = *this*(*5*) **and** *T* = *this*(*6*)
show *?case*
proof (*intro allI impI*)
fix L' *mark a b*
assume *a* @ *Propagated L*$'$ *mark* # *b* = *trail T*
then consider
(*hd*) *a* = [] **and** *L* = L' **and** *mark* = *C* **and** *b* = *trail S*
| (*tl*) *tl a* @ *Propagated L*$'$ *mark* # *b* = *trail S*
using *T undef* **by** (*cases a*) *fastforce+*
then show *b* $\models$*as CNot* (*mark* − {#L'#}) $\wedge$ L' $\in$# *mark*
using *mark-confl confl LC* **by** *cases auto*
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* = *Decided L* # *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 $\forall$ *La mark a b*. *a* @ *Propagated La mark* # *b* = *Propagated L C*$'$ # *M*
$\longrightarrow$ *b* $\models$*as CNot* (*mark* − {#*La*#}) $\wedge$ *La* $\in$# *mark*
using *mark-confl* **unfolding** *skip.hyps*(*1*) **by** *simp*
ultimately show *b* $\models$*as CNot* (*mark* − {#L'#}) $\wedge$ L' $\in$# *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*(*7*)

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*#}) # *a*) @ *Propagated* L' *mark* # *b*
      = *Propagated L* (*C* + {#*L*#}) # *M*
      using *T tr-S* **by** *auto*
    then show *b* ⊨*as CNot* (*mark* − {#L'#}) ∧ L' ∈# *mark*
      using *mark-confl* **unfolding** *tr-S* **by** (*metis Cons-eq-appendI list.sel*(*3*))
  qed
next
  case *restart*
  then show *?case* **by** *auto*
next
  case *forget*
  then show *?case* **using** *mark-confl* **by** *auto*
next
  case (*backtrack L D K i M1 M2 T*) **note** *conf* = *this*(*1*) **and** *LD* = *this*(*2*) **and** *decomp* = *this*(*3*)
and
    lev-K = *this*(*7*) **and** *T* = *this*(*8*)
  have ∀ *l* ∈ *set M2*. ¬*is-decided l*
    using *get-all-ann-decomposition-snd-not-decided decomp* **by** *blast*
  obtain *M0* **where** *M*: *trail S* = *M0* @ *M2* @ *Decided K* # *M1*
    using *decomp* **by** *auto*
  have [*simp*]: *trail* (*reduce-trail-to M1* (*add-learned-cls D*
    (*update-backtrack-lvl i* (*update-conflicting None S*)))) = *M1*
    using *decomp lev* **by** (*auto simp*: $cdcl_W$*-M-level-inv-decomp*)
  let *?D* = *D*
  let *?D*′ = (*remove1-mset L D*)
  show *?case*
    proof (*intro allI impI*)
      fix *La* :: ′*v literal* **and** *mark* :: ′*v clause* **and**
        a b :: (′*v*, ′*v clause*) *ann-lits*
      assume *a* @ *Propagated La mark* # *b* = *trail T*
      then consider
          (*hd-tr*) *a* = [] **and**
            (*Propagated La mark* :: (′*v*, ′*v clause*) *ann-lit*) = *Propagated L ?D* **and**
            b = *M1*
        | (*tl-tr*) *tl a* @ *Propagated La mark* # *b* = *M1*
        using *M T decomp lev* **by** (*cases a*) (*auto simp*: $cdcl_W$*-M-level-inv-def*)
      then show *b* ⊨*as CNot* (*mark* − {#*La*#}) ∧ *La* ∈# *mark*
        proof *cases*
          case *hd-tr* **note** *A* = *this*(*1*) **and** *P* = *this*(*2*) **and** *b* = *this*(*3*)
          have *trail S* ⊨*as CNot ?D* **using** *conf confl* **by** *auto*
          then have *vars-of-D*: *atms-of ?D* ⊆ *atm-of* ‘ *lits-of-l* (*trail S*)
            unfolding *atms-of-def*
            by (*meson image-subsetI true-annots-CNot-all-atms-defined*)
          have *vars-of-D*: *atms-of ?D*′ ⊆ *atm-of* ‘ *lits-of-l M1*
            using *backtrack-atms-of-D-in-M1*[*of S D L i K M1 M2 T*] *T backtrack lev confl*
            by (*auto simp*: $cdcl_W$*-M-level-inv-decomp*)
          have *no-dup* (*trail S*) **using** *lev* **by** (*auto simp*: $cdcl_W$*-M-level-inv-decomp*)
          then have ∀ *x* ∈ *atms-of ?D*′. *x* ∉ *atm-of* ‘ *lits-of-l* (*M0* @ *M2* @ *Decided K* # [])
            using *vars-of-D distinct-atms-of-incl-not-in-other*[*of*
              M0 @ *M2* @ *Decided K* # [] *M1*] **unfolding** *M* **by** *auto*
          then have *M1* ⊨*as CNot ?D*′
            using *true-annots-remove-if-notin-vars*[*of M0* @ *M2* @ *Decided K* # []

$M1$ *CNot* $?D'$] ⟨*trail* S ⊨*as CNot* $?D$⟩ **unfolding** M *lits-of-def*
by (*simp add*: *true-annot-CNot-diff*)
then show b ⊨*as CNot* (*mark* − {#*La*#}) ∧ *La* ∈# *mark*
using P *LD* b **by** *auto*
next
case *tl-tr*
then obtain c' **where** c' @ *Propagated La mark* # b = *trail* S
unfolding M **by** *auto*
then show b ⊨*as CNot* (*mark* − {#*La*#}) ∧ *La* ∈# *mark*
using *mark-confl* **by** *auto*
qed
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**
decided-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*)
case (*skip* L C' M D T) **note** *tr-S* = *this*(1) **and** *confl* = *this*(2) **and** *L-D* = *this*(3) **and** T = *this*(5)
let $?D = D$
have D: *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-l* M
using *in-CNot-implies-uminus*(2)[*of* L $?D$ *Propagated* L C' # M]
⟨*Propagated* L C' # M ⊨*as CNot* $?D$⟩ **by** *simp*
then show *False*
by (*metis* (*no-types*, *hide-lams*) *M-lev* $cdcl_W$*-M-level-inv-decomp*(1) *consistent-interp-def*
image-insert insert-iff list.set(2) *lits-of-def ann-lit.sel*(2) *tr-S*)
qed
ultimately show *?case*
using *tr-S confl L-D T* **unfolding** $cdcl_W$*-M-level-inv-def*
by (*auto intro*: *true-annots-CNot-lit-of-notin-skip*)
next
case (*resolve* L C M D T) **note** *tr* = *this*(1) **and** *LC* = *this*(2) **and** *confl* = *this*(4) **and** *LD* = *this*(5)
and T = *this*(7)
let $?C$ = *remove1-mset* L C
let $?D$ = *remove1-mset* (−L) D
show *?case*
proof (*intro allI impI*)
fix T'
have *tl* (*trail* S) ⊨*as CNot* $?C$ **using** *tr decided-confl* **by** *fastforce*
moreover
have *distinct-mset* ($?D$ + {#− L#}) **using** *confl dist LD*
unfolding *distinct-*$cdcl_W$*-state-def* **by** *auto*

```
        then have −L ∉# ?D unfolding distinct-mset-def
          by (meson ‹distinct-mset (?D + {#− L#})› distinct-mset-single-add)
        have M ⊨as CNot ?D
          proof −
            have Propagated L (?C + {#L#}) # M ⊨as CNot ?D ∪ CNot {#− L#}
              using confl tr confl-inv LC by (metis CNot-plus LD insert-DiffM2 option.simps(9))
            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 ⊨as CNot T'
        using tr T by auto
    qed
qed (auto simp: M-lev cdcl_W-M-level-inv-decomp)

lemma cdcl_W-conflicting-decomp:
  assumes cdcl_W-conflicting S
  shows ∀ T. conflicting S = Some T ⟶ trail S ⊨as CNot T
  and ∀ L mark a b. a @ Propagated L mark # b = (trail S)
    ⟶ (b ⊨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 ⊨as CNot T
  using assms unfolding cdcl_W-conflicting-def by blast+

lemma cdcl_W-conflicting-S0-cdcl_W[simp]:
  cdcl_W-conflicting (init-state N)
  unfolding cdcl_W-conflicting-def by auto
```

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-ann-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-ann-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-ann-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-ann-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-ann-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-ann-decomposition* (*trail* (*init-state* N))) **and**
$cdcl_W$*-learned-clause* (*init-state* N) **and**

```
    ∀T. conflicting (init-state N) = Some T ⟶ (trail (init-state N))⊨as CNot T and
    no-strange-atm (init-state N) and
    consistent-interp (lits-of-l (trail (init-state N))) and
    ∀L mark a b. a @ Propagated L mark # b = trail (init-state N) ⟶
     (b ⊨as CNot (mark − {#L#}) ∧ L ∈# mark) and
     distinct-cdcl_W-state (init-state N)
  using assms by auto
```

Item 6 page 81 of Weidenbach's book

```
lemma cdcl_W-only-propagated-vars-unsat:
  assumes
    decided: ∀x ∈ set M. ¬ is-decided x and
    DN: D ∈# clauses S and
    D: M ⊨as CNot D and
    inv: all-decomposition-implies-m N (get-all-ann-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 ¬ unsatisfiable (set-mset N)
  then obtain I where
    I: I ⊨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-mm N ∪ atms-of-mm U = atms-of-mm 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 then have total-over-m I (set-mset (learned-clss S))
    using atm-incl state unfolding no-strange-atm-def total-over-m-def total-over-set-def
    by auto
  moreover have N ⊨psm U using learned-cl state unfolding cdcl_W-learned-clause-def by auto
  ultimately have I-D: I ⊨ D
    using I DN cons state unfolding true-clss-clss-def true-clss-def Ball-def
    by (auto simp add: clauses-def)

  have l0: {unmark L |L. is-decided L ∧ L ∈ set M} = {} using decided by auto
  have atms-of-ms (set-mset N ∪ unmark-l M) = atms-of-mm N
    using atm-incl state unfolding no-strange-atm-def by auto
  then have total-over-m I (set-mset N ∪ unmark-l M)
    using tot unfolding total-over-m-def by auto
  then have I ⊨s unmark-l 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-l M by auto
  {
    fix K
    assume K ∈# D
    then have −K ∈ lits-of-l M
      using D unfolding true-annots-def Ball-def CNot-def true-annot-def true-cls-def true-lit-def
      Bex-def by force
    then have −K ∈ I using IM true-clss-singleton-lit-of-implies-incl lits-of-def by fastforce }
  then have ¬ I ⊨ D using cons unfolding true-cls-def true-lit-def consistent-interp-def by auto
  then show False using I-D by blast
```

qed

Item 5 page 81 of Weidenbach's book

We have actually a much stronger theorem, namely *all-decomposition-implies ?N* (*get-all-ann-decomposition ?M*) $\Longrightarrow$ *?N* $\cup$ {*unmark L* |*L. is-decided L* $\wedge$ *L* $\in$ *set ?M*} $\models$*ps unmark-l ?M*, that show that the only choices we made are decided in the formula

lemma
  assumes *all-decomposition-implies-m N* (*get-all-ann-decomposition M*)
  and $\forall$ *m* $\in$ *set M*. $\neg$*is-decided m*
  shows *set-mset N* $\models$*ps unmark-l M*
proof –
  have *T*: {*unmark L* |*L. is-decided 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

Item 7 page 81 of Weidenbach's book (part 1)

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

Item 7 page 81 of Weidenbach's book (part 2)

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*

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*)

case (*backtrack L D K i M1 M2 T*) **note** *confl* = *this*(*1*)
have *consistent-interp* (*lits-of-l* (*trail S*)) **using** *lev* **by** (*auto simp*: $cdcl_W$*-M-level-inv-decomp*)
moreover
have *trail S* ⊨*as CNot D*
using *conflicting confl* **unfolding** $cdcl_W$*-conflicting-def* **by** *auto*
then have *lits-of-l* (*trail S*) ⊨*s CNot D*
using *true-annots-true-cls* **by** *blast*
ultimately have ¬*tautology D* **using** *consistent-CNot-not-tautology* **by** *blast*
then show *?case* **using** *backtrack no-tauto lev*
by (*auto simp*: $cdcl_W$*-M-level-inv-decomp split*: *if-split-asm*)
next
case *restart*
then show *?case* **using** *learned-clss-restart-state state-eq-learned-clss no-tauto*
by (*metis* (*no-types*, *lifting*) *set-mset-mono subsetCE*)
qed (*auto dest*!: *in-diffD*)

definition *final-*$cdcl_W$*-state* (*S* :: *'st*)
⟷ (*trail S* ⊨*asm init-clss S*
∨ ((∀ *L* ∈ *set* (*trail S*). ¬*is-decided L*) ∧
(∃ *C* ∈# *init-clss S*. *trail S* ⊨*as CNot C*)))

definition *termination-*$cdcl_W$*-state* (*S* :: *'st*)
⟷ (*trail S* ⊨*asm init-clss S*
∨ ((∀ *L* ∈ *atms-of-mm* (*init-clss S*). *L* ∈ *atm-of* ' *lits-of-l* (*trail S*))
∧ (∃ *C* ∈# *init-clss S*. *trail S* ⊨*as CNot C*)))

2.1.4 CDCL Strong Completeness

lemma $cdcl_W$*-can-do-step*:
assumes
consistent-interp (*set M*) **and**
distinct M **and**
atm-of ' (*set M*) ⊆ *atms-of-mm N*
shows ∃ *S*. *rtranclp* $cdcl_W$ (*init-state N*) *S*
∧ *state S* = (*map* (λ*L*. *Decided L*) *M*, *N*, {#}, *length M*, *None*)
using *assms*
proof (*induct M*)
case *Nil*
then show *?case* **apply** − **by** (*rule exI*[*of* - *init-state N*]) *auto*
next
case (*Cons L M*) **note** *IH* = *this*(*1*)
have *consistent-interp* (*set M*) **and** *distinct M* **and** *atm-of* ' *set M* ⊆ *atms-of-mm 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* = (*map* (λ*L*. *Decided L*) *M*, *N*, {#}, *length M*, *None*)
using *IH* **by** *blast*
let $?S_0$ = *incr-lvl* (*cons-trail* (*Decided L*) *S*)
have *undefined-lit* (*map* (λ*L*. *Decided L*) *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* ∈ *atms-of-mm 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-map*)
ultimately have $cdcl_W$ *S* $?S_0$

using $cdcl_W.other[OF\ cdcl_W\text{-}o.decide[OF\ decide\text{-}rule[of\ S\ L\ ?S_0]]]\ S$
by (*auto simp*: *state-eq-def simp del*: *state-simp*)
then have $cdcl_W^{**}$ (*init-state N*) $?S_0$
using *st* **by** *auto*
then show *?case*
using *S undef* **by** (*auto intro*!: *exI*[*of* - $?S_0$] *del*: *simp del*:)
qed

theorem 2.9.11 page 84 of Weidenbach's book

lemma $cdcl_W$*-strong-completeness*:
assumes
MN: *set M* $\models$*sm N* **and**
cons: *consistent-interp* (*set M*) **and**
dist: *distinct M* **and**
atm: *atm-of* ‘ (*set M*) $\subseteq$ *atms-of-mm N*
obtains *S* **where**
state S = (*map* ($\lambda L.$ *Decided L*) *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* = (*map* ($\lambda L.$ *Decided L*) *M*, *N*, {#}, *length M*, *None*)
using $cdcl_W$*-can-do-step*[*OF cons dist atm*] **by** *auto*
have *lits-of-l* (*map* ($\lambda L.$ *Decided L*) *M*) = *set M*
by (*induct M*, *auto*)
then have *map* ($\lambda L.$ *Decided L*) *M* $\models$*asm N* **using** *MN 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

2.1.5 Higher level strategy

The rules described previously do not lead to a conclusive state. We have to add a strategy.

Definition

lemma *tranclp-conflict*:
tranclp conflict S S' $\Longrightarrow$ *conflict S S'*
apply (*induct rule*: *tranclp.induct*)
apply *simp*
by (*metis conflictE conflicting-update-conflicting option.distinct*(*1*) *state-eq-conflicting*)

lemma *tranclp-conflict-iff*[*iff*]:
full1 conflict S S' $\longleftrightarrow$ *conflict S S'*
proof −
have *tranclp conflict S S'* $\Longrightarrow$ *conflict S S'* **by** (*meson tranclp-conflict rtranclpD*)
then show *?thesis* **unfolding** *full1-def*
by (*metis conflict.simps conflicting-update-conflicting option.distinct*(*1*)
state-eq-conflicting tranclp.intros(*1*))
qed

inductive $cdcl_W$*-cp* :: *'st* $\Rightarrow$ *'st* $\Rightarrow$ *bool* **where**
conflict'[*intro*]: *conflict S S'* $\Longrightarrow$ $cdcl_W$*-cp S S'* |

propagate': *propagate* S S' $\Longrightarrow$ $cdcl_W$*-cp* S S'

lemma *rtranclp-*$cdcl_W$*-cp-rtranclp-*$cdcl_W$:
  $cdcl_W$*-cp*** S T $\Longrightarrow$ $cdcl_W{}^{**}$ 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**
    $S \sim S'$ **and**
    $T \sim T'$
  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**
    $S \sim S'$ **and**
    $T \sim T'$
  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* **and** $cdcl_W$*-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 $S \neq$ *None* $\Longrightarrow$ *full* $cdcl_W$*-cp* S S
  unfolding *full-def rtranclp-unfold tranclp-unfold*
  by (*auto simp add*: $cdcl_W$*-cp.simps elim*: *conflictE propagateE*)

lemma *skip-unique*:
  skip S T $\Longrightarrow$ *skip* S T' $\Longrightarrow$ $T \sim T'$
  by (*fastforce simp*: *state-eq-def simp del*: *state-simp elim*: *skipE*)

lemma *resolve-unique*:
  resolve S T $\Longrightarrow$ *resolve* S T' $\Longrightarrow$ $T \sim T'$
  by (*fastforce simp*: *state-eq-def simp del*: *state-simp elim*: *resolveE*)

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 $\Longrightarrow$ $\neg$*conflict T U*
by (*metis conflictE conflicting-update-conflicting option.distinct*(*1*) *state-simp*(*5*))

lemma *no-propagate-after-conflict*:
conflict S T $\Longrightarrow$ $\neg$*propagate T U*
by (*metis conflictE conflicting-update-conflicting option.distinct*(*1*) *propagate.cases state-eq-conflicting*)

lemma *tranclp-cdcl$_W$-cp-propagate-with-conflict-or-not*:
assumes *cdcl$_W$-cp*$^{++}$ *S U*
shows (*propagate*$^{++}$ *S U* $\wedge$ *conflicting U = None*)
$\vee$ ($\exists$ *T D*. *propagate*** *S T* $\wedge$ *conflict T U* $\wedge$ *conflicting U = Some D*)
proof −
have *propagate*$^{++}$ *S U* $\vee$ ($\exists$ *T*. *propagate*** *S T* $\wedge$ *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 elim*!: *propagateE*)
moreover
have $\bigwedge$*T*. *conflict T U* $\Longrightarrow$ $\exists$ *D*. *conflicting U = Some D*
by (*auto elim*!: *conflictE simp*: *state-eq-def simp del*: *state-simp*)
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 elim*: *conflictE propagateE simp*: *state-eq-def simp del*: *state-simp*)
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 *cdcl$_W$-cp*$^{\downarrow}$ *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''*

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 elim*: *conflictE propagateE*)+

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 elim*: *conflictE propagateE*)+

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*
  by (*metis rtranclp-*$cdcl_W$*-cp-rtranclp-*$cdcl_W$ *rtranclp-unfold tranclp-*$cdcl_W$*-consistent-inv*)

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 elim*: *conflictE propagateE*)

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-decided* l)
using *assms* **by** *induction* (*fastforce elim*: *conflictE propagateE*)+

lemma *rtranclp-*$cdcl_W$*-cp-dropWhile-trail'*:
assumes $cdcl_W$*-cp*** S S'
obtains M :: (*'v*, *'v clause*) *ann-lits* **where**
trail S' = M @ *trail* S **and** $\forall l \in$ *set* M. $\neg$*is-decided* 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-decided* l)
using *assms* **by** *induction* (*fastforce elim*: *conflictE propagateE*)+

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-decided* 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-mm* (*init-clss S*))
  shows *length* (*trail S*) $\leq$ *card* (*atms-of-mm* (*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-l* (*trail S*))
    using *assms*(*1*,*3*) **unfolding** *S* **by** (*auto simp add*: *finite-subset*)
  have *length* (*trail S*) = *card* (*atm-of* ‘ *lits-of-l* (*trail S*))
    using *no-dup-length-eq-card-atm-of-lits-of-l 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-mm* (*init-clss S*)) − *length* (*trail S*)
      + (*if conflicting S = None then 1 else 0*)) *S*
    > (λS. *card* (*atms-of-mm* (*init-clss S*)) − *length* (*trail S*)
      + (*if conflicting S = None then 1 else 0*)) *T*
  using *assms*
proof −
  have *length* (*trail T*) $\leq$ *card* (*atms-of-mm* (*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 elim*!: *conflictE propagateE*
    simp del: *state-simp simp*: *state-eq-def*)+
qed

lemma *cdcl$_W$-cp-wf*: *wf* {(*b*, *a*). (*cdcl$_W$-M-level-inv a* $\wedge$ *no-strange-atm a*) $\wedge$ *cdcl$_W$-cp a b*}
  apply (*rule wf-wf-if-measure'*[*of less-than - -*
      (λS. *card* (*atms-of-mm* (*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 (λa *b*. (*cdcl$_W$-M-level-inv a* $\wedge$ *no-strange-atm a*) $\wedge$ *cdcl$_W$-cp a b*)** *S T*
    $\longleftrightarrow$ *cdcl$_W$-cp*** *S T*

(**is** *?I S T* $\longleftrightarrow$ *?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 ($\lambda a\ b.$ ($cdcl_W$*-M-level-inv a* $\wedge$ *no-strange-atm a*) $\wedge$ $cdcl_W$*-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* = $\lambda a.$ ($cdcl_W$*-M-level-inv a* $\wedge$ *no-strange-atm a*)
  obtain *T* **where** *T*: *full* ($\lambda a\ b.$ *?inv a* $\wedge$ $cdcl_W$*-cp a b*) *S T*
    using $cdcl_W$*-cp-wf wf-exists-normal-form*[*of* $\lambda a\ b.$ *?inv a* $\wedge$ $cdcl_W$*-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 *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-mm (init-clss S) − atm-of `lits-of-l (trail S)) arbitrary: S)
  case 0 note card = this(1) and alien = this(2)
  then have atm: atms-of-mm (init-clss S) = atm-of ` lits-of-l (trail S)
    unfolding no-strange-atm-def by auto
  { assume a: ∃S'. conflict S S'
    then obtain S' where S': conflict S S' by metis
    then have ∀S''. ¬cdcl_W-cp S' S''
      by (auto simp: cdcl_W-cp.simps elim!: conflictE propagateE
        simp del: state-simp simp: state-eq-def)
    then have ?case using a S' cdcl_W-cp.conflict' unfolding full-def by blast
  }
  moreover {
    assume a: ∃S'. propagate S S'
    then obtain S' where propagate S S' by blast
    then obtain E L where
      S: conflicting S = None and
      E: E ∈# clauses S and
      LE: L ∈# E and
      tr: trail S |=as CNot (E − {#L#}) and
      undef: undefined-lit (trail S) L and
      S': S' ∼ cons-trail (Propagated L E) S
      by (elim propagateE) simp
    have atms-of-mm (learned-clss S) ⊆ atms-of-mm (init-clss S)
      using alien S unfolding no-strange-atm-def by auto
    then have atm-of L ∈ atms-of-mm (init-clss S)
      using E LE S undef unfolding clauses-def by (force simp: in-implies-atm-of-on-atms-of-ms)
    then have False using undef S unfolding atm unfolding lits-of-def
      by (auto simp add: defined-lit-map)
  }
  ultimately show ?case unfolding full-def by (metis cdcl_W-cp.cases rtranclp.rtrancl-refl)
next
  case (Suc n) note IH = this(1) and card = this(2) and alien = this(3)
  { assume a: ∃S'. conflict S S'
    then obtain S' where S': conflict S S' by metis
    then have ∀S''. ¬cdcl_W-cp S' S''
      by (auto simp: cdcl_W-cp.simps elim!: conflictE propagateE
        simp del: state-simp simp: state-eq-def)
    then have ?case unfolding full-def Ex-def using S' cdcl_W-cp.conflict' by blast
  }
  moreover {
    assume a: ∃S'. propagate S S'
    then obtain S' where propagate: propagate S S' by blast
    then obtain E L where
      S: conflicting S = None and
      E: E ∈# clauses S and
      LE: L ∈# E and
      tr: trail S |=as CNot (E − {#L#}) and
      undef: undefined-lit (trail S) L and
      S': S' ∼ cons-trail (Propagated L E) S
      by (elim propagateE) simp
    then have atm-of L ∉ atm-of ` lits-of-l (trail S)
      unfolding lits-of-def by (auto simp add: defined-lit-map)
    moreover
      have no-strange-atm S' using alien propagate propagate-no-strange-atm-inv by blast
```

then have *atm-of L* ∈ *atms-of-mm* (*init-clss S*)
using *S′ LE E undef* **unfolding** *no-strange-atm-def*
by (*auto simp*: *clauses-def in-implies-atm-of-on-atms-of-ms*)
then have $\bigwedge A$. {*atm-of L*} ⊆ *atms-of-mm* (*init-clss S*) − *A* ∨ *atm-of L* ∈ *A* **by** *force*
moreover have *Suc n* − *card* {*atm-of L*} = *n* **by** *simp*
moreover have *card* (*atms-of-mm* (*init-clss S*) − *atm-of* ‘ *lits-of-l* (*trail S*)) = *Suc n*
using *card S S′* **by** *simp*
ultimately
have *card* (*atms-of-mm* (*init-clss S*) − *atm-of* ‘ *insert L* (*lits-of-l* (*trail S*))) = *n*
by (*metis* (*no-types*) *Diff-insert card-Diff-subset finite.emptyI finite.insertI image-insert*)
then have *n* = *card* (*atms-of-mm* (*init-clss S′*) − *atm-of* ‘ *lits-of-l* (*trail S′*))
using *card S S′ undef* **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

Literal of highest level in conflicting clauses

One important property of the $cdcl_W$ 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*
proof (*clarify*)
fix *D*
assume *D* ∈# *local.clauses S* **and** *conflicting S* = *None* **and** *trail S* ⊨*as CNot D*
then show *False*
using *conflict-rule*[*of S D update-conflicting* (*Some D*) *S*] *assms*
by *auto*
qed

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** *auto*

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** *auto*

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 elim*: *conflictE*)
next
case *propagate'*
then show *?case* **by** (*auto elim*: *propagateE*)
qed

lemma *no-chained-conflict*:
assumes *conflict* S S' **and** *conflict* S' S''
shows *False*
using *assms* **unfolding** *conflict.simps*
by (*metis conflicting-update-conflicting option.distinct*(*1*) *state-eq-conflicting*)

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 elim*: *conflictE*)

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* | (*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 *full* **unfolding** *full-def*
apply (*metis* (*no-types*) *rtranclpD tranclp-cdcl*$_W$*-cp-no-more-init-clss*)
by (*metis* (*mono-tags*, *lifting*) *full 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 clause option*)
by (*auto elim*: *propagateE*)
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
obtain *D′* **where**
conflicting T = None **and**
D′: *D′* $\in$# *clauses T* **and**
tr: *trail T* $\models$*as CNot* (*D′*) **and**
U: *U* $\sim$ *update-conflicting* (*Some* (*D′*)) *T*
using *TU* **by** (*auto elim*!: *conflictE*)
have *init-clss T = init-clss S* **and** *learned-clss T = learned-clss S*
using *U* ‹*init-clss U = init-clss S*› ‹*learned-clss U = learned-clss S*› **by** *auto*
then have *D* $\in$# *clauses S*
using *confl U D′* **by** (*auto simp*: *clauses-def*)

then have ¬ *trail S* $\models$*as CNot D*
  using *cls-f CT* **by** *simp*

moreover
  obtain *M* **where** *tr-U*: *trail U* = *M* @ *trail S* **and** *nm*: ∀ *m*∈*set M*. ¬*is-decided m*
    by (*metis* (*mono-tags*, *lifting*) *assms*(*1*) *full-def rtranclp-cdcl*$_W$*-cp-dropWhile-trail*)
  have *trail U* $\models$*as CNot D*
    using *tr confl U* **by** (*auto elim*!: *conflictE*)
ultimately obtain *L* **where** *L* ∈# *D* **and** −*L* ∈ *lits-of-l M*
  unfolding *tr-U CNot-def true-annots-def Ball-def true-annot-def true-cls-def* **by** *force*

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*, ′*v clause*) *ann-lit* **and**
      xb :: (′*v*, ′*v clause*) *ann-lit*
    assume *a1*: *atm-of L* = *atm-of* (*lit-of xb*)
    moreover assume *a2*: − *L* = *lit-of x*
    moreover assume *a3*: (λ*l*. *atm-of* (*lit-of l*)) ‘ *set M*
      ∩ (λ*l*. *atm-of* (*lit-of l*)) ‘ *set* (*trail S*) = {}
    moreover assume *a4*: *x* ∈ *set M*
    moreover assume *a5*: *xb* ∈ *set* (*trail S*)
    moreover have *atm-of* (− *L*) = *atm-of L*
      by *auto*
    ultimately have *False*
      by *auto*
   }
  then have *LS*: *atm-of L* ∉ *atm-of* ‘ *lits-of-l* (*trail S*)
    using ⟨−*L* ∈ *lits-of-l 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 count-decided* (*trail S*) ≠ *0*, *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-level M L* = *0*
      using *nm* **by** *auto*
    ultimately show *?thesis* **using** *LS ne US* **unfolding** *tr-U*
      by (*simp add*: *lits-of-def filter-empty-conv*)
  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 *count-decided* (*trail S*) = *backtrack-lvl S*
      using *ne lev* **unfolding** *cdcl*$_W$*-M-level-inv-def* **by** *auto*
    moreover have *atm-of L* ∈ *atm-of* ‘ *lits-of-l M*
      using ⟨−*L* ∈ *lits-of-l 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 get-level-skip-in-all-not-decided*[*of M L*] **unfolding** *lits-of-def US tr-U*

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

Literal of highest level in decided literals

definition *mark-is-false-with-level* :: *'st* ⇒ *bool* **where**
mark-is-false-with-level S' ≡
$\forall D\ M1\ M2\ L.\ M1 \ @\ Propagated\ L\ D\ \#\ M2 = trail\ S' \longrightarrow D - \{\#L\#\} \neq \{\#\}$
$\longrightarrow (\exists L.\ L \in\# D \wedge get\text{-}level\ (trail\ S')\ L = count\text{-}decided\ M1)$

definition *no-more-propagation-to-do* :: *'st* ⇒ *bool* **where**
no-more-propagation-to-do S ≡
$\forall D\ M\ M'\ L.\ D + \{\#L\#\} \in\# clauses\ S \longrightarrow trail\ S = M' \ @\ M \longrightarrow M \models as\ CNot\ D$
$\longrightarrow undefined\text{-}lit\ M\ L \longrightarrow count\text{-}decided\ M < backtrack\text{-}lvl\ S$
$\longrightarrow (\exists L.\ L \in\# D \wedge get\text{-}level\ (trail\ S)\ L = count\text{-}decided\ M)$

lemma *propagate-no-more-propagation-to-do*:
assumes *propagate*: *propagate S S'*
and *H*: *no-more-propagation-to-do S*
and *lev-inv*: $cdcl_W$*-M-level-inv S*
shows *no-more-propagation-to-do S'*
using *assms*
proof −
obtain *E L* **where**
S: *conflicting S = None* **and**
E: $E \in\# clauses\ S$ **and**
LE: $L \in\# E$ **and**
tr: $trail\ S \models as\ CNot\ (E - \{\#L\#\})$ **and**
undefL: *undefined-lit* (*trail S*) *L* **and**
S': $S' \sim cons\text{-}trail\ (Propagated\ L\ E)\ S$
using *propagate* **by** (*elim propagateE*) *simp*
let *?M'* = *Propagated L E # trail S*
show *?thesis* **unfolding** *no-more-propagation-to-do-def*
proof (*intro allI impI*)
fix *D M1 M2 L'*
assume
D-L: $D + \{\#L'\#\} \in\# clauses\ S'$ **and**
trail S' = *M2* @ *M1* **and**
get-max: *count-decided M1* < *backtrack-lvl S'* **and**
$M1 \models as\ CNot\ D$ **and**
undef: *undefined-lit M1 L'*
have *tl M2* @ *M1* = *trail S* ∨ (*M2* = [] ∧ *M1* = *Propagated L E # trail S*)
using ‹*trail S'* = *M2* @ *M1*› *S' S undefL lev-inv*
by (*cases M2*) (*auto simp*:$cdcl_W$*-M-level-inv-decomp*)
moreover {
assume *tl M2* @ *M1* = *trail S*
moreover have $D + \{\#L'\#\} \in\# clauses\ S$
using *D-L S S' undefL* **unfolding** *clauses-def* **by** *auto*
moreover have *count-decided M1* < *backtrack-lvl S*
using *get-max S S' undefL* **by** *auto*
ultimately obtain *L'* **where** $L' \in\# D$ **and**
get-level (*trail S*) *L'* = *count-decided 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 lev-inv cdcl_W.propagate[OF propagate] by blast
          then have no-dup ?M' using S' undefL unfolding cdcl_W-M-level-inv-def by auto
          moreover
            have atm-of L' ∈ atm-of ` (lits-of-l 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-l (trail S))
              using ‹tl M2 @ M1 = trail S›[symmetric] S undefL 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' = count-decided M1
        using S S' undefL by auto
    }
    moreover {
      assume M2 = [] and M1: M1 = Propagated L E # trail S
      have cdcl_W-M-level-inv S'
        using cdcl_W-consistent-inv[OF - lev-inv] cdcl_W.propagate[OF propagate] by blast
      then have count-decided M1 = backtrack-lvl S'
        using S' M1 undefL unfolding cdcl_W-M-level-inv-def by (auto intro: Max-eqI)
      then have False using get-max by auto
    }
    ultimately show ∃ L. L ∈# D ∧ get-level (trail S') L = count-decided 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 by (force elim!: conflictE)

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: *conflicting S = Some E* **and**
LE: *L* ∈# *E* **and**
L: *get-level* (*trail S*) *L = backtrack-lvl S* **and**
D: *get-maximum-level* (*trail S*) (*remove1-mset L E*) < *backtrack-lvl S* **and**
bt: *backtrack-lvl S = get-maximum-level* (*trail S*) *E* **and**
M-L: $cdcl_W$*-M-level-inv S*
shows $\exists S'$. $cdcl_W$*-o S S'*
proof −
have *L-D*: *get-level* (*trail S*) *L = get-maximum-level* (*trail S*) *E*
using *L D bt* **by** (*simp add*: *get-maximum-level-plus*)
let *?i = get-maximum-level* (*trail S*) (*remove1-mset L E*)
obtain *K M1 M2* **where**
K: (*Decided K # M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail S*)) **and**
lev-K: *get-level* (*trail S*) *K = Suc ?i*
using *backtrack-ex-decomp*[*OF M-L*, *of ?i*] *D S* **by** *auto*
show *?thesis* **using** *backtrack-rule*[*OF S LE K L*, *of ?i*] *bt L lev-K bj* **by** (*auto simp*: $cdcl_W$*-bj.simps*)
qed

lemma $cdcl_W$*-stgy-final-state-conclusive*:
assumes
termi: $\forall S'$. ¬$cdcl_W$*-stgy S S'* **and**
decomp: *all-decomposition-implies-m* (*init-clss S*) (*get-all-ann-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*
consider
(*None*) *conflicting S = None*
| (*Some-Empty*) *E* **where** *conflicting S = Some E* **and** *E* = {#}
| (*Some*) *E'* **where** *conflicting S = Some E'* **and**
conflicting S = Some (*E'*) **and** *E'* ≠ {#}
by (*cases conflicting S*, *simp*) *auto*
then show *?thesis*
proof *cases*

```
  case (Some-Empty E)
  then have conflicting S = Some {#} by auto
  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)
  then show ?thesis using Some-Empty by auto
next
  case None
  { assume ¬?M ⊨asm ?N
    have atm-of ` (lits-of-l ?M) = atms-of-mm ?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)
            moreover have conflicting S = None
              using None by auto
            ultimately have ∃S'. cdcl_W-o S S'
              using cdcl_W-o.decide decide-rule ‹l ∈ ?B› no-strange-atm-def
              by (metis 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-l ?M)
      using ‹D ∈# ?N› unfolding ‹atm-of ` (lits-of-l ?M) = atms-of-mm ?N› atms-of-ms-def
      by (auto simp add: atms-of-def)
    then have a1: atm-of ` set-mset D ⊆ atm-of ` lits-of-l (trail S)
      by (auto simp add: atms-of-def lits-of-def)
    have total-over-m (lits-of-l ?M) {D}
      using ‹atms-of D ⊆ atm-of ` (lits-of-l ?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› 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
  then show ?thesis
    using None by auto
```

next
  case (*Some E′*) **note** *conf* = *this*(*1*) **and** *LD* = *this*(*2*) **and** *nempty* = *this*(*3*)
  then obtain *L D* **where**
    E′[*simp*]: *E′* = *D* + {#*L*#} **and**
    lev-L: *get-level ?M L* = *?k*
    by (*metis* (*mono-tags*) *confl-k insert-DiffM2*)
  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*

  have *g-k*: *get-maximum-level* (*trail S*) *D* ≤ *?k*
    using *count-decided-ge-get-maximum-level*[*of ?M*] *level-inv*
    unfolding $cdcl_W$*-M-level-inv-def*
    by *auto*
  {
    assume *decided*: *is-decided* (*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′* **where** *L′*: *hd ?M* = *Decided L′* **using** *decided* **by** (*cases hd ?M*) *auto*
    have ∗: ⋀*list. no-dup list* ⟹
          − *L* ∈ *lits-of-l list* ⟹ *atm-of L* ∈ *atm-of* ‘ *lits-of-l list*
      by (*metis atm-of-uminus imageI*)
    have *L′-L*: *L′* = −*L*
      proof (*rule ccontr*)
        assume ¬ *?thesis*
        moreover have −*L* ∈ *lits-of-l ?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** *consistent-interp-def*
          by (*subst* (*asm*) (*2*) *M*) (*auto simp add*: *atm-of-eq-atm-of L′*)
        moreover
          have *count-decided* (*trail S*) = *?k*
            using *level-inv* **unfolding** $cdcl_W$*-M-level-inv-def* **by** *auto*
          then have *count*: *count-decided* (*tl* (*trail S*)) = *?k* − *1*
            using *level-inv* **unfolding** $cdcl_W$*-M-level-inv-def*
            by (*subst* (*asm*) *M*) (*auto simp add*: *L′*)
          then have *get-level* (*tl ?M*) *L* < *?k*
            using *count-decided-ge-get-level*[*of L tl ?M*] **unfolding** *count k′*[*symmetric*]
            by *auto*
        finally show *False* **using** *lev-L M* **by** *auto*
      qed
    have *L*: *hd ?M* = *Decided* (−*L*) **using** *L′-L L′* **by** *auto*

    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″* ∈# *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* ≠ *L″* **using** *no-dup* ‹*L″* ∈# *D*›
          unfolding *distinct-*$cdcl_W$*-state-def LD*
          by (*metis E′ add.right-neutral add-diff-cancel-right′*
            distinct-mem-diff-mset union-commute 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* ≠ *L''*› *get-level-skip-beginning[of L'' hd ?M tl ?M]* **unfolding** *L*
by (*auto simp*: *atm-of-eq-atm-of*)
moreover
have *d*: *dropWhile* (*λS. atm-of* (*lit-of S*) ≠ *atm-of L*) (*tl* (*trail S*)) = []
using *level-inv* **unfolding** *cdcl*$_W$*-M-level-inv-def* **apply** (*subst* (*asm*)(*2*) *M*)
by (*auto simp*: *image-iff L' L'-L*)
have *get-level* (*tl* (*trail S*)) *L = 0*
by (*auto simp*: *filter-empty-conv d*)
moreover
have *get-level* (*tl* (*trail S*)) *L''* ≤ *count-decided* (*tl* (*trail S*))
by *auto*
then have *get-level* (*tl* (*trail S*)) *L''* < *backtrack-lvl S*
using *level-inv* **unfolding** *cdcl*$_W$*-M-level-inv-def* **apply** (*subst* (*asm*)(*5*) *M*)
by (*auto simp*: *image-iff L' L'-L simp del*: *count-decided-ge-get-level*)
ultimately show *False*
apply −
apply (*subst* (*asm*) *M*, *subst* (*asm*)(*3*) *M*, *subst* (*asm*) *L'*)
using *L-k*
apply (*auto simp*: *L' L'-L split*: *if-splits*)
apply (*subst* (*asm*)(*3*) *M*, *subst* (*asm*) *L'*)
using ‹*L''* ≠ − *L*› **by** (*auto simp*: *L' L'-L split*: *if-splits*)
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-l ?M*)
using *level-inv* **unfolding** *cdcl*$_W$*-M-level-inv-def* **by** *auto*
then have ¬*?M* ⊨*as CNot ?D*
using *taut* **by** (*metis* ‹*L''* = − *L*› ‹*L''* ∈# *D*› *add.commute consistent-interp-def diff-union-cancelR in-CNot-implies-uminus*(*2*) *in-diffD 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 note *H = this*
have *get-maximum-level* (*trail S*) *D* < *get-maximum-level* (*trail S*) (*D* + {#*L*#})
using *H* **by** (*auto simp*: *get-maximum-level-plus lev-L max-def*)
moreover have *backtrack-lvl S = get-maximum-level* (*trail S*) (*D* + {#*L*#})
using *H* **by** (*auto simp*: *get-maximum-level-plus lev-L max-def*)
ultimately have *False*
using *backtrack-no-decomp[OF conf - lev-L] level-inv termi*
cdcl$_W$*-then-exists-cdcl*$_W$*-stgy-step[of S] alien* **unfolding** *E'*
by (*auto simp add*: *lev-L max-def*)
} **note** *not-is-decided = this*

moreover {
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*
assume *nm*: ¬*is-decided* (*hd ?M*)
then obtain *L' C* **where** *L'C*: *hd-trail S = Propagated L' C* **using** ‹*trail S* ≠ []›
by (*cases hd-trail S*) *auto*
then have *hd ?M = Propagated L' C*

using ‹*trail S* $\neq$ []› **by** *fastforce*
then have *M*: *?M* = *Propagated L' C* # *tl ?M*
using ‹*?M* $\neq$ []› *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 $-L' \notin\#$ *?D*
then have *Ex* (*skip S*)
using *skip-rule*[*OF M conf*] **unfolding** *E'* **by** *auto*
then have *False*
using cdcl_W*-then-exists-*cdcl_W*-stgy-step*[*of S*] *alien level-inv termi*
by (*auto dest*: cdcl_W*-o.intros* cdcl_W*-bj.intros*)
}
moreover {
assume *L'D*: $-L' \in\#$ *?D*
then obtain *D'* **where** *D'*: *?D* = *D'* + {#$-$*L'*#} **by** (*metis insert-DiffM2*)
then have *get-maximum-level* (*trail S*) *D'* $\leq$ *?k*
using *count-decided-ge-get-maximum-level*[*of Propagated L' C* # *tl ?M*] *M*
level-inv **unfolding** cdcl_W*-M-level-inv-def* **by** *auto*
then have *get-maximum-level* (*trail S*) *D'* = *?k*
$\vee$ *get-maximum-level* (*trail S*) *D'* < *?k*
using *le-neq-implies-less* **by** *blast*
moreover {
assume *g-D'-k*: *get-maximum-level* (*trail S*) *D'* = *?k*
then have *f1*: *get-maximum-level* (*trail S*) *D'* = *backtrack-lvl S*
using *M* **by** *auto*
then have *Ex* (cdcl_W*-o S*)
using *f1 resolve-rule*[*of S L' C* , *OF* ‹*trail S* $\neq$ []› - - *conf*] *conf g-D'-k*
L'C L'D **unfolding** *C' D' E'*
by (*fastforce simp add*: *D' intro*: cdcl_W*-o.intros* cdcl_W*-bj.intros*)
then have *False*
by (*meson alien* cdcl_W*-then-exists-*cdcl_W*-stgy-step termi level-inv*)
}
moreover {
assume *a1*: *get-maximum-level* (*trail S*) *D'* < *?k*
then have *f3*: *get-maximum-level* (*trail S*) *D'* < *get-level* (*trail S*) ($-$*L'*)
using *a1 lev-L* **by** (*metis D' get-maximum-level-ge-get-level insert-noteq-member*
not-less)
moreover have *backtrack-lvl S* = *get-level* (*trail S*) *L'*
apply (*subst M*)
using *level-inv* **unfolding** cdcl_W*-M-level-inv-def*
by (*subst* (*asm*)(*3*) *M*) (*auto simp add*: cdcl_W*-M-level-inv-decomp*)[]
moreover
then have *get-level* (*trail S*) *L'* = *get-maximum-level* (*trail S*) (*D'* + {#$-$ *L'*#})
using *a1* **by** (*auto simp add*: *get-maximum-level-plus max-def*)
ultimately have *False*
using *M backtrack-no-decomp*[*of S* - $-$*L'*, *OF conf*]
cdcl_W*-then-exists-*cdcl_W*-stgy-step L'D level-inv termi alien*
unfolding *D' E'* **by** *auto*
}
ultimately have *False* **by** *blast*
}
ultimately have *False* **by** *blast*
}
ultimately show *?thesis* **by** *blast*
qed
qed

lemma $cdcl_W$*-cp-tranclp-*$cdcl_W$:
$cdcl_W$*-cp* S $S' \Longrightarrow cdcl_W{}^{++}$ S S'
apply (*induct rule*: $cdcl_W$*-cp.induct*)
by (*meson* $cdcl_W$*.conflict* $cdcl_W$*.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 $S' = S'' \lor cdcl_W\text{-}cp^{++}\ S'\ S''$
by (*simp add*: *rtranclp-unfold full-def*)
then show *?case*
using *other'* **by** (*meson* $cdcl_W$*.other 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 *not-empty-get-maximum-level-exists-lit*:
assumes *n*: $D \neq \{\#\}$
and *max*: *get-maximum-level* M $D = n$
shows $\exists L \in\# D.$ *get-level* M $L = n$
proof −
have *f*: *finite* (*insert 0* (($\lambda L.$ *get-level* M L) ' *set-mset* D)) **by** *auto*
then have $n \in$ (($\lambda L.$ *get-level* M L) ' *set-mset* D)
using *n max get-maximum-level-exists-lit-of-max-level image-iff*
unfolding *get-maximum-level-def* **by** *force*
then show $\exists L \in\# D.$ *get-level* M $L = n$ **by** *auto*
qed

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*)
case (*resolve L C M D T*) **note** *tr-S = this(1)* **and** *confl = this(4)* **and** *LD = this(5)* **and** *T = this(7)*
have *uL-not-D*: $-L \notin\#$ *remove1-mset* $(-L)$ *D*
using *n-d confl* **unfolding** *distinct-$cdcl_W$-state-def distinct-mset-def*
by (*metis distinct-$cdcl_W$-state-def distinct-mem-diff-mset multi-member-last n-d*)
moreover have *L-not-D*: $L \notin\#$ *remove1-mset* $(-L)$ *D*
proof (*rule ccontr*)
assume ¬ *?thesis*
then have $L \in\# D$
by (*auto simp*: *in-remove1-mset-neq*)
moreover have *Propagated L C # M* $\models$*as CNot D*
using *conflicting confl tr-S* **unfolding** *$cdcl_W$-conflicting-def* **by** *auto*
ultimately have $-L \in$ *lits-of-l* (*Propagated L C # M*)
using *in-CNot-implies-uminus(2)* **by** *blast*
moreover have *no-dup* (*Propagated L C # 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-lit.sel(2) distinct-consistent-interp*)
qed

ultimately
have *g-D*: *get-maximum-level* (*Propagated L C # M*) (*remove1-mset* $(-L)$ *D*)
= *get-maximum-level M* (*remove1-mset* $(-L)$ *D*)
using *get-maximum-level-skip-first*[*of L remove1-mset* $(-L)$ *D C M*]
by (*simp add*: *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set atms-of-def*)
have *lev-L*[*simp*]: *get-level M L = 0*
apply (*rule atm-of-notin-get-rev-level-eq-0*)
using *lev* **unfolding** *$cdcl_W$-M-level-inv-def tr-S* **by** (*auto simp*: *lits-of-def*)

have *D*: *get-maximum-level M* (*remove1-mset* $(-L)$ *D*) = *backtrack-lvl S*
using *resolve.hyps(6) LD* **unfolding** *tr-S* **by** (*auto simp*: *get-maximum-level-plus max-def g-D*)
have *get-maximum-level M* (*remove1-mset L C*) $\leq$ *backtrack-lvl S*
using *count-decided-ge-get-maximum-level*[*of M*] *lev* **unfolding** *tr-S $cdcl_W$-M-level-inv-def* **by** *auto*
then have
get-maximum-level M (*remove1-mset* $(- L)$ *D* #∪ *remove1-mset L C*) =
backtrack-lvl S
by (*auto simp*: *get-maximum-level-union-mset get-maximum-level-plus max-def D*)
then show *?case*
using *tr-S not-empty-get-maximum-level-exists-lit*[*of*
remove1-mset $(- L)$ *D* #∪ *remove1-mset L C M*] *T*
by *auto*
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**
$La \in\# D$ **and**
get-level (*Propagated L C' # M*) *La = backtrack-lvl S*
using *skip confl-inv* **by** *auto*
moreover
have *atm-of La* $\neq$ *atm-of L*
proof (*rule ccontr*)
assume ¬ *?thesis*
then have *La*: *La = L* **using** ‹$La \in\# D$› ‹$- L \notin\# D$›
by (*auto simp add*: *atm-of-eq-atm-of*)

```
        have Propagated L C' # M ⊨as CNot D
          using conflicting tr-S D unfolding cdcl_W-conflicting-def by auto
        then have −L ∈ lits-of-l M
          using ‹La ∈# D› in-CNot-implies-uminus(2)[of L D
            Propagated L C' # M] unfolding La
          by auto
        then show False using lev tr-S unfolding cdcl_W-M-level-inv-def consistent-interp-def by auto
      qed
    then have get-level (Propagated L C' # M) La = get-level M La by auto
  ultimately show ?case using D tr-S T by auto
next
  case backtrack
  then show ?case
    by (auto split: if-split-asm simp: cdcl_W-M-level-inv-decomp lev)
qed auto
```

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 propagate-high-levelE:
  assumes propagate S T
  obtains M' N' U k L C where
    state S = (M', N', U, k, None) and
    state T = (Propagated L (C + {#L#}) # M', N', U, k, None) and
    C + {#L#} ∈# local.clauses S and
    M' ⊨as CNot C and
    undefined-lit (trail S) L
proof −
  obtain E L where
    conf: conflicting S = None and
    E: E ∈# clauses S and
    LE: L ∈# E and
    tr: trail S ⊨as CNot (E − {#L#}) and
    undef: undefined-lit (trail S) L and
    T: T ∼ cons-trail (Propagated L E) S
    using assms by (elim propagateE) simp
  obtain M N U k where
    S: state S = (M, N, U, k, None)
    using conf by auto
  show thesis
    using that[of M N U k L remove1-mset L E] S T LE E tr undef
    by auto
qed

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-l (*trail S*) $\subseteq$ *set M* **and**
init-clss S = *N* **and**
propagate** *S S'* **and**
learned-clss S = {#}
shows *length* (*trail S*) $\leq$ *length* (*trail S'*) $\wedge$ *lits-of-l* (*trail S'*) $\subseteq$ *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*) $\leq$ *length* (*trail Y*) **and** *LM*: *lits-of-l* (*trail Y*) $\subseteq$ *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*#} $\in$# *clauses Y* **and**
M'-C: *M'* $\models$*as CNot C* **and**
undefined-lit (*trail Y*) *L*
using *propa* **by** (*auto elim*: *propagate-high-levelE*)
have *init-clss S* = *init-clss Y*
using *st* **by** *induction* (*auto elim*: *propagateE*)
then have [*simp*]: *N'* = *N* **using** *NS Y Z* **by** *simp*
have *learned-clss Y* = {#}
using *st learned* **by** *induction* (*auto elim*: *propagateE*)
then have [*simp*]: *U* = {#} **using** *Y* **by** *auto*
have *set M* $\models$*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* $\models$ *C* + {#*L*#}
using *MN C learned Y NS* ‹*init-clss S* = *init-clss Y*› ‹*learned-clss Y* = {#}›
unfolding *true-clss-def clauses-def* **by** *fastforce*
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
assumes *propagate*** *S X*
shows
rtranclp-propagate-init-clss: *init-clss X* = *init-clss S* **and**
rtranclp-propagate-learned-clss: *learned-clss X* = *learned-clss S*
using *assms* **by** (*induction rule*: *rtranclp-induct*) (*auto elim*: *propagateE*)

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-l* (*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** (*blast dest*: *rtranclp-propagate-init-clss*)
have *learnedX*: *learned-clss X* = {#}
using *X learned* **by** (*auto dest*: *rtranclp-propagate-learned-clss*)
obtain *E* **where**
E: *E* ∈# *init-clss X* + *learned-clss X* **and**
Not-E: *trail X* ⊨*as CNot E*
using *Xconf* **by** (*auto simp add*: *clauses-def elim*!: *conflictE*)
have *lits-of-l* (*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* ⊨*s CNot E*
using *Not-E*
by (*fastforce simp add*: *true-annots-def true-annot-def true-clss-def true-cls-def*)
have ¬ *set M* ⊨*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*. ¬ *is-decided l*)

lemma *rtranclp-propagate-is-trail-append*:
propagate** *S T* $\Longrightarrow$ $\exists c$. *trail T* = *c* @ *trail S*
by (*induction rule*: *rtranclp-induct*) (*auto elim*: *propagateE*)

lemma *rtranclp-propagate-is-update-trail*:
propagate** *S T* $\Longrightarrow$ *cdcl*$_W$*-M-level-inv S* $\Longrightarrow$
init-clss S = *init-clss T* $\wedge$ *learned-clss S* = *learned-clss T* $\wedge$ *backtrack-lvl S* = *backtrack-lvl T*
$\wedge$ *conflicting S* = *conflicting T*
proof (*induction rule*: *rtranclp-induct*)
case *base*
then show *?case* **unfolding** *state-eq-def* **by** (*auto simp*: *cdcl*$_W$*-M-level-inv-decomp*)
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*: *elim*: *propagateE*)
qed

lemma $cdcl_W$*-stgy-strong-completeness-n*:
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**
atm-incl: *atm-of* ‘ (*set M*) ⊆ *atms-of-mm N* **and**
distM: *distinct M* **and**
length: $n \leq$ *length M*
shows
∃ *M′ k S*. *length M′* ≥ *n* ∧
lits-of-l M′ ⊆ *set M* ∧
no-dup M′ ∧
state S = (*M′*, *N*, {#}, *k*, *None*) ∧
$cdcl_W$*-stgy*** (*init-state N*) *S*
using *length*
proof (*induction n*)
case *0*
have *state* (*init-state N*) = ([], *N*, {#}, *0*, *None*)
by (*auto simp*: *state-eq-def simp del*: *state-simp*)
moreover have
0 ≤ *length* [] **and**
lits-of-l [] ⊆ *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′* ≥ *n* **and**
M′: *lits-of-l M′* ⊆ *set M* **and**
n-d[*simp*]: *no-dup M′* **and**
S: *state S* = (*M′*, *N*, {#}, *k*, *None*) **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 $cdcl_W$*-M-level-inv-S0-*$cdcl_W$ *rtranclp-*$cdcl_W$*-stgy-consistent-inv st* **apply** *blast*
using $cdcl_W$*-M-level-inv-S0-*$cdcl_W$ *no-strange-atm-S0 rtranclp-*$cdcl_W$*-no-strange-atm-inv*
rtranclp-$cdcl_W$*-stgy-rtranclp-*$cdcl_W$ *st* **by** *blast*

{ **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*: *comp-def*)
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-l* (*trail S′*) ⊆ *set M*
using *S′ full* $cdcl_W$*-cp-propagate-completeness*[*OF assms*(*1*−*3*), *of S*] *M′ S*

by (*auto simp*: *comp-def*)
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*: *comp-def rev-map*)
with *propa* **have** *length* (*trail S'*) > *n*
using *l-M' propa* **by** (*induction rule*: *tranclp.induct*) (*auto elim*: *propagateE*)
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*: *comp-def*)
have *S-S'*: *state S'* = (*trail S'*, *N*, {#}, *backtrack-lvl S'*, *None*)
using *S* **by** *auto*
have *cdcl*$_W$*-stgy*** (*init-state N*) *S'*
apply (*rule rtranclp.rtrancl-into-rtrancl*)
using *st* **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 n-d* **by** *blast*
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*: *true-clss-def comp-def rev-map*
clauses-def elim!: *conflictE*)
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-l M′*) = *length M′*
by (*induction M′*) (*auto simp add*: *lits-of-def card-insert-if*)
then have *lits-of-l M′* ⊂ *set M*
using *n M′ n′ lenM* **by** *auto*
then obtain *L* **where** *L*: *L* ∈ *set M* **and** *undef-m*: *L* ∉ *lits-of-l M′* **by** *auto*
moreover have *undef*: *undefined-lit M′ L*
using *M′ Decided-Propagated-in-iff-in-lits-of-l calculation*(*1*,*2*) *cons*
consistent-interp-def **by** (*metis* (*no-types*, *lifting*) *subset-eq*)
moreover have *atm-of L* ∈ *atms-of-mm* (*init-clss S*)
using *atm-incl calculation S* **by** *auto*
ultimately
have *dec*: *decide S* (*cons-trail* (*Decided L*) (*incr-lvl S*))
using *decide-rule*[*of S - cons-trail* (*Decided L*) (*incr-lvl S*)] *S*
by *auto*
let *?S′* = *cons-trail* (*Decided L*) (*incr-lvl S*)
have *lits-of-l* (*trail ?S′*) ⊆ *set M* **using** *L M′ S undef* **by** *auto*
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*
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-l* (*trail S′′*) ⊆ *set M*
using *S′′ full cdcl*$_W$*-cp-propagate-completeness*[*OF assms*(*1*−*3*), *of ?S′ S′′*] *L M′ S undef*
by *simp*
then have *Suc n* ≤ *length* (*trail S′′*) ∧ *lits-of-l* (*trail S′′*) ⊆ *set M*
using *l-M′ S undef* **by** *auto*
moreover
have *cdcl*$_W$*-M-level-inv* (*cons-trail* (*Decided L*)
(*update-backtrack-lvl* (*Suc* (*backtrack-lvl S*)) *S*))
using *S* ‹*cdcl*$_W$*-M-level-inv* (*cons-trail* (*Decided L*) (*incr-lvl S*))› **by** *auto*
then have *S′′*:
state S′′ = (*trail S′′*, *N*, {#}, *backtrack-lvl S′′*, *None*)
using *rtranclp-propagate-is-update-trail*[*OF S′′*] *S undef n-d′′ lev′′*
by *auto*
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

theorem 2.9.11 page 84 of Weidenbach's book (with strategy)

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-mm N* **and**
distM: *distinct M*
shows
$\exists$ *M' k S*.
lits-of-l M' = *set M* $\wedge$
state S = (*M'*, *N*, {#}, *k*, *None*) $\wedge$
$cdcl_W$*-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-l M'* $\subseteq$ *set M* **and**
no-dup: *no-dup M'* **and**
T: *state T* = (*M'*, *N*, {#}, *k*, *None*) **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* (λ*l*. *atm-of* (*lit-of l*)) *M'*))) = *length M'*
using *distinct-card no-dup* **by** *fastforce*
moreover have *card* (*lits-of-l M'*) = *card* (*set* ((*map* (λ*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-l M'*) **using** *l* **unfolding** *lits-of-def* **by** *auto*
then have *set M* = *lits-of-l 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*
ultimately show *?thesis* **using** *st T* **by** *blast*
qed

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*) $\equiv$
($\forall$ *M K M' D*. *M'* @ *Decided K* # *M* = *trail S* $\longrightarrow$ *D* $\in$# *clauses S*
$\longrightarrow$ $\neg$*M* $\models$*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*)
case (*decide L T*) **note** *confl* = *this*(*1*) **and** *undef* = *this*(*2*) **and** *T* = *this*(*4*)
have [*simp*]: *clauses T* = *clauses S*
using *T undef* **by** *auto*
show *?case*
proof (*intro allI impI*)
fix *M'' K M' Da*
assume *M''* @ *Decided K* # *M'* = *trail T*
and *D*: *Da* ∈# *local.clauses T*
then have *tl M''* @ *Decided K* # *M'* = *trail S*
∨ (*M''* = [] ∧ *Decided K* # *M'* = *Decided L* # *trail S*)
using *T undef* **by** (*cases M''*) *auto*
moreover {
assume *tl M''* @ *Decided K* # *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 *Decided K* # *M'* = *Decided L* # *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 L D K i M1 M2 T*) **note** *confl* = *this*(*1*) **and** *LD* = *this*(*2*) **and** *decomp* = *this*(*3*)
and
T = *this*(*8*)
obtain *c* **where** *M*: *trail S* = *c* @ *M2* @ *Decided K* # *M1*
using *decomp* **by** *auto*

show *?case*
proof (*intro allI impI*)
fix *M ia K' M' Da*
assume *M'* @ *Decided K'* # *M* = *trail T*
then have *tl M'* @ *Decided K'* # *M* = *M1*
using *T decomp lev* **by** (*cases M'*) (*auto simp*: *cdcl*$_W$*-M-level-inv-decomp*)
let *?S'* = (*cons-trail* (*Propagated L D*)
(*reduce-trail-to M1* (*add-learned-cls D*
(*update-backtrack-lvl i* (*update-conflicting None S*)))))
assume *D*: *Da* ∈# *clauses T*
moreover{
assume *Da* ∈# *clauses S*
then have ¬*M* ⊨*as CNot Da* **using** ‹*tl M'* @ *Decided K'* # *M* = *M1*› *M confl smaller*
unfolding *no-smaller-confl-def* **by** *auto*

```
    }
    moreover {
      assume Da: Da = D
      have ¬M ⊨as CNot Da
        proof (rule ccontr)
          assume ¬ ?thesis
          then have −L ∈ lits-of-l M
            using LD unfolding Da by (simp add: in-CNot-implies-uminus(2))
          then have −L ∈ lits-of-l (Propagated L D # M1)
            using UnI2 ‹tl M' @ Decided K' # M = M1›
            by auto
          moreover
            have backtrack S ?S'
              using backtrack-rule[of S] backtrack.hyps
              by (force simp: state-eq-def simp del: state-simp)
            then have cdcl_W-M-level-inv ?S'
              using cdcl_W-consistent-inv[OF - lev] other[OF bj] by (auto intro: cdcl_W-bj.intros)
            then have no-dup (Propagated L D # M1)
              using decomp lev unfolding cdcl_W-M-level-inv-def by auto
          ultimately show False
            using Decided-Propagated-in-iff-in-lits-of-l defined-lit-map by auto
        qed
    }
    ultimately show ¬M ⊨as CNot Da
      using T 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 elim: conflictE)

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 M'' D
  assume M': M'' @ Decided K # 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 elim: propagate-high-levelE)
  have tl M'' @ Decided K # M' = trail S using M' S S'
    by (metis Pair-inject list.inject list.sel(3) ann-lit.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*) *cdcl$_W$-cp.simps* **by** *auto*
then show *?case* **using** *full-cdcl$_W$-cp-no-smaller-confl-inv*[*of S' S''*] *other'.hyps* **by** *blast*
qed

lemma *is-conflicting-exists-conflict*:
assumes ¬(∀ *D*∈#*init-clss S'* + *learned-clss S'*. ¬ *trail S'* ⊨*as CNot D*)
and *conflicting S'* = *None*
shows ∃ *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*)
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': $x = \{\#-L'\#\}$ $L' \in\# D$ by *auto*
obtain L'' where $L'' \in\# x$ and L'': *lits-of-l* (*Decided L* # *?M*) $\models l$ L''
using *M-D x T undef* unfolding *true-annots-def Ball-def true-annot-def CNot-def true-cls-def Bex-def* by *auto*
show $\exists L \in\# x$. *lits-of-l ?M* $\models l$ L unfolding *Bex-def*
using $L'(1)$ $L'(2)$ ‹− $L \notin\# D$› ‹$L'' \in\# x$›
‹*lits-of-l* (*Decided L* # *trail S*) $\models l$ L''› by *auto*
qed
then show *False* using ‹¬ *?M* $\models as$ *CNot D*› by *auto*
qed
have *atm-of L* $\notin$ *atm-of* ‘ (*lits-of-l ?M*)
using *undef defined-lit-map* unfolding *lits-of-def* by *fastforce*
then have *get-level* (*Decided L* # *?M*) $(-L)$ = *?k* + *1*
using *lev* unfolding $cdcl_W$*-M-level-inv-def* by *auto*
then have $-L \in\# D \wedge$ *get-level ?M'* $(-L)$ = *backtrack-lvl T*
using ‹$-L \in\# D$› *T undef* by *auto*
then show $\exists La$. $La \in\# D \wedge$ *get-level ?M' La* = *backtrack-lvl T*
by *blast*
qed
next
case *resolve*
then show *?case* by *auto*
next
case *skip*
then show *?case* by *auto*
next
case (*backtrack L D K i M1 M2 T*) note *decomp* = *this*(*3*) and *lev-K* = *this*(*7*) and *T* = *this*(*8*)
show *?case*
proof (*rule HOL.disjI2*, *clarify*)
fix *Da*
assume *Da*: $Da \in\#$ *clauses T* and *M-D*: *trail T* $\models as$ *CNot Da*
obtain *c* where *M*: *trail S* = *c* @ *M2* @ *Decided K* # *M1*
using *decomp* by *auto*
have *tr-T*: *trail T* = *Propagated L D* # *M1*
using *T decomp lev* by (*auto simp*: $cdcl_W$*-M-level-inv-decomp*)
have *backtrack S T*
using *backtrack-rule*[*of S*] *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* $cdcl_W$*-bj.backtrack* $cdcl_W$*-o.bj* by *blast*
then have $-$ $L \notin$ *lits-of-l M1*
using *lev* $cdcl_W$*-M-level-inv-def tr-T* unfolding *consistent-interp-def* by (*metis insert-iff list.simps*(*15*) *lits-of-insert ann-lit.sel*(*2*))
{ assume $Da \in\#$ *clauses S*
then have ¬*M1* $\models as$ *CNot Da* using *no-l M* unfolding *no-smaller-confl-def* by *auto*
}
moreover {
assume *Da*: *Da* = *D*
have ¬*M1* $\models as$ *CNot Da* using ‹− $L \notin$ *lits-of-l M1*› unfolding *Da*
using *backtrack.hyps*(*2*) *in-CNot-implies-uminus*(*2*) by *auto*
}
ultimately have ¬*M1* $\models as$ *CNot Da*
using *Da T decomp lev* by (*fastforce simp*: $cdcl_W$*-M-level-inv-decomp*)
then have $-L \in\# Da$
using *M-D* ‹− $L \notin$ *lits-of-l M1*› *T* unfolding *tr-T true-annots-true-cls true-clss-def*
by (*auto simp*: *uminus-lit-swap*)

have *no-dup* (*Propagated L D # M1*)
  using *lev lev′ T decomp* **unfolding** *cdcl*$_W$*-M-level-inv-def* **by** *auto*
then have *L*: *atm-of L* ∉ *atm-of ‘ lits-of-l M1* **unfolding** *lits-of-def* **by** *auto*
have *get-level* (*Propagated L D # M1*) (−*L*) = *i*
  using *lev-K lev* **unfolding** *cdcl*$_W$*-M-level-inv-def*
  by (*simp add*: *M image-Un atm-lit-of-set-lits-of-l*)

then have −*L* ∈# *Da* ∧ *get-level* (*trail T*) (−*L*) = *backtrack-lvl T*
  using ‹−*L* ∈# *Da*› *T decomp lev* **by** (*auto simp*: *cdcl*$_W$*-M-level-inv-def*)
then show ∃ *La*. *La* ∈# *Da* ∧ *get-level* (*trail T*) *La* = *backtrack-lvl T*
  by *blast*
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* **and**
lev: *cdcl*$_W$*-M-level-inv S*
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* **and**
[*simp*]: *learned-clss U* = *learned-clss S* **and**
[*simp*]: *init-clss U* = *init-clss S*
using *no-confl rtranclp-propagate-is-update-trail lev* **by** *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 *meson*
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***and**
lev: $cdcl_W$*-M-level-inv S*
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 lev rtranclp-propagate-is-update-trail* **by** *auto*
have *c*: *learned-clss U = learned-clss T init-clss U = init-clss T*
using *conf* **by** (*auto elim*: *conflictE*)
obtain *D* **where** *trail T* ⊨*as CNot D* ∧ *conflicting U = Some D* ∧ *D* ∈# *clauses S*
using *conf p c* **by** (*fastforce simp*: *clauses-def elim!*: *conflictE*)
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'
∨ (*conflicting* S' = *None* ⟶ (∀ *D*∈#*clauses* S'. *trail* S' ⊨*as CNot D*
⟶ (∃ *L*. *L* ∈# *D* ∧ *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' ≠ *None*
have *conflicting* S ≠ *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: ∀ *D* ∈# *clauses* S'. *trail* S' ⊨*as CNot D*
⟶ (∃ *L*. *L* ∈# *D* ∧ *get-level* (*trail* S') *L* = *backtrack-lvl* S')
{ assume ∀ *D*∈#*clauses* S'. ¬ *trail* S' ⊨*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 ¬(∀ *D*∈#*clauses* S'. ¬ *trail* S' ⊨*as CNot D*)
then obtain *T D* **where**
propagate** S' *T* **and**
conflict T S'' **and**
D: *D* ∈# *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*) *lev*$'$ **by** (*metis* (*mono-tags*, *lifting*) *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-decided 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-l* (*trail* S')
using ‹*trail* S' $\models$*as CNot* D› ‹$L \in\#$ D› *in-CNot-implies-uminus*(*2*) **by** *blast*
{ fix x :: ($'v$, $'v$ *clause*) *ann-lit* **and**
xb :: ($'v$, $'v$ *clause*) *ann-lit*
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-l* 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-l* M
using ‹*trail* S'' $\models$*as CNot* D› **unfolding** M
by (*auto simp add*: *true-cls-def M true-annots-def true-annot-def*
split: *if-split-asm*)
{ fix x :: ($'v$, $'v$ *clause*) *ann-lit* **and**
xb :: ($'v$, $'v$ *clause*) *ann-lit*
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* ∉ *atm-of* ‘ *lits-of-l* (*trail S′*)
  using *nd* ‹*L* ∈# *D*› *LM* **unfolding** *M* **by** (*auto simp add*: *lits-of-def*)
show *?thesis*
  proof −
    have *atm-of L* ∈ *atm-of* ‘ *lits-of-l M*
      using ‹−*L* ∈ *lits-of-l M*›
      by (*simp add*: *atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set lits-of-def*)
    then have *get-level* (*M* @ *trail S′*) *L* = *backtrack-lvl S′*
      using *lev′ LS′ nm* **unfolding** $cdcl_W$*-M-level-inv-def* **by** *auto*
    then show *?thesis*
      using *nm* ‹*L*∈#*D*› ‹*conflicting S″* = *Some D*›
      unfolding *lits-of-def btS M*
      by *auto*
  qed
qed
}
ultimately have *conflict-is-false-with-level S″* **by** *blast*
}
moreover
{
  assume *conflicting S′* ≠ *None*
  have *no-clause-is-false S′* **using** ‹*conflicting S′* ≠ *None*› **by** *auto*
  then have *conflict-is-false-with-level S″* **using** *calculation*(*3*) **by** *presburger*
}
ultimately show *?case* **by** *blast*
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-ann-decomposition* (*trail S*)) **and**
    learned: $cdcl_W$*-learned-clause S* **and**
    alien: *no-strange-atm S*
  shows *no-smaller-confl S′* ∧ *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

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* {#} ∧ *unsatisfiable* (*set-mset* (*init-clss S′*)))
∨ (*conflicting S′* = *None* ∧ *trail S′* ⊨*asm init-clss S′*)
proof −
let *?S* = *init-state N*
have
termi: ∀ *S″*. ¬*cdcl$_W$-stgy S′ S″* **and**
step: *cdcl$_W$-stgy** ?S 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-ann-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 ∀ *D*∈#*N*. ¬ [] ⊨*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′* ≠ *None*
using *cp cdcl$_W$-cp.intros* **by** (*auto elim!*: *conflictE simp*: *state-eq-def simp del*: *state-simp*)
then have *cdcl$_W$-cp^{++} S S′* **by** *blast*
moreover have *no-step cdcl$_W$-cp S′*
using ‹*conflicting S′* ≠ *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 $S \neq$ *None*
shows *False*
using *assms* **by** (*induct rule*: *cdcl*$_W$*-cp.induct*) (*auto elim*: *propagateE conflictE*)

lemma *cdcl*$_W$*-o-fst-empty-conflicting-false*:
assumes *cdcl*$_W$*-o* S S'
and *trail* S = []
and *conflicting* $S \neq$ *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* $S \neq$ *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' $\Longrightarrow$ *conflicting* S = *Some* {#} $\Longrightarrow$ *False*
by (*induction rule*: *cdcl*$_W$*-cp.induct*) (*auto elim*: *propagateE conflictE*)

lemma *rtranclp-cdcl*$_W$*-cp-conflicting-is-false*:
cdcl$_W$*-cp*$^{++}$ S S' $\Longrightarrow$ *conflicting* S = *Some* {#} $\Longrightarrow$ *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' $\Longrightarrow$ *conflicting* S = *Some* {#} $\Longrightarrow$ *False*
by (*induction rule*: *cdcl*$_W$*-o-induct*) *auto*

lemma *cdcl*$_W$*-stgy-conflicting-is-false*:
cdcl$_W$*-stgy* S S' $\Longrightarrow$ *conflicting* S = *Some* {#} $\Longrightarrow$ *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' $\Longrightarrow$ *conflicting* S = *Some* {#} $\Longrightarrow$ $S' = S$
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
$\forall$ $m\in$ *set* M. $\neg$*is-decided* 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-ann-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 $\exists M''$. *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 *fastforce*
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*: $M \models as$ *CNot* $(p - \{\#K\#\})$ **and** *Kp*: $K \in\# p$
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*
obtain p' **where**
p': *hd-trail* S = *Propagated* K p' **and**
pp': $p' = p$
using *S K* **by** (*cases hd-trail S*) *auto*
have *conflicting S* = *Some D*
using *S E* **by** (*cases conflicting S*) *auto*
then have *DD*: $D = D$
using *S E* **by** *auto*
consider (D) $D = \{\#\}$ | (D') $D \neq \{\#\}$ **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 elim*: *propagateE conflictE*)
then have *no-step* $cdcl_W$*-cp S* **by** (*auto simp*: $cdcl_W$*-cp.simps*)
have *res-skip*: $\exists T$. (*resolve S T* $\wedge$ *no-step skip S* $\wedge$ *full* $cdcl_W$*-cp T T*)
$\vee$ (*skip S T* $\wedge$ *no-step resolve S* $\wedge$ *full* $cdcl_W$*-cp T T*)
proof *cases*
assume −*lit-of L* $\notin\#$ *D*
then obtain *T* **where** *sk*: *skip S T*
using *S D' K skip-rule* **unfolding** *E* **by** *fastforce*
then have *res*: *no-step resolve S*
using ‹−*lit-of L* $\notin\#$ *D*› *S D' K* **unfolding** *E*
by (*auto elim*!: *skipE resolveE*)
have *full* $cdcl_W$*-cp T T*
using *sk* **by** (*auto intro*!: *option-full-*$cdcl_W$*-cp elim*: *skipE*)

then show *?thesis*
using *sk res* **by** *blast*
next
assume *LD*: $\neg -$*lit-of L* $\notin\#$ *D*
then have *D*: *Some D* = *Some* ((*D* − {#−*lit-of L*#}) + {#−*lit-of L*#})
by (*auto simp add*: *multiset-eq-iff*)

have $\bigwedge$*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' get-level-skip-all-not-decided get-maximum-level-exists-lit nm not-gr0*)
qed
then obtain *T* **where** *sk*: *resolve S T*
using *resolve-rule*[*of S K p' D*] *S p'* ‹*K* ∈# *p*› *D LD*
unfolding *K' D E pp'* **by** *auto*
then have *res*: *no-step skip S*
using *LD S D' K* **unfolding** *E*
by (*auto elim*!: *skipE resolveE*)
have *full* $cdcl_W$*-cp T T*
using *sk* **by** (*auto simp*: *option-full-*$cdcl_W$*-cp elim*: *resolveE*)
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-ann-decomposition* (*L* # *M*) = [([], *L#M*)]
using *nm* **unfolding** *K* **apply** (*induction M rule*: *ann-lit-list-induct*, *simp*)
by (*rename-tac L xs*, *case-tac hd* (*get-all-ann-decomposition xs*), *auto*)+
then have *no-b*: *no-step backtrack S*
using *nm S* **by** (*auto elim*: *backtrackE*)
have *no-d*: *no-step decide S*
using *S E* **by** (*auto elim*: *decideE*)

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* $cdcl_W$*-cp-state-eq-compatible resolve-unique skip-unique* **unfolding** $cdcl_W$*-stgy.simps* $cdcl_W$*-o.simps full-unfold full1-def* **by** (*blast dest*!: *tranclpD elim*!: $cdcl_W$*-bj.cases*)+
then obtain *D'* **where** *T*: *state T* = (*M*, *N*, *U*, *0*, *Some D'*)
using *S E* **by** (*auto elim*!: *skipE resolveE simp*: *state-eq-def simp del*: *state-simp*)

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*[*of init-state N*] **by** *auto*

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*[*of ?S*] **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 *conf*: ¬ *?thesis*
obtain *E* **where**
ES: *E* ∈# *init-clss St* **and**
E: *E* = {#}
using *empty cls-St* **by** *metis*
then have ∃ *T*. *conflict St T*

using *empty cls-St conflict-rule*[*of St E*] *ES conf* **unfolding** *E*
by (*auto simp*: *clauses-def dest*:)
then show *False* **using** *fullSt* **unfolding** *full1-def* **by** *blast*
qed

have *1*: $\forall$ *m*$\in$*set* (*trail St*). $\neg$ *is-decided 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-ann-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* $\neq$ *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* $\neq$ *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 satisfiable-def true-cls-empty true-clss-def*)
ultimately show *?thesis* **by** *auto*
qed

theorem 2.9.9 page 83 of Weidenbach's book

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* {#} $\wedge$ *unsatisfiable* (*set-mset* (*init-clss S'*)))
$\vee$ (*conflicting S'* = *None* $\wedge$ *trail S'* $\models$*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*

theorem 2.9.9 page 83 of Weidenbach's book

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* {#} $\wedge$ *unsatisfiable* (*set-mset N*))
$\vee$ (*conflicting S'* = *None* $\wedge$ *trail S'* $\models$*asm N* $\wedge$ *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' $\models$*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-l* (*trail* S')) **unfolding** *cdcl$_W$-M-level-inv-def* **by** *blast*
moreover have *lits-of-l* (*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 *conflict-driven-clause-learning$_W$*
begin

2.1.6 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 $\longleftrightarrow$
no-strange-atm S $\wedge$
cdcl$_W$-M-level-inv S $\wedge$
($\forall$ *s* $\in$# *learned-clss S*. $\neg$*tautology s*) $\wedge$
distinct-cdcl$_W$-state S $\wedge$
cdcl$_W$-conflicting S $\wedge$
all-decomposition-implies-m (*init-clss S*) (*get-all-ann-decomposition* (*trail S*)) $\wedge$
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-ann-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 $\forall$ *s*$\in$#*learned-clss* S'. $\neg$ *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*)

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*)
case (*backtrack* L C K i *M1 M2* T) **note** *decomp* = *this*(*3*) **and** *undef* = *this*(*6*) **and** T = *this*(*8*) **and**
D-T = *this*(*10*) **and** *D-S* = *this*(*11*)
then have $D = C$
using *not-gr0 lev* **by** (*auto simp*: *cdcl*$_W$*-M-level-inv-decomp*)
then show *?case*
using T *backtrack.hyps*(*1*−*5*) *backtrack.intros*[*OF backtrack.hyps*(*1*,*2*)] *backtrack.hyps*(*3*−*7*)
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* $D \in\#$ *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*
have *cdcl*$_W$*-stgy*$^{++}$ *S″ U*
using *st″ o* **by** *force*
then show *?thesis*
by (*meson bt confl rtranclp-unfold st′*)
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-Decided-lit*:
  assumes *propagate* S S'
  shows *Decided* $K \in$ *set* (*trail* S) $\longleftrightarrow$ *Decided* $K \in$ *set* (*trail* S')
  using *assms* **by** (*auto elim*: *propagateE*)

lemma *conflict-no-more-Decided-lit*:
  assumes *conflict* S S'
  shows *Decided* $K \in$ *set* (*trail* S) $\longleftrightarrow$ *Decided* $K \in$ *set* (*trail* S')
  using *assms* **by** (*auto elim*: *conflictE*)

lemma $cdcl_W$*-cp-no-more-Decided-lit*:
  assumes $cdcl_W$*-cp* S S'
  shows *Decided* $K \in$ *set* (*trail* S) $\longleftrightarrow$ *Decided* $K \in$ *set* (*trail* S')
  using *assms* **apply** (*induct rule*: $cdcl_W$*-cp.induct*)
  using *conflict-no-more-Decided-lit propagate-no-more-Decided-lit* **by** *auto*

lemma *rtranclp-*$cdcl_W$*-cp-no-more-Decided-lit*:
  assumes $cdcl_W$*-cp*** S S'
  shows *Decided* $K \in$ *set* (*trail* S) $\longleftrightarrow$ *Decided* $K \in$ *set* (*trail* S')
  using *assms* **apply** (*induct rule*: *rtranclp-induct*)
  using $cdcl_W$*-cp-no-more-Decided-lit* **by** *blast+*

lemma $cdcl_W$*-o-no-more-Decided-lit*:
  assumes $cdcl_W$*-o* S S' **and** *lev*: $cdcl_W$*-M-level-inv* S **and** $\neg$*decide* S S'
  shows *Decided* $K \in$ *set* (*trail* S') $\longrightarrow$ *Decided* $K \in$ *set* (*trail* S)
  using *assms*
proof (*induct rule*: $cdcl_W$*-o-induct*)
  case *backtrack* **note** *decomp* = *this*(*3*) **and** *undef* = *this*(*8*) **and** T = *this*(*9*)
  then show *?case* **using** *lev* **by** (*auto simp*: $cdcl_W$*-M-level-inv-decomp*)
next
  case (*decide* L T)
  then show *?case* **using** *decide-rule*[*OF decide.hyps*] **by** *blast*
qed *auto*

lemma $cdcl_W$*-new-decided-at-beginning-is-decide*:
  assumes $cdcl_W$*-stgy* S S' **and**
  lev: $cdcl_W$*-M-level-inv* S **and**
  trail S' = M' @ *Decided* L # 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 *Decided* $L \in$ *set* (*trail* S') **and** *Decided* $L \notin$ *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-Decided-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 *Decided L* $\in$ *set* (*trail U*) **and** *Decided L* $\notin$ *set* (*trail S*)
using *no-dup* **unfolding** *S S' cdcl*$_W$*-M-level-inv-def* **by** (*auto simp add*: *rev-image-eqI*)
then have *Decided L* $\in$ *set* (*trail T*)
using *st rtranclp-cdcl*$_W$*-cp-no-more-Decided-lit* **unfolding** *full-def* **by** *blast*
then show *?case*
using *cdcl*$_W$*-o-no-more-Decided-lit*[*OF o*] ‹*Decided L* $\notin$ *set* (*trail S*)› *ns lev* **by** *meson*
qed

lemma *cdcl*$_W$*-o-is-decide*:
assumes *cdcl*$_W$*-o S T* **and** *lev*: *cdcl*$_W$*-M-level-inv S*
trail T = *drop* (*length* M_0) M' @ *Decided L* # *H* @ *M***and**
$\neg$ ($\exists M'$. *trail S* = M' @ *Decided L* # *H* @ *M*)
shows *decide S T*
using *assms*
proof (*induction rule*:*cdcl*$_W$*-o-induct*)
case (*backtrack L D K i M1 M2 T*)
then obtain *c* **where** *trail S* = *c* @ *M2* @ *Decided K* # *M1*
by *auto*
show *?case*
using *backtrack lev*
apply (*cases drop* (*length* M_0) M')
apply (*auto simp*: *cdcl*$_W$*-M-level-inv-decomp*)
using ‹*trail S* = *c* @ *M2* @ *Decided K* # *M1*›
by (*auto simp*: *cdcl*$_W$*-M-level-inv-decomp*)
next
case *decide*
show *?case* **using** *decide-rule*[*of S*] *decide*(*1*−*4*) **by** *auto*
qed *auto*

lemma *rtranclp-cdcl*$_W$*-new-decided-at-beginning-is-decide*:
assumes *cdcl*$_W$*-stgy*** *R U* **and**
trail U = M' @ *Decided L* # *H* @ *M* **and**
trail R = *M* **and**
cdcl$_W$*-M-level-inv R*
shows
$\exists$ *S T T'*. *cdcl*$_W$*-stgy*** *R S* $\wedge$ *decide S T* $\wedge$ *cdcl*$_W$*-stgy*** *T U* $\wedge$ *cdcl*$_W$*-stgy*** *S U* $\wedge$
no-step cdcl$_W$*-cp S* $\wedge$ *trail T* = *Decided L* # *H* @ *M* $\wedge$ *trail S* = *H* @ *M* $\wedge$ *cdcl*$_W$*-stgy S T'* $\wedge$
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* $\exists M'$. *trail T* = M' @ *Decided L* # *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** *ndecided*: $\forall$ *l*$\in$*set* M_0. $\neg$ *is-decided l*

using rtranclp-cdcl$_W$-cp-dropWhile-trail unfolding full1-def rtranclp-unfold by meson
then have MV: M' @ Decided L # H @ M = M_0 @ trail T unfolding W by simp
then have V: trail T = drop (length M_0) (M' @ Decided L # H @ M)
by auto
have takeWhile (Not o is-decided) M' = M_0 @ takeWhile (Not ∘ is-decided) (trail T)
using arg-cong[OF MV, of takeWhile (Not o is-decided)] ndecided
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 ndecided: ∀ l∈set M_0. ¬ is-decided l
using rtranclp-cdcl$_W$-cp-dropWhile-trail cp unfolding full-def by meson
then have MV: M' @ Decided L # H @ M = M_0 @ trail T' unfolding U by simp
then have V: trail T' = drop (length M_0) (M' @ Decided L # H @ M)
by auto
have takeWhile (Not o is-decided) M' = M_0 @ takeWhile (Not ∘ is-decided) (trail T')
using arg-cong[OF MV, of takeWhile (Not o is-decided)] ndecided
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 tr-T': trail T' = drop (length M_0) M' @ Decided L # H @ M using V by auto
then have LT': Decided L ∈ 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-Decided-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'› ‹Decided L ∈ set (trail T')› nd tr-T'
by (auto simp add: Cons-eq-append-conv elim: decideE)
have T': drop (length M_0) M' @ Decided L # H @ M = Decided L # trail T
using ‹decide T T'› ‹Decided L ∈ set (trail T')› nd tr-T'
by (auto elim: decideE)
have trail T' = Decided L # trail T
using ‹decide T T'› ‹Decided L ∈ set (trail T')› tr-T'
by (auto elim: decideE)
then have 5: trail T' = Decided L # 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' = Decided L # trail T›
‹trail T' = drop (length M_0) M' @ Decided L # 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* $T = M'$ @ *Decided* L # 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'' = *Decided* L # 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-decided-at-beginning-is-decide'*:
assumes $cdcl_W$*-stgy*** R U **and**
trail $U = M'$ @ *Decided* L # 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* $y = c$ @ *Decided* L # H @ M)
$\wedge$ ($\lambda a\ b.$ $cdcl_W$*-stgy* $a\ b \wedge$ ($\exists c.$ *trail* $a = c$ @ *Decided* L # 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 = *Decided* L # 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-decided-at-beginning-is-decide*[*OF assms*] **by** *blast*
have *n*: ¬ ($\exists c.$ *trail* $S' = c$ @ *Decided* L # H @ M) **using** *trS'* **by** *auto*
show *?thesis*
using *rtranclp-trans*[*OF st*] *rtranclp-exists-last-with-prop*[*of* $cdcl_W$*-stgy* S' T' -
λa -. ¬($\exists c.$ *trail* $a = c$ @ *Decided* L # H @ M), *OF S'T' T'U n*]
by *meson*
qed

lemma *beginning-not-decided-invert*:
assumes A: M @ $A = M'$ @ *Decided* K # H **and**
nm: $\forall m \in$*set* M. ¬*is-decided* m
shows $\exists M.$ $A = M$ @ *Decided* K # H
proof −
have A = *drop* (*length* M) (M' @ *Decided* K # H)

using *arg-cong*[*OF A*, *of drop* (*length M*)] **by** *auto*
moreover have *drop* (*length M*) (M' @ *Decided K* # *H*) = *drop* (*length M*) M' @ *Decided K* # *H*
using *nm* **by** (*metis* (*no-types*, *lifting*) *A drop-Cons'* *drop-append ann-lit.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-decided-is-decide-step*:
assumes $cdcl_W$*-stgy S T*
$\neg$ ($\exists$ *c. trail S* = *c* @ *Decided L* # *H* @ *M*) **and**
(λ*a b.* $cdcl_W$*-stgy a b* $\wedge$ ($\exists$ *c. trail a* = *c* @ *Decided L* # *H* @ *M*))** *T U* **and**
$\exists$ M'. *trail U* = M' @ *Decided L* # *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' @ *Decided L* # *H* @ *M* **by** *metis*
obtain M'' **where** M'': *trail T* = M'' @ *trail S* **and** *nm*: $\forall$ *m*$\in$ *set* M''. $\neg$*is-decided m*
using *cp* **unfolding** *full1-def*
by (*metis rtranclp-*$cdcl_W$*-cp-dropWhile-trail'* *tranclp-into-rtranclp*)
have *False*
using *beginning-not-decided-invert*[*of* M'' *trail S* M' *L 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 *tr*U' = *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 $\exists$ M'. *trail T* = M' @ *Decided L* # *H* @ *M*
using *tr*U' *beginning-not-decided-invert*[*of - trail T - L H* @ *M*] **by** *metis*
then obtain M' **where** M': *trail T* = M' @ *Decided L* # *H* @ *M*
by *auto*
with *o lev nd cp ns*
show *?case*
proof (*induction rule*: $cdcl_W$*-o-induct*)
case (*decide L*) **note** *dec* = *this*(*1*) **and** *cp* = *this*(*5*) **and** *ns* = *this*(*4*)
then have *decide S* (*cons-trail* (*Decided L*) (*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* L' *D K j M1 M2 T*) **note** *decomp* = *this*(*3*) **and** *undef* = *this*(*8*) **and**
T = *this*(*9*) **and** *trT* = *this*(*13*)
obtain *MS3* **where** *MS3*: *trail S* = *MS3* @ *M2* @ *Decided K* # *M1*
using *get-all-ann-decomposition-exists-prepend*[*OF decomp*] **by** *metis*
have *tl* (M' @ *Decided L* # *H* @ *M*) = *tl* M' @ *Decided L* # *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' @ *Decided L* # *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 $(\lambda a\ b.\ cdcl_W\text{-}stgy\ a\ b \wedge (\exists c.\ trail\ a = c\ @\ Decided\ L\ \#\ H\ @\ M))^{**}\ T\ U$ **and**
$\exists M'.\ trail\ U = M'\ @\ Decided\ L\ \#\ H\ @\ M$
shows $\exists M'.\ trail\ T = M'\ @\ Decided\ L\ \#\ H\ @\ M$
using *assms* **by** (*induction rule*: *rtranclp-induct*) *auto*

lemma *remove1-mset-eq-remove1-mset-same*:
$remove1\text{-}mset\ L\ D = remove1\text{-}mset\ L'\ D \Longrightarrow L \in\#\ D \Longrightarrow L = L'$
by (*metis diff-single-trivial insert-DiffM multi-drop-mem-not-eq single-eq-single union-right-cancel*)

lemma *cdcl*$_W$*-o-cannot-learn*:
assumes
cdcl$_W$*-o y z* **and**
lev: *cdcl*$_W$*-M-level-inv y* **and**
M: $trail\ y = c\ @\ Decided\ Kh\ \#\ H$ **and**
DL: $D \notin\#\ learned\text{-}clss\ y$ **and**
LD: $L \in\#\ D$ **and**
DH: $atms\text{-}of\ (remove1\text{-}mset\ L\ D) \subseteq atm\text{-}of\ `\ lits\text{-}of\text{-}l\ H$ **and**
LH: $atm\text{-}of\ L \notin atm\text{-}of\ `\ lits\text{-}of\text{-}l\ H$ **and**
learned: $\forall T.\ conflicting\ y = Some\ T \longrightarrow trail\ y \models as\ CNot\ T$ **and**
z: $trail\ z = c'\ @\ Decided\ Kh\ \#\ H$
shows $D \notin\#\ learned\text{-}clss\ z$
using *assms*(*1*−*2*) *M DL DH LH learned z*
proof (*induction rule*: *cdcl*$_W$*-o-induct*)
case (*backtrack* $L'\ D'$ *K j M1 M2 T*) **note** *confl* = *this*(*1*) **and** LD' = *this*(*2*) **and** *decomp* = *this*(*3*) **and** *levL* = *this*(*4*) **and** *levD* = *this*(*5*) **and** *j* = *this*(*6*) **and** *lev-K* = *this*(*7*) **and** *T* = *this*(*8*) **and** *z* = *this*(*15*)
def $i \equiv get\text{-}level\ (trail\ T)\ Kh$
have *levT*: *cdcl*$_W$*-M-level-inv T*
using *backtrack-rule*[*OF confl* LD' *decomp levL levD - - T*] *lev-K j lev*
by (*metis Suc-eq-plus1 cdcl*$_W$*.simps cdcl*$_W$*-bj.simps cdcl*$_W$*-consistent-inv cdcl*$_W$*-o.simps*)
obtain *M3* **where** *M3*: $trail\ y = M3\ @\ M2\ @\ Decided\ K\ \#\ M1$
using *decomp get-all-ann-decomposition-exists-prepend* **by** *metis*
have $c'\ @\ Decided\ Kh\ \#\ H = Propagated\ L'\ D'\ \#\ trail\ (reduce\text{-}trail\text{-}to\ M1\ y)$
using *z decomp T lev* **by** (*force simp*: *cdcl*$_W$*-M-level-inv-def*)
then obtain *d* **where** *d*: $M1 = d\ @\ Decided\ Kh\ \#\ H$
by (*metis* (*no-types*) *decomp in-get-all-ann-decomposition-trail-update-trail list.inject list.sel*(*3*) *ann-lit.distinct*(*1*) *self-append-conv2 tl-append2*)

have $atm\text{-}of\ Kh \notin atm\text{-}of\ `\ lits\text{-}of\text{-}l\ c'$
using *levT* **unfolding** *cdcl*$_W$*-M-level-inv-def z*
by (*auto simp*: *atm-lit-of-set-lits-of-l*)
then have *count-H*: $count\text{-}decided\ H = i - 1\ i > 0$
unfolding *z i-def* **by** *auto*
have *n-d-y*: *no-dup* (*trail y*) **and** *bt-y*: $backtrack\text{-}lvl\ y = count\text{-}decided\ (trail\ y)$
using *lev* **unfolding** *cdcl*$_W$*-M-level-inv-def* **by** *auto*
have *tr-T*: $trail\ T = Propagated\ L'\ D'\ \#\ M1$

using *decomp T n-d-y* **by** *auto*

show *?case*
proof
assume $D \in\#$ *learned-clss T*
then have *DLD'*: $D = D'$
using *DL T neq0-conv decomp n-d-y* **by** *fastforce*
have *L-cKh*: *atm-of L* $\in$ *atm-of ‘ lits-of-l* (*c* @ [*Decided Kh*])
using *LH learned M DLD'*[*symmetric*] *confl LD' LD*

apply (*auto simp add*: *image-iff dest*!: *in-CNot-implies-uminus*)
apply (*metis atm-of-uminus*)+ **done**
then consider (*Lc*) *atm-of L* $\in$ *atm-of ‘ lits-of-l c* **and** *atm-of L* $\neq$ *atm-of Kh* |
(*LKh*) *atm-of L* = *atm-of Kh* **and** *atm-of L* $\notin$ *atm-of ‘ lits-of-l c*
using *n-d-y M* **by** (*auto simp*: *atm-lit-of-set-lits-of-l*)
then have *lev-L-c-Kh*: *get-level* (*c* @ [*Decided Kh*]) $L \geq 1$
by *cases auto*
have *get-level* (*trail y*) *L* = *get-level* (*c* @ [*Decided Kh*]) *L* + *count-decided H*
using *get-rev-level-skip-end*[*OF L-cKh*, *of H*] **unfolding** *M* **by** *simp*
then have *get-level* (*trail y*) $L \geq i$
using *count-H lev-L-c-Kh* **by** *linarith*
then have *i-le-bt-y*: $i \leq$ *backtrack-lvl y*
using $cdcl_W$*-M-level-inv-get-level-le-backtrack-lvl*[*OF lev*, *of L*] **by** *linarith*
have *DD'*[*simp*]: *remove1-mset L D* = $D' - \{\#L'\#\}$
proof (*rule ccontr*)
assume *DD'*: $\neg$ *?thesis*
then have $L' \in\#$ *remove1-mset L D* **using** *DLD' LD* **by** (*metis LD' in-remove1-mset-neq*)
then have *get-level* (*trail y*) $L' \leq$ *get-maximum-level* (*trail y*) (*remove1-mset L D*)
using *get-maximum-level-ge-get-level* **by** *blast*
moreover
have $\forall x \in$ *atms-of* (*remove1-mset L D*). $x \notin$ *atm-of ‘ lits-of-l* (*c* @ *Decided Kh* # [])
using *DH n-d-y* **unfolding** *M* **by** (*auto simp*: *atm-lit-of-set-lits-of-l*)
from *get-maximum-level-skip-beginning*[*OF this*, *of H*]
have *get-maximum-level* (*trail y*) (*remove1-mset L D*) =
get-maximum-level H (*remove1-mset L D*)
unfolding *M* **by** (*simp add*: *get-maximum-level-skip-beginning*)
moreover
have *atm-of Kh* $\notin$ *atm-of ‘ lits-of-l c'*
using *levT* **unfolding** $cdcl_W$*-M-level-inv-def z*
by (*auto simp*: *atm-lit-of-set-lits-of-l*)
then have *count-decided H* < *i*
unfolding *i-def z* **by** *auto*
then have *0* < *i* − *count-decided H*
by *presburger*
ultimately have *get-maximum-level* (*trail y*) (*remove1-mset L D*) < *i*
by (*metis* (*no-types*) *count-decided-ge-get-maximum-level diff-is-0-eq diff-le-mono2 not-le*)
moreover
have $L \in\#$ *remove1-mset L' D'*
using *DLD'*[*symmetric*] *DD' LD* **by** (*metis in-remove1-mset-neq*)
then have *get-maximum-level* (*trail y*) (*remove1-mset L' D'*) $\geq$
get-level (*trail y*) *L*
using *get-maximum-level-ge-get-level* **by** *blast*
moreover
have *get-maximum-level* (*trail y*) (*remove1-mset L' D'*)
< *get-level* (*trail y*) *L*

using ⟨*get-level* (*trail y*) $L' \leq$ *get-maximum-level* (*trail y*) (*remove1-mset L D*)⟩
calculation(*1*) *i-le-bt-y levL* **by** *linarith*
ultimately show *False* **using** *backtrack.hyps*(*4*) **by** *linarith*
qed
then have LL': $L = L'$
using *LD* LD' *remove1-mset-eq-remove1-mset-same* **unfolding** DLD'[*symmetric*] **by** *fast*

have [*simp*]: *atm-of K* $\notin$ *atm-of* ' *lits-of-l M2* **and**
[*simp*]: *atm-of K* $\notin$ *atm-of* ' *lits-of-l M3*
using *lev* **unfolding** *M3* $cdcl_W$*-M-level-inv-def* **by** (*auto simp*: *atm-lit-of-set-lits-of-l*)
{ assume *D*: *remove1-mset L* $D' = \{\#\}$
then have *j0*: $j = 0$ **using** *levD j* **by** (*simp add*: LL')
have $\forall m \in$ *set M1*. $\neg$*is-decided m*
using *lev-K* **unfolding** *j0 M3* **by** (*auto simp*: *atm-lit-of-set-lits-of-l image-Un filter-empty-conv*)
then have *False* **using** *d* **by** *auto*
}
moreover {
assume *D*[*simp*]: *remove1-mset L* $D' \neq \{\#\}$
have $i \leq j$
using *lev count-H lev-K* **unfolding** *M3 d* $cdcl_W$*-M-level-inv-def* **by** (*auto simp add*: *atm-lit-of-set-lits-of-l*)
have $j > 0$ **apply** (*rule ccontr*)
using ⟨$i > 0$⟩ *lev-K* **unfolding** *M3 d*
by (*auto simp add*: *rev-swap*[*symmetric*] *dest*!: *upt-decomp-lt*)
obtain L'' **where**
$L''\in\#$ *remove1-mset L* D' **and**
$L''D'$: *get-level* (*trail y*) L'' = *get-maximum-level* (*trail y*)
(*remove1-mset L* D')
using *get-maximum-level-exists-lit-of-max-level*[*OF D*, *of trail y*] **by** *auto*
have $L''M$: *atm-of* $L'' \in$ *atm-of* ' *lits-of-l* (*trail y*)
using *get-level-ge-0-atm-of-in*[*of 0* L'' *trail y*] ⟨$j>0$⟩ *levD* $L''D'$
i-le-bt-y levL **by** (*simp add*: LL' *j*)
then have $L'' \in$ *lits-of-l* (*Decided Kh* # *d*)
proof −
{
assume $L''H$: *atm-of* $L'' \in$ *atm-of* ' *lits-of-l H*
then have *atm-of* $L'' \notin$ *atm-of* ' *lits-of-l* (*c* @ [*Decided Kh*])
using *n-d-y* **unfolding** *M* **by** (*auto simp*: *lits-of-def atm-of-eq-atm-of*)
then have *get-level* (*trail y*) L'' = *get-level H* L''
using $L''H$ **unfolding** *M* **by** *auto*
moreover have *get-level H* $L'' \leq$ *count-decided H*
by *auto*
ultimately have *False*
using ⟨$j>0$⟩ ⟨$i \leq j$⟩ $L''D'$ LL' ⟨*get-level H* $L'' \leq$ *count-decided H*⟩ *count-H*(*1*) *j*
unfolding *count-H* **by** *presburger*
}
moreover
have *atm-of* $L'' \in$ *atm-of* ' *lits-of-l H*
using DD' *DH* ⟨$L'' \in\#$ *remove1-mset L* D'⟩ *atm-of-lit-in-atms-of* LL' *LD*
LD' **by** *fastforce*
ultimately show *?thesis*
using DD' *DH* ⟨$L'' \in\#$ *remove1-mset L* D'⟩ *atm-of-lit-in-atms-of*
by *auto*
qed
moreover

have *atm-of* $L'' \in$ *atms-of* (*remove1-mset L D'*)
using ‹$L''\in\#$ *remove1-mset L D'*› **by** (*auto simp*: *atms-of-def*)

then have *atm-of* $L'' \in$ *atm-of* ‘ *lits-of-l H*
using *DH* **unfolding** *DD'* **unfolding** *LL'* **by** *blast*
ultimately have *False*
using *n-d-y* **unfolding** *M3 d LL'* **by** (*auto simp*: *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* @ *Decided Kh* # *H* **and**
D $\notin\#$ *learned-clss y* **and**
LD: *L* $\in\#$ *D* **and**
DH: *atms-of* (*remove1-mset L D*) $\subseteq$ *atm-of* ‘ *lits-of-l H* **and**
LH: *atm-of L* $\notin$ *atm-of* ‘ *lits-of-l H* **and**
$\forall$ *T*. *conflicting y* = *Some T* $\longrightarrow$ *trail y* $\models$*as CNot T* **and**
trail z = *c'* @ *Decided Kh* # *H*
shows *D* $\notin\#$ *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** *LD* = *this*(*7*) **and** *DH* = *this*(*8*) **and** *LH* = *this*(*9*) **and** *confl* = *this*(*10*) **and** *trU* = *this*(*11*)
obtain *c'* **where** *c'*: *trail T* = *c'* @ *Decided Kh* # *H*
using *cp beginning-not-decided-invert*[*of - trail T c' Kh 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 LD 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
(λ*a b*. *cdcl*$_W$*-stgy a b* $\wedge$ ($\exists$ *c*. *trail a* = *c* @ *Decided K*# *H* @ []))** *S z* **and**
cdcl$_W$*-all-struct-inv S* **and**
trail S = *c* @ *Decided K* # *H* **and**
D $\notin\#$ *learned-clss S* **and**
LD: *L* $\in\#$ *D* **and**
DH: *atms-of* (*remove1-mset L D*) $\subseteq$ *atm-of* ‘ *lits-of-l H* **and**
LH: *atm-of L* $\notin$ *atm-of* ‘ *lits-of-l H* **and**
$\exists$ *c'*. *trail z* = *c'* @ *Decided K* # *H*
shows *D* $\notin\#$ *learned-clss z*
using *assms*(*1*–*4*,*8*)
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* @ *Decided K* # *H* **using** *s* **by** *auto*
  obtain *c′* **where** *c′*: *trail U* = *c′* @ *Decided K* # *H* **using** *trU* **by** *blast*
  have $cdcl_W{}^{**}$ *S T*
    proof −
      have ∀ *p pa*. ∃ *s sa*. ∀ *sb sc sd se*. (¬ p^{**} (*sb*::′*st*) *sc* ∨ *p s sa* ∨ pa^{**} *sb sc*)
        ∧ (¬ *pa s sa* ∨ ¬ p^{**} *sd se* ∨ 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′*: ∀ *Ta*. *conflicting T* = *Some Ta* ⟶ *trail T* ⊨*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 - LD 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 ∉# *learned-clss S* **and**
    E ∈# *learned-clss T*
  shows ∃ *S′*. *backtrack S S′* ∧ *conflicting S* = *Some E* ∧ *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* ∈# *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

theorem 2.9.7 page 83 of Weidenbach's book

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* ∈# *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** *L K* :: *'v literal* **and** *M1 M2-loc* :: *('v, 'v clause) ann-lits*
  and *i* :: *nat* **where**
   T: *T* ∼ *cons-trail (Propagated L E)*
    (reduce-trail-to M1 (add-learned-cls E
      (update-backtrack-lvl i (update-conflicting None S))))
    and
  decomp: *(Decided K # M1, M2-loc)* ∈
        set (get-all-ann-decomposition (trail S)) **and**
  LD: *L* ∈# *E* **and**
  k: *get-level (trail S) L = backtrack-lvl S* **and**
  level: *get-level (trail S) L = get-maximum-level (trail S) E* **and**
  confl-S: *conflicting S = Some E* **and**
  i: *i = get-maximum-level (trail S) (remove1-mset L E)* **and**
  lev-K: *get-level (trail S) K = Suc i*
  using *confl* **apply** *(induction rule: backtrack.induct)*
    apply *(simp del: state-simp)*
    by *blast*
obtain *M2* **where**
  M: *trail S = M2 @ Decided K # M1*
  using *get-all-ann-decomposition-exists-prepend[OF decomp]* **unfolding** *i* **by** *(metis append-assoc)*
let *?E = E*
let *?E' = remove1-mset L ?E*
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* ⊨*as CNot ?E* **unfolding** *cdcl$_W$-conflicting-def confl-S* **by** *auto*
then have *MD*: *trail S* ⊨*as CNot ?E* **by** *auto*
then have *MD'*: *trail S* ⊨*as CNot ?E'* **using** *true-annot-CNot-diff* **by** *blast*
have *lev'*: *cdcl$_W$-M-level-inv S* **using** *invS* **unfolding** *cdcl$_W$-all-struct-inv-def* **by** *blast*

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 ?E'* ⊆ *atm-of ‘ lits-of-l M1*
  using *backtrack-atms-of-D-in-M1[OF lev' - decomp - - -, of E - i T] confl-S conf T decomp k*
  level lev' lev-K **unfolding** *i cdcl$_W$-conflicting-def* **by** *(auto simp: cdcl$_W$-M-level-inv-decomp)*
have *no-dup (trail S)* **using** *lev'* **by** *(auto simp: cdcl$_W$-M-level-inv-decomp)*
have *vars-in-M1*:
  ∀ *x* ∈ *atms-of ?E'. x* ∉ *atm-of ‘ lits-of-l (M2 @ [Decided K])*
  unfolding *Set.Ball-def* **apply** *(intro impI allI)*
    apply *(rule vars-of-D distinct-atms-of-incl-not-in-other[of*
    M2 @ Decided K # [] M1 ?E'])
    using ‹*no-dup (trail S)*› *M vars-of-D* **by** *simp-all*
have *M1-D*: *M1* ⊨*as CNot ?E'*
  using *vars-in-M1 true-annots-remove-if-notin-vars[of M2 @ Decided K # [] M1 CNot ?E']*
  MD' M **by** *simp*

have *backtrack-lvl S > 0* **using** *lev'* **unfolding** *cdcl$_W$-M-level-inv-def M* **by** *auto*

obtain *M1' K' Ls* **where**
  M': *trail S = Ls @ Decided K' # M1'* **and**
  Ls: ∀ *l* ∈ *set Ls.* ¬ *is-decided l* **and**
  set M1 ⊆ *set M1'*
  proof −
    let *?Ls = takeWhile (Not o is-decided) (trail S)*
    have *MLs*: *trail S = ?Ls @ dropWhile (Not o is-decided) (trail S)*

by *auto*
have *dropWhile* (*Not o is-decided*) (*trail S*) $\neq$ [] **unfolding** *M* **by** *auto*
moreover
from *hd-dropWhile*[*OF this*] **have** *is-decided*(*hd* (*dropWhile* (*Not o is-decided*) (*trail S*)))
by *simp*
ultimately
obtain K' **where**
$K'k$: *dropWhile* (*Not o is-decided*) (*trail S*)
= *Decided* K' # *tl* (*dropWhile* (*Not o is-decided*) (*trail S*))
by (*cases dropWhile* (*Not* ∘ *is-decided*) (*trail S*);
cases hd (*dropWhile* (*Not* ∘ *is-decided*) (*trail S*)))
simp-all
moreover have $\forall l \in$ *set ?Ls*. ¬*is-decided l* **using** *set-takeWhileD* **by** *force*
moreover have *set M1* ⊆ *set* (*tl* (*dropWhile* (*Not o is-decided*) (*trail S*)))
unfolding *M* **by** (*induction M2*) *auto*
ultimately show *?thesis* **using** *that*[*of takeWhile* (*Not* ∘ *is-decided*) (*trail S*)
K' *tl* (*dropWhile* (*Not o is-decided*) (*trail S*))] *MLs* **by** *simp*
qed

have *M1'-D*: *M1'* $\models$*as CNot ?E'* **using** *M1-D* ‹*set M1* ⊆ *set M1'*› **by** (*auto intro*: *true-annots-mono*)
have $-L \in$ *lits-of-l* (*trail S*) **using** *conf confl-S LD* **unfolding** $cdcl_W$*-conflicting-def*
by (*auto simp*: *in-CNot-implies-uminus*)
have *L-notin*: *atm-of L* ∈ *atm-of* ‘ *lits-of-l Ls* ∨ *atm-of L* = *atm-of* K'
proof (*rule ccontr*)
assume ¬ *?thesis*
then have *atm-of L* ∉ *atm-of* ‘ *lits-of-l* (*Decided* K' # *rev Ls*) **by** *simp*
then have *get-level* (*trail S*) *L* = *get-level M1' L*
unfolding M' **by** *auto*
moreover
have *get-level M1' L* ≤ *count-decided M1'*
by *auto*
then have *get-level M1' L* < *backtrack-lvl S*
using *lev'* **unfolding** $cdcl_W$*-M-level-inv-def* M'
by (*auto simp del*: *count-decided-ge-get-level*)
ultimately show *False* **using** *k* **by** *linarith*
qed
obtain *Y Z* **where**
RY: $cdcl_W$*-stgy*** *R Y* **and**
YZ: $cdcl_W$*-stgy Y Z* **and**
nt: ¬ (∃ *c*. *trail Y* = *c* @ *Decided* K' # *M1'* @ []) **and**
Z: (λ*a b*. $cdcl_W$*-stgy a b* ∧ (∃ *c*. *trail a* = *c* @ *Decided* K' # *M1'* @ []))** *Z S*
using *rtranclp-*$cdcl_W$*-new-decided-at-beginning-is-decide'*[*OF st' - - lev*, *of Ls* K'
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' @ *Decided* K' # *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-decided-is-decide-step*[*OF YZ nt Z*] M' **by** *auto*
have *trY*: *trail Y* = *M1'*
proof −

obtain M' **where** M: *trail* $Z = M'$ @ *Decided* K' # $M1'$
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** $\forall m\in set\ M''$. $\neg$*is-decided* m
using $Y'Z$ *rtranclp-cdcl*$_W$*-cp-dropWhile-trail'* **unfolding** *full-def* **by** *blast*
obtain M''' **where** *trail* $Y' = M'''$ @ *Decided* K' # $M1'$
using M'' **unfolding** M
by (*metis* (*no-types*, *lifting*) ‹$\forall m\in set\ M''$. $\neg$ *is-decided* m› *beginning-not-decided-invert*)
then show *?thesis* **using** *dec nt* **by** (*induction* M''') (*auto elim*: *decideE*)
qed
have *Y-CT*: *conflicting* Y = *None* **using** ‹*decide* Y Y'› **by** (*auto elim*: *decideE*)
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*: $E \in\#$ *clauses* Y
have *atm-of* $L \notin$ *atm-of* ‘ *lits-of-l M1*
apply (*rule backtrack-lit-skiped*[*of* - S])
using *decomp i k lev' lev-K* **unfolding** *cdcl*$_W$*-M-level-inv-def* **by** *auto*
then have *LM1*: *undefined-lit M1 L*
by (*metis Decided-Propagated-in-iff-in-lits-of-l 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 *Ex* (*propagate* Y)
using *propagate-rule*[*of* Y E L] *DL* $M1'$*-D L-trY Y-CT trY LD* **by** *auto*
then have *False* **using** ‹*no-step cdcl*$_W$*-cp* Y› *propagate'* **by** *blast*
}
moreover {
assume *DL*: $E \notin\#$ *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 elim*: *decideE*)
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 n: $E \notin\#$ *learned-clss* Z
using *DL lY-lZ YZ* **unfolding** *clauses-def* **by** *auto*
have *?E* $\notin\#$*learned-clss* S
apply (*rule rtranclp-cdcl*$_W$*-stgy-with-trail-end-has-not-been-learned*[*OF Z invZ trZ*])
apply (*simp add*: n)
using *LD* **apply** *simp*
apply (*metis* (*no-types*, *lifting*) ‹*set M1* $\subseteq$ *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 rtranclp-cdcl*$_W$*-stgy-no-more-init-clss*[*of* R S]
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.induct*) (*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 elim*: *decideE skipE resolveE*)
    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*)

Decrease of a Measure

fun $cdcl_W$*-measure* **where**
$cdcl_W$*-measure S* =
  [(*3*::*nat*) ^ (*card* (*atms-of-mm* (*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-mm* (*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-mm* (*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 *conflict-driven-clause-learning*$_W$
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-mm* (*learned-clss S*))
proof −
  have *set-mset* (*learned-clss S*) ⊆ *simple-clss* (*atms-of-mm* (*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-mm* (*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 $cdcl_W$*-measure-decreasing*:
  fixes *S* :: *'st*
  assumes
    $cdcl_W$ *S S'* **and**
    no-restart:
      ¬(*learned-clss S* ⊆# *learned-clss S'* ∧ [] = *trail S'* ∧ *conflicting S'* = *None*)
     and
    no-forget: *learned-clss S* ⊆# *learned-clss S'* **and**
    no-relearn: ⋀*S'*. *backtrack S S'* ⟹ ∀ *T*. *conflicting S* = *Some T* ⟶ *T* ∉# *learned-clss S*
      and
    alien: *no-strange-atm S* **and**
    M-level: $cdcl_W$*-M-level-inv S* **and**
    no-taut: ∀ *s* ∈# *learned-clss S*. ¬*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*) ∈ *lexn less-than 3*
  using *assms*(*1*) *M-level assms*(*2*,*3*)
proof (*induct rule*: $cdcl_W$*-all-induct*)
  case (*propagate C L*) **note** *conf* = *this*(*1*) **and** *undef* = *this*(*5*) **and** *T* = *this*(*6*)
  have *propa*: *propagate S* (*cons-trail* (*Propagated L C*) *S*)
    using *propagate-rule*[*OF propagate.hyps*(*1*,*2*)] *propagate.hyps* **by** *auto*
  then have *no-dup'*: *no-dup* (*Propagated L C* # *trail S*)
    using *M-level* $cdcl_W$*-M-level-inv-decomp*(*2*) *undef defined-lit-map* **by** *auto*

  let *?N* = *init-clss S*
  have *no-strange-atm* (*cons-trail* (*Propagated L C*) *S*)
    using *alien* $cdcl_W$*.propagate* $cdcl_W$*-no-strange-atm-inv propa M-level* **by** *blast*

then have *atm-of ‘ lits-of-l* (*Propagated L C* # *trail S*)
⊆ *atms-of-mm* (*init-clss S*)
using *undef* **unfolding** *no-strange-atm-def* **by** *auto*
then have *card* (*atm-of ‘ lits-of-l* (*Propagated L C* # *trail S*))
≤ *card* (*atms-of-mm* (*init-clss S*))
by (*meson atms-of-ms-finite card-mono finite-set-mset*)
then have *length* (*Propagated L C* # *trail S*) ≤ *card* (*atms-of-mm ?N*)
using *no-dup-length-eq-card-atm-of-lits-of-l no-dup′* **by** *fastforce*
then have *H*: *card* (*atms-of-mm* (*init-clss S*)) − *length* (*trail S*)
= *Suc* (*card* (*atms-of-mm* (*init-clss S*)) − *Suc* (*length* (*trail S*)))
by *simp*
show *?case* **using** *conf T undef* **by** (*auto simp*: *H lexn3-conv*)
next
case (*decide L*) **note** *conf* = *this*(*1*) **and** *undef* = *this*(*2*) **and** *T* = *this*(*4*)
moreover
have *dec*: *decide S* (*cons-trail* (*Decided L*) (*incr-lvl S*))
using *decide-rule decide.hyps* **by** *force*
then have $cdcl_W$:$cdcl_W$ *S* (*cons-trail* (*Decided L*) (*incr-lvl S*))
using $cdcl_W$*.simps* $cdcl_W$*-o.intros* **by** *blast*
moreover
have *lev*: $cdcl_W$*-M-level-inv* (*cons-trail* (*Decided L*) (*incr-lvl S*))
using $cdcl_W$ *M-level* $cdcl_W$*-consistent-inv*[*OF* $cdcl_W$] **by** *auto*
then have *no-dup*: *no-dup* (*Decided L* # *trail S*)
using *undef* **unfolding** $cdcl_W$*-M-level-inv-def* **by** *auto*
have *no-strange-atm* (*cons-trail* (*Decided L*) (*incr-lvl S*))
using *M-level alien calculation*(*4*) $cdcl_W$*-no-strange-atm-inv* **by** *blast*
then have *length* (*Decided L* # (*trail S*))
≤ *card* (*atms-of-mm* (*init-clss S*))
using *no-dup undef*
length-model-le-vars[*of cons-trail* (*Decided L*) (*incr-lvl S*)]
by *fastforce*
ultimately show *?case* **using** *conf* **by** (*simp add*: *lexn3-conv*)
next
case (*skip L C′ M D*) **note** *tr* = *this*(*1*) **and** *conf* = *this*(*2*) **and** *T* = *this*(*5*)
show *?case* **using** *conf T* **by** (*simp add*: *tr lexn3-conv*)
next
case *conflict*
then show *?case* **by** (*simp add*: *lexn3-conv*)
next
case *resolve*
then show *?case* **using** *finite* **by** (*simp add*: *lexn3-conv*)
next
case (*backtrack L D K i M1 M2 T*) **note** *conf* = *this*(*1*) **and** *decomp* = *this*(*3*) **and** *T* = *this*(*8*) **and** *lev* = *this*(*9*)
let *?S′* = *T*
have *bt*: *backtrack S ?S′*
using *backtrack.hyps backtrack.intros*[*of S D L K*] **by** *auto*
have *D* ∉# *learned-clss S*
using *no-relearn conf bt* **by** *auto*
then have *card-T*:
card (*set-mset* ({#*D*#} + *learned-clss S*)) = *Suc* (*card* (*set-mset* (*learned-clss S*)))
by *simp*
have *distinct-*$cdcl_W$*-state ?S′*
using *bt M-level distinct-*$cdcl_W$*-state-inv no-dup other* $cdcl_W$*-o.intros* $cdcl_W$*-bj.intros* **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*)) $\leq$ *3* ^ *card* (*atms-of-mm* (*learned-clss T*))
by (*auto simp*: *learned-clss-less-upper-bound*)
then have *H*: *card* (*set-mset* ({#*D*#} + *learned-clss S*))
$\leq$ *3* ^ *card* (*atms-of-mm* ({#*D*#} + *learned-clss S*))
using *T decomp M-level* **by** (*simp add*: $cdcl_W$-*M-level-inv-decomp*)
moreover
have *atms-of-mm* ({#*D*#} + *learned-clss S*) $\subseteq$ *atms-of-mm* (*init-clss S*)
using *alien conf* **unfolding** *no-strange-atm-def* **by** *auto*
then have *card-f*: *card* (*atms-of-mm* ({#*D*#} + *learned-clss S*))
$\leq$ *card* (*atms-of-mm* (*init-clss S*))
by (*meson atms-of-ms-finite card-mono finite-set-mset*)
then have (*3*::*nat*) ^ *card* (*atms-of-mm* ({#*D*#} + *learned-clss S*))
$\leq$ *3* ^ *card* (*atms-of-mm* (*init-clss S*)) **by** *simp*
ultimately have (*3*::*nat*) ^ *card* (*atms-of-mm* (*init-clss S*))
$\geq$ *card* (*set-mset* ({#*D*#} + *learned-clss S*))
using *le-trans* **by** *blast*
then show *?case* **using** *decomp diff-less-mono2 card-T T M-level*
by (*auto simp*: $cdcl_W$-*M-level-inv-decomp lexn3-conv*)
next
case *restart*
then show *?case* **using** *alien* **by** (*auto simp*: *state-eq-def simp del*: *state-simp*)
next
case (*forget C T*) **note** *no-forget* = *this*(*9*)
then have *C* $\in$# *learned-clss S* **and** *C* $\notin$# *learned-clss T*
using *forget.hyps* **by** *auto*
then have $\neg$ *learned-clss S* $\subseteq$# *learned-clss T*
by (*auto simp add*: *mset-leD*)
then show *?case* **using** *no-forget* **by** *blast*
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 less-than 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 less-than 3*
apply (*rule* $cdcl_W$-*measure-decreasing*)
using *assms*(*1*) *conflict* **apply** *blast*
using *assms*(*1*) **apply** (*auto simp*: *state-eq-def simp del*: *state-simp elim!*: *conflictE*)[*3*]
using *assms*(*2*) **apply** (*auto simp add*: $cdcl_W$-*all-struct-inv-def elim*: *conflictE*)
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 less-than 3*
apply (*rule* $cdcl_W$-*measure-decreasing*)

```
  using assms(1) decide other apply blast
          using assms(1) apply (auto simp: state-eq-def simp del: state-simp elim!: decideE)[3]
        using assms(2) apply (auto simp add: cdcl_W-all-struct-inv-def elim: decideE)
  done

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) ∈ lexn less-than 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) ∈ lexn less-than 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 less-than 3 by blast

  moreover have (cdcl_W-measure U, cdcl_W-measure T) ∈ lexn less-than 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-less-than] 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 less-than 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 less-than 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 less-than 3*
proof (*induction rule*:$cdcl_W$*-o-induct*)
case (*decide T*)
then show *?case* **using** *decide-measure-decreasing H decide.intros*[*OF decide.hyps*] **by** *blast*
next
case (*backtrack L D K i M1 M2 T*) **note** *conf* = *this*(*1*) **and** *decomp* = *this*(*3*) **and**
undef = *this*(*8*) **and** *T* = *this*(*9*)
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 conf*
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** (*auto simp*: *lexn3-conv*)
next
case *resolve*
then show *?case* **by** (*auto simp*: *lexn3-conv*)
qed
ultimately show *?case*
by (*metis* (*full-types*) *lexn-transI transD trans-less-than*)
qed
qed

Roughly corresponds to theorem 2.9.15 page 86 of Weidenbach's book (using a different bound)

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 $(cdcl_W\text{-}measure\ S,\ cdcl_W\text{-}measure\ R) \in lexn\ less\text{-}than\ 3$
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-less-than*, *of 3*] **unfolding** *trans-def* **by** *blast*

lemma *tranclp-*$cdcl_W$*-stgy-S0-decreasing*:
fixes *R S T* :: *'st*
assumes
pl: $cdcl_W\text{-}stgy^{++}$ (*init-state N*) *S* **and**
no-dup: *distinct-mset-mset N*
shows $(cdcl_W\text{-}measure\ S,\ cdcl_W\text{-}measure\ (init\text{-}state\ N)) \in lexn\ less\text{-}than\ 3$
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* ∧ $cdcl_W\text{-}stgy^{++}$ (*init-state N*) *S*}
apply (*rule wf-wf-if-measure'-notation2*[*of lexn less-than 3 - -* $cdcl_W$*-measure*])
 apply (*simp add*: *wf wf-lexn*)
using *tranclp-*$cdcl_W$*-stgy-S0-decreasing* **by** *blast*

lemma $cdcl_W$*-cp-wf-all-inv*:
wf {(*S'*, *S*). $cdcl_W$*-all-struct-inv S* ∧ $cdcl_W$*-cp S S'*}
(**is** *wf ?R*)
proof (*rule wf-bounded-measure*[*of -*
$\lambda S.\ card\ (atms\text{-}of\text{-}mm\ (init\text{-}clss\ S))+1$
$\lambda S.\ length\ (trail\ S) + (if\ conflicting\ S = None\ then\ 0\ else\ 1)$], *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 $cdcl_W$*-cp.simps* $cdcl_W$*-all-struct-inv-inv conflict* $cdcl_W$*.intros* $cdcl_W$*-all-struct-inv-inv*
by *blast+*
ultimately show *?case*
by (*auto simp*:$cdcl_W$*-cp.simps state-eq-def simp del*: *state-simp elim*!: *conflictE propagateE*
dest: *length-model-le-vars-all-inv*)
qed

end

end

2.2 Merging backjump rules

theory *CDCL-W-Merge*
imports *CDCL-W-Termination*
begin

Before showing that Weidenbach's CDCL is included in NOT's CDCL, we need to work on a variant of Weidenbach's calculus: *conflict-driven-clause-learning*$_W$*.conflict*, *conflict-driven-clause-learning*$_W$*.resolve*, *conflict-driven-clause-learning*$_W$*.skip*, and *conflict-driven-clause-learning*$_W$*.backtrack* have to be done in a single step since they have a single counterpart in NOTs CDCL.

We show that this new calculus has the same final states than Weidenbach's CDCL if the calculus starts in a state such that the invariant holds and no conflict has been found yet. The latter condition holds for initial state.

2.2.1 Inclusion of the states

context *conflict-driven-clause-learning*$_W$
begin
declare *cdcl*$_W$*.intros*[*intro*] *cdcl*$_W$*-bj.intros*[*intro*] *cdcl*$_W$*-o.intros*[*intro*]

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*!: *backtrackE simp*: *cdcl*$_W$*-M-level-inv-def*)
   apply (*elim resolveE*, *force elim*!: *backtrackE simp*: *cdcl*$_W$*-M-level-inv-def*)
  apply *standard*
  apply (*elim backtrackE*)
  apply (*force simp del*: *state-simp simp add*: *state-eq-def cdcl*$_W$*-M-level-inv-decomp*)
  done

skip-or-resolve corresponds to the *analyze* function in the code of MiniSAT.

inductive *skip-or-resolve* :: $'st \Rightarrow 'st \Rightarrow bool$ **where**
s-or-r-skip[*intro*]: *skip* S $T \Longrightarrow$ *skip-or-resolve* S T |
s-or-r-resolve[*intro*]: *resolve* S $T \Longrightarrow$ *skip-or-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*$_W$]
        by (*blast intro*: *skip-or-resolve.cases*)
      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** (*auto simp*: *cdcl*$_W$*-bj.simps dest*!: *skip-or-resolve.intros*)
    next
      case *SU*
      then show *?thesis*
        using *bj* **by** (*auto simp*: *cdcl*$_W$*-bj.simps dest*!: *skip-or-resolve.intros*)

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 simp*: *skip-or-resolve.simps*)

definition *backjump-l-cond* :: *'v clause* ⇒ *'v clause* ⇒ *'v literal* ⇒ *'st* ⇒ *'st* ⇒ *bool* **where**
backjump-l-cond ≡ $\lambda C\ C'\ L'\ S\ T$. *True*

definition *inv*$_{NOT}$ :: *'st* ⇒ *bool* **where**
inv$_{NOT}$ ≡ λS. *no-dup* (*trail S*)

declare *inv*$_{NOT}$*-def*[*simp*]
end

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

2.2.2 More lemmas conflict–propagate and backjumping

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-ann-decomposition-exists-prepend*
elim!: *backtrackE skipE resolveE*
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*
- λT. *length* (*trail T*) + (*if conflicting T* = *None then 0 else 1*), *simplified*])
using *cdcl*$_W$*-bj-measure* **by** *simp*

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* ($\lambda a\ b$. *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* $cdcl_W$] **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
$\exists M.$ *trail S* = *M* @ *trail T* $\wedge$ ($\forall m \in$*set M*. $\neg$*is-decided m*)
init-clss S = *init-clss T*
learned-clss S = *learned-clss T*
backtrack-lvl S = *backtrack-lvl T*
conflicting S = *conflicting T*
using *assms* **by** (*induction rule*: *rtranclp-induct*)
(*auto simp del*: *state-simp simp*: *state-eq-def elim*!: *skipE*)

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 *K i M1 M2 L D* **where**
conf: *conflicting V* = *Some D* **and**
LD: *L* $\in$# *D* **and**
decomp: (*Decided K* # *M1*, *M2*) $\in$ *set* (*get-all-ann-decomposition* (*trail V*)) **and**
lev-L: *get-level* (*trail V*) *L* = *backtrack-lvl V* **and**
max: *get-level* (*trail V*) *L* = *get-maximum-level* (*trail V*) *D* **and**
max-D: *get-maximum-level* (*trail V*) (*remove1-mset L D*) $\equiv$ *i* **and**
lev-k: *get-level* (*trail V*) *K* = *Suc i* **and**
W: *W* $\sim$ *cons-trail* (*Propagated L D*)
(*reduce-trail-to M1*
(*add-learned-cls D*
(*update-backtrack-lvl i*
(*update-conflicting None V*))))
using *bt inv* **by** (*elim backtrackE*) *metis+*
obtain *L' C' M E* **where**
tr: *trail T* = *Propagated L' C'* # *M* **and**
raw: *conflicting T* = *Some E* **and**
LE: $-L' \notin$# *E* **and**

E: $E \neq \{\#\}$ **and**
V: $V \sim$ *tl-trail T*
using *skip* **by** (*elim skipE*) *metis*
let *?M* = *Propagated L′ C′* # *trail V*
have *tr-M*: *trail T* = *?M*
using *tr V* **by** *auto*
have *MT*: *M* = *tl* (*trail T*) **and** *MV*: *M* = *trail V*
using *tr V* **by** *auto*
have *DE*[*simp*]: *D* = *E*
using *V conf raw* **by** (*auto simp add*: *state-eq-def simp del*: *state-simp*)
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 *tr-M* **unfolding** $cdcl_W$*-M-level-inv-def* **by** *auto*
let *?k* = *backtrack-lvl T*
have [*simp*]:
backtrack-lvl V = *?k*
using *V* **by** *simp*
have *?k* > *0*
using *decomp M-lev V tr* **unfolding** $cdcl_W$*-M-level-inv-def* **by** *auto*
then have *atm-of L* ∈ *atm-of* ‘ *lits-of-l* (*trail V*)
using *lev-L get-level-ge-0-atm-of-in*[*of 0 L* (*trail V*)] **by** *auto*
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-l* (*trail V*)
using *n-d′* **unfolding** *lits-of-def* **by** *auto*
have *?M* ⊨*as CNot D*
using *inv′ raw* **unfolding** $cdcl_W$*-conflicting-def* $cdcl_W$*-all-struct-inv-def tr-M* **by** *auto*
then have *L′* ∉# (*remove1-mset L D*)
using *L-L′ L′-M* ‹*Propagated L′ C′* # *trail V* ⊨*as CNot D*›
unfolding *true-annots-true-cls true-clss-def*
by (*auto simp*: *uminus-lit-swap atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set dest*!: *in-diffD*)
have [*simp*]: *trail* (*reduce-trail-to M1 T*) = *M1*
using *decomp tr W V* **by** *auto*
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* = *init-clss V* **and** [*simp*]: *learned-clss S* = *learned-clss V*
using *rtranclp-skip-state-decomp*[*OF* ‹$skip^{**}$ *S V*›] *V*
by (*auto simp del*: *state-simp simp*: *state-eq-def*)
then have
W-S: $W \sim$ *cons-trail* (*Propagated L E*) (*reduce-trail-to M1*
(*add-learned-cls E* (*update-backtrack-lvl i* (*update-conflicting None T*))))
using *W V M-lev decomp tr*
by (*auto simp del*: *state-simp simp*: *state-eq-def* $cdcl_W$*-M-level-inv-def*)

obtain *M2′* **where**
decomp′: (*Decided K* # *M1*, *M2′*) ∈ *set* (*get-all-ann-decomposition* (*trail T*))
using *decomp V* **unfolding** *tr-M* **by** (*cases hd* (*get-all-ann-decomposition* (*trail V*)),
cases get-all-ann-decomposition (*trail V*)) *auto*
moreover
from *L-L′* **have** *get-level ?M L* = *?k*
using *lev-L V* **by** (*auto split*: *if-split-asm*)

moreover
  have *atm-of L′ ∉ atms-of D*
    by (*metis DE LE L-L′* ‹*L′* ∉# (*remove1-mset L D*)› *in-remove1-mset-neq*
      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*
    using *calculation*(*2*) *lev-L max* **by** *auto*
moreover
  have *atm-of L′ ∉ atms-of* ((*remove1-mset L D*))
    by (*metis DE LE* ‹*L′* ∉# (*remove1-mset L D*)›
      atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set atms-of-def in-remove1-mset-neq
      in-atms-of-remove1-mset-in-atms-of)
  have *i = get-maximum-level ?M* ((*remove1-mset L D*))
    using *max-D* ‹*atm-of L′ ∉ atms-of* ((*remove1-mset L D*))› **by** *auto*
moreover have *atm-of L′ ≠ atm-of K*
  using *inv′ get-all-ann-decomposition-exists-prepend*[*OF decomp*]
  unfolding *cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-M-level-inv-def tr MV* **by** *auto*
ultimately have *backtrack T W*
  apply −
  apply (*rule backtrack-rule*[*of T - L K M1 M2′ i W*, *OF raw*])
  unfolding *tr-M*[*symmetric*]
        using *LD* **apply** *simp*
       apply *simp*
      apply *simp*
     apply *simp*
    apply *auto*[]
   using *W-S lev-k tr MV* **apply** *auto*[]
  using *W-S lev-k* **apply** *auto*[]
  done
then show *?thesis* **using** *IH inv* **by** *blast*
qed

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*!: *backtrackE*)
  then obtain *K i M1 M2 L D* **where**
    S: *conflicting S = Some D* **and**
    LD: *L* ∈# *D* **and**
    decomp: (*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail S*)) **and**
    lev-l: *get-level* (*trail S*) *L = backtrack-lvl S* **and**
    lev-l-D: *get-level* (*trail S*) *L = get-maximum-level* (*trail S*) *D* **and**
    i: *get-maximum-level* (*trail S*) (*remove1-mset L D*) ≡ *i* **and**
    lev-K: *get-level* (*trail S*) *K = Suc i* **and**
    W: *W* ∼ *cons-trail* (*Propagated L D*)
              (*reduce-trail-to M1*
                (*add-learned-cls D*
                  (*update-backtrack-lvl i*
                    (*update-conflicting None S*))))
    using *bt* **by** (*elim backtrackE*)

(*simp-all add*: *$cdcl_W$-M-level-inv-decomp state-eq-def del*: *state-simp*)
let *?D = remove1-mset L D*

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*: *trail S = MS @ M_T* **and** *M_T*: *M_T = trail T* **and** *nm*: *∀ m∈set MS. ¬is-decided m*
  using *rtranclp-skip-state-decomp*(*1*)[*OF skip*] *S M-lev* **by** *auto*
have *T*: *state T* = (*M_T*, *init-clss S*, *learned-clss S*, *backtrack-lvl S*, *Some D*)
  using *M_T rtranclp-skip-state-decomp*[*of S T*] *skip S*
  by (*auto simp del*: *state-simp simp*: *state-eq-def*)

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*
then have *∀ L∈#D. atm-of L ∈ atm-of ‘ lits-of-l M_T*
  by (*meson atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set true-annots-true-cls-def-iff-negation-in-model*)
moreover have *no-dup* (*trail S*)
  using *inv* **unfolding** *$cdcl_W$-all-struct-inv-def $cdcl_W$-M-level-inv-def* **by** *auto*
ultimately have *∀ L∈#D. atm-of L ∉ atm-of ‘ lits-of-l MS*
  unfolding *M* **unfolding** *lits-of-def* **by** *auto*
then have *H*: *⋀L. L∈#D ⟹ get-level* (*trail S*) *L = get-level M_T L*
  unfolding *M* **by** (*fastforce simp*: *lits-of-def*)
have [*simp*]: *get-maximum-level* (*trail S*) *D = get-maximum-level M_T D*
  using ‹*M_T ⊨as CNot D*› *M nm* ‹*∀ L∈#D. atm-of L ∉ atm-of ‘ lits-of-l MS*›
  by (*auto simp*: *get-maximum-level-skip-un-decided-not-present*)

have *lev-l′*: *get-level M_T L = backtrack-lvl S*
  using *lev-l LD* **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-decided-invert get-all-ann-decomposition-exists-prepend reduce-trail-to-trail-tl-trail-decomp*)
have *W*: *W ∼ cons-trail* (*Propagated L D*) (*reduce-trail-to M1* (*add-learned-cls D* (*update-backtrack-lvl i* (*update-conflicting None T*))))
  using *W T i decomp* **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*
  using *lev-l-D LD* **by** (*auto simp*: *H*)
have [*simp*]: *get-maximum-level* (*trail S*) *?D = get-maximum-level M_T ?D*
  by (*smt H get-maximum-level-exists-lit get-maximum-level-ge-get-level in-diffD le-antisym not-gr0 not-less*)
then have *i′*: *i = get-maximum-level M_T ?D*
  using *i* **by** *auto*
have *Decided K # M1 ∈ set* (*map fst* (*get-all-ann-decomposition* (*trail S*)))
  using *Set.imageI*[*OF decomp*, *of fst*] **by** *auto*
then have *Decided K # M1 ∈ set* (*map fst* (*get-all-ann-decomposition M_T*))
  using *fst-get-all-ann-decomposition-prepend-not-decided*[*OF nm*] **unfolding** *M* **by** *auto*
then obtain *M2′* **where** *decomp′*:(*Decided K # M1*, *M2′*) ∈ *set* (*get-all-ann-decomposition M_T*)

by *auto*
moreover
have *atm-of K* $\notin$ *atm-of ' lits-of-l MS*
using ⟨*no-dup* (*trail S*)⟩ *decomp'* **unfolding** *M* M_T
by (*auto simp*: *lits-of-def*)
then have *get-level* (*trail T*) *K* = *get-level* (*trail S*) *K*
unfolding *M* M_T **by** *auto*
ultimately show *backtrack T W*
apply −
apply (*rule backtrack.intros*[*of T D*])
using *T lev-l' lev-l-D' i' W LD lev-K i* **apply** *auto*[*7*]
using *T W* **unfolding** *i'*[*symmetric*]
by (*auto simp del*: *state-simp simp*: *state-eq-def*)
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*
∨ (∃ *U*. *skip-or-resolve*** *S U* ∧ *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 (∃ *U*. *skip-or-resolve*** *S U* ∧ *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 skip-or-resolve.simps*)+
qed

lemma *resolve-skip-deterministic*:
resolve S T ⟹ *skip S U* ⟹ *False*
by (*auto elim*!: *skipE resolveE*)

lemma *list-same-level-decomp-is-same-decomp*:
assumes *M-K*: *M* = *M1* @ *Decided K* # *M2* **and** *M-K'*: *M* = *M1'* @ *Decided K'* # *M2'* **and**
lev-KK': *get-level M K* = *get-level M K'* **and**
n-d: *no-dup M*

shows $K = K'$ **and** $M1 = M1'$ **and** $M2 = M2'$
proof –
  {
    fix *j j' K K' M1 M1' M2 M2'*
    assume
      M-K: $M = M1$ @ *Decided K* # *M2* **and**
      M-K': $M = M1'$ @ *Decided K'* # *M2'* **and**
      levKK': *get-level M K* = *get-level M K'* **and**
      j: *M* ! *j* = *Decided K* **and** *j-M*: $j <$ *length M* **and**
      j': *M* ! *j'* = *Decided K'* **and** *j'-M*: $j' <$ *length M* **and**
      jj: $j' > j$
    have $j \geq$ *length M1*
      proof (*rule ccontr*)
        assume ¬ *length M1* ≤ *j*
        then have $j <$ *length M1*
          by *auto*
        then have *Decided K* ∈ *set M1*
          using *j* **unfolding** *M-K*
          by (*auto simp*: *nth-append in-set-conv-nth split*: *if-splits*)
        from *Set.imageI*[*OF this*, *of* λ*L*. *atm-of* (*lit-of L*)]
        show *False* **using** *n-d* **unfolding** *M-K* **by** *auto*
      qed
    moreover then have $j' - Suc$ (*length M1*) < *length M2*
      using *j'-M jj M-K* **unfolding** *M-K'* **by** (*metis One-nat-def Suc-eq-plus1 add.left-commute*
        le-less-trans length-append less-diff-conv2 list.size(*4*) *not-less not-less-eq*)
    ultimately have *dec*: *Decided K'* ∈ *set M2*
      using *jj j j' j'-M* **unfolding** *M-K* **by** (*auto simp*: *nth-append in-set-conv-nth List.nth-Cons'*)
    obtain *xs ys* **where**
      M2: *M2* = *xs* @ *Decided K'* # *ys*
      using *List.split-list*[*OF dec*] **by** *auto*
    have [*simp*]: *atm-of K* ≠ *atm-of K'*
      using *n-d* **unfolding** *M-K M2* **by** *auto*
    have *atm-of K* ∉ *atm-of* ‘ *lits-of-l M1* **and** *atm-of K'* ∉ *atm-of* ‘ *lits-of-l M1* **and**
    atm-of K' ∉ *atm-of* ‘ *lits-of-l xs*
      using *n-d Set.imageI*[*OF dec*, *of* λ*L*. *atm-of* (*lit-of L*)] **unfolding** *M-K*
      using *n-d* **unfolding** *M-K M2*
      by (*auto simp*: *lits-of-def*)
    then have *False*
      using *M2 levKK'* **unfolding** *M-K* **by** (*auto simp*: *split*: *if-splits*)
  } **note** *H* = *this*
  have *Decided K* ∈ *set M* **and** *Decided K'* ∈ *set M*
     using *M-K* **apply** *simp*
    using *M-K'* **by** *simp*
  then obtain *j j'* **where**
    j: *M* ! *j* = *Decided K* **and** *j-M*: $j <$ *length M* **and**
    j': *M* ! *j'* = *Decided K'* **and** *j'-M*: $j' <$ *length M*
      using *in-set-conv-nth* **by** *metis*

  have [*simp*]: $j = j'$ **using** *H*[*OF M-K M-K' - j j-M j' j'-M*]
    H[*OF M-K' M-K - j' j'-M j j-M*] *lev-KK'* **by** *presburger*
  then show *KK'*: $K = K'$ **using** *j j'* **by** *auto*

  have *j-M1*: $j =$ *length M1*
    proof (*rule ccontr*)
      assume $j \neq$ *length M1*
      moreover then have $j - Suc$ (*length M1*) < *length M2* ∨ $j <$ *length M1*

using *j-M M-K* **unfolding** *M-K′* **by** *force*
ultimately have *Decided K* ∈ *set* (*M1* @ *M2*)
using *j* **unfolding** *M-K* **by** (*auto simp*: *nth-append in-set-conv-nth split*: *if-splits*)
from *Set.imageI*[*OF this*, *of* λ*L*. *atm-of* (*lit-of L*)]
show *False* **using** *n-d* **unfolding** *M-K* **by** *auto*
qed
have *j-M2*: *j′* = *length M1′*
proof (*rule ccontr*)
assume *j′* ≠ *length M1′*
moreover then have *j′* − *Suc* (*length M1′*) < *length M2′* ∨ *j′* < *length M1′*
using *j′-M M-K′* **unfolding** *M-K* **by** *force*
ultimately have *Decided K′* ∈ *set* (*M1′* @ *M2′*)
using *j′* **unfolding** *M-K′* **by** (*auto simp*: *nth-append in-set-conv-nth split*: *if-splits*)
from *Set.imageI*[*OF this*, *of* λ*L*. *atm-of* (*lit-of L*)]
show *False* **using** *n-d* **unfolding** *M-K′* **by** *auto*
qed

show *M1* = *M1′ M2* = *M2′*
using *arg-cong*[*OF M-K*, *of take j*] *j-M1 arg-cong*[*OF M-K′*, *of take j′*] *j-M2*
using *arg-cong*[*OF M-K*, *of drop* (*j*+*1*)] *j-M1 arg-cong*[*OF M-K′*, *of drop* (*j′*+*1*)] *j-M2*
by *auto*
qed

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* ∼ *U*
proof −
have *lev*: $cdcl_W$*-M-level-inv S*
using *inv* **unfolding** $cdcl_W$*-all-struct-inv-def* **by** *auto*
then obtain *K i M1 M2 L D* **where**
S: *conflicting S* = *Some D* **and**
LD: *L* ∈# *D* **and**
decomp: (*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail S*)) **and**
lev-l: *get-level* (*trail S*) *L* = *backtrack-lvl S* **and**
lev-l-D: *get-level* (*trail S*) *L* = *get-maximum-level* (*trail S*) *D* **and**
i: *get-maximum-level* (*trail S*) (*remove1-mset L D*) ≡ *i* **and**
lev-K: *get-level* (*trail S*) *K* = *Suc i* **and**
T: *T* ∼ *cons-trail* (*Propagated L D*)
(*reduce-trail-to M1*
(*add-learned-cls D*
(*update-backtrack-lvl i*
(*update-conflicting None S*))))
using *bt-T* **by** (*elim backtrackE*) (*force simp*: $cdcl_W$*-M-level-inv-def*)+

obtain *K′ i′ M1′ M2′ L′ D′* **where**
S′: *conflicting S* = *Some D′* **and**
LD′: *L′* ∈# *D′* **and**
decomp′: (*Decided K′* # *M1′*, *M2′*) ∈ *set* (*get-all-ann-decomposition* (*trail S*)) **and**
lev-l: *get-level* (*trail S*) *L′* = *backtrack-lvl S* **and**
lev-l-D: *get-level* (*trail S*) *L′* = *get-maximum-level* (*trail S*) *D′* **and**
i′: *get-maximum-level* (*trail S*) (*remove1-mset L′ D′*) ≡ *i′* **and**
lev-K′: *get-level* (*trail S*) *K′* = *Suc i′* **and**
U: *U* ∼ *cons-trail* (*Propagated L′ D′*)

(*reduce-trail-to M1′*
(*add-learned-cls D′*
(*update-backtrack-lvl i′*
(*update-conflicting None S*))))
using *bt-U lev* **by** (*elim backtrackE*) (*force simp*: $cdcl_W$*-M-level-inv-def*)+
obtain *c* **where** *M*: *trail S* = *c* @ *M2* @ *Decided K* # *M1*
using *decomp* **by** *auto*
obtain *c′* **where** *M′*: *trail S* = *c′* @ *M2′* @ *Decided K′* # *M1′*
using *decomp′* **by** *auto*
have *n-d*: *no-dup* (*trail S*) **and** *bt*: *backtrack-lvl S* = *count-decided* (*trail S*)
using *lev* **unfolding** $cdcl_W$*-M-level-inv-def* **by** *auto*
then have *atm-of K* ∉ *atm-of* ‘ *lits-of-l* (*c* @ *M2*)
by (*auto simp*: *lits-of-def M*)
then have *i* < *backtrack-lvl S*
using *lev-K* **unfolding** *M bt* **by** (*auto simp add*: *image-Un*)

have [*simp*]: *L′* = *L*
proof (*rule ccontr*)
assume ¬ *?thesis*
then have *L′* ∈# *remove1-mset L D*
using *S S′ LD LD′* **by** (*simp add*: *in-remove1-mset-neq*)
then have *get-maximum-level* (*trail S*) (*remove1-mset L D*) ≥ *backtrack-lvl S*
using ‹*get-level* (*trail S*) *L′* = *backtrack-lvl S*› *get-maximum-level-ge-get-level*
by *metis*
then show *False* **using** *i′ i* ‹*i* < *backtrack-lvl S*› **by** *auto*
qed
then have [*simp*]: *D′* = *D*
using *S S′* **by** *auto*
have [*simp*]: *i′* = *i*
using *i i′* **by** *auto*
have [*simp*]: *K* = *K′* **and** [*simp*]: *M1* = *M1′*
apply (*rule list-same-level-decomp-is-same-decomp*[*of trail S c* @ *M2 K M1*
c′ @ *M2′ K′ M1′*])
using *lev-K lev-K′ M M′ n-d* **apply** (*auto*)[*4*]
apply (*rule list-same-level-decomp-is-same-decomp*[*of trail S c* @ *M2 K M1*
c′ @ *M2′ K′ M1′*])
using *lev-K lev-K′ M M′ n-d* **apply** (*auto*)[*4*]
done
show *?thesis* **using** *T U inv decomp* **by** (*auto simp del*: *state-simp simp*: *state-eq-def*
$cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-decomp*)
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 E D* **where**
U: *trail U* ≠ [] **and**
tr-U: *hd-trail U* = *Propagated L E* **and**
LE: *L* ∈# *E* **and**
confl-U: *conflicting U* = *Some D* **and**

LD: $-L \in\#$ *D* **and**
get-maximum-level (*trail U*) ((*remove1-mset* ($-L$) *D*)) = *backtrack-lvl U* **and**
V: *V* $\sim$ *update-conflicting* (*Some* (*resolve-cls L D E*)) (*tl-trail U*)
using *resolve* **by** (*auto elim*!: *resolveE*)
have *inv-U*: $cdcl_W$*-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+*
have *inv-V*: $cdcl_W$*-all-struct-inv V*
using *mono-rtranclp*[*of resolve* $cdcl_W$] *inv-U resolve* $cdcl_W$*.simps* $cdcl_W$*-all-struct-inv-inv* $cdcl_W$*-bj.resolve* $cdcl_W$*-o.simps* **by** *blast*
have
S: *init-clss U* = *init-clss S*
learned-clss U = *learned-clss S*
backtrack-lvl U = *backtrack-lvl S*
backtrack-lvl V = *backtrack-lvl S*
conflicting S = *Some D*
using *rtranclp-skip-state-decomp*[*OF skip*] *U confl-U V*
by (*auto simp del*: *state-simp simp*: *state-eq-def*)
obtain M_0 **where**
tr-S: *trail S* = M_0 @ *trail U* **and**
nm: $\forall m \in set\ M_0.\ \neg is\text{-}decided\ m$
using *rtranclp-skip-state-decomp*[*OF skip*] **by** *blast*

obtain *K' i' M1' M2' L' D'* **where**
S': *conflicting S* = *Some D'* **and**
LD': $L' \in\#$ *D'* **and**
decomp': (*Decided K'* # *M1'*, *M2'*) $\in$ *set* (*get-all-ann-decomposition* (*trail S*)) **and**
lev-l: *get-level* (*trail S*) *L'* = *backtrack-lvl S* **and**
lev-l-D: *get-level* (*trail S*) *L'* = *get-maximum-level* (*trail S*) *D'* **and**
i': *get-maximum-level* (*trail S*) (*remove1-mset L' D'*) $\equiv$ *i'* **and**
lev-K': *get-level* (*trail S*) *K'* = *Suc i'* **and**
R: *T* $\sim$ *cons-trail* (*Propagated L' D'*)
(*reduce-trail-to M1'*
(*add-learned-cls D'*
(*update-backtrack-lvl i'*
(*update-conflicting None S*))))
using *bt* **by** (*elim backtrackE*) *metis*
obtain *c* **where** *M*: *trail S* = *c* @ *M2'* @ *Decided K'* # *M1'*
using *get-all-ann-decomposition-exists-prepend*[*OF decomp'*] **by** *auto*
have *i'* < *backtrack-lvl S*
using *count-decided-ge-get-level*[*of K' trail S*] *inv*
unfolding $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-def lev-K'*
by *linarith*

have *U*: *trail U* = *Propagated L E* # *trail V*
using *tr-S U S V tr-U* ‹*trail U* $\neq$ []› **by** (*cases trail U*) (*auto simp*: *lits-of-def*)
have *DD'*[*simp*]: *D'* = *D*
using *U S' S* **by** *auto*
have [*simp*]: *L'* = $-L$
proof (*rule ccontr*)
assume $\neg$ *?thesis*
then have $-L \in\#$ *remove1-mset L' D'*
using *DD' LD' LD* **by** (*simp add*: *in-remove1-mset-neq*)
moreover

have *M'*: *trail S* = M_0 @ *Propagated L E # trail V*
using *tr-S* **unfolding** *U* **by** *auto*
have *no-dup* (*trail S*)
using *inv U* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-def* **by** *auto*
then have *atm-L-notin-M*: *atm-of L* ∉ *atm-of* ‘ (*lits-of-l* (*trail V*))
using *M' U S* **by** (*auto simp*: *lits-of-def*)
have *get-lev-L*:
get-level(*Propagated L E # trail V*) *L* = *backtrack-lvl V*
using *inv-V* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-M-level-inv-def* **by** *auto*
have *atm-of L* ∉ *atm-of* ‘ (*lits-of-l* (*rev* M_0))
using ‹*no-dup* (*trail S*)› *M'* **by** (*auto simp*: *lits-of-def*)
then have *get-level* (*trail S*) *L* = *backtrack-lvl S*
using *get-lev-L S* **unfolding** *M'* **by** *auto*
ultimately
have *get-maximum-level* (*trail S*) (*remove1-mset L' D'*) ≥ *backtrack-lvl S*
by (*metis get-maximum-level-ge-get-level get-level-uminus*)
then show *False*
using ‹*i'* < *backtrack-lvl S*› *i'* **by** *auto*
qed
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 E # trail V* ⊨*as CNot D'*
using *U confl-U* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-conflicting-def* **by** *auto*
then have ∀ *L'*∈# (*remove1-mset L' D'*) .
atm-of L' ∈ *atm-of* ‘ *lits-of-l* (*Propagated L E # trail U*)
using *U atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set in-CNot-implies-uminus*(*2*)
by (*fastforce dest*: *in-diffD*)
then have ∀ *L'*∈# (*remove1-mset L' D'*) .
atm-of L' ∉ *atm-of* ‘ *lits-of-l* M_0
using ‹*no-dup* (*trail S*)› **unfolding** *tr-S U* **by** (*fastforce simp*: *lits-of-def image-image*)
then have *get-maximum-level* (*trail S*) (*remove1-mset L' D'*) = *backtrack-lvl S*
using *get-maximum-level-skip-un-decided-not-present*[*of remove1-mset L' D'*
M_0 *trail U*] *tr-S nm U*
‹*get-maximum-level* (*trail U*) ((*remove1-mset* (− *L*) *D*)) = *backtrack-lvl U*›
by (*auto simp*: *S*)
then show *False*
using *i'* ‹*i'* < *backtrack-lvl S*› **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-or-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 skip-or-resolve.simps*)

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* ∨ *?R S W* ∨ *?SB S W* ∨ *?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* ∨ *?S S W* **using** *IH* **by** *blast*

have *cdcl*$_W$** *S W* **using** *mono-rtranclp*[*of cdcl*$_W$*-bj cdcl*$_W$] *st* **by** *blast*
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 skip-or-resolve.cases)
      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 skip-or-resolve.intros)
           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 $\wedge$ *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 $(\exists\, vr19\ vr16\ vr17\ vr18.\ vr19\ (vr16{::}'st)\ vr17 \wedge vr19^{**}\ vr17\ vr18$
$\wedge \neg\ vr19^{**}\ vr16\ vr18)$
$\vee \neg$ (*skip-or-resolve* T U $\wedge$ *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 $\wedge$ *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 skip-or-resolve.intros*)
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 ∼ V ∨ $cdcl_W$-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 ∼ 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 (*meson backtrack-state-eq-compatible $cdcl_W$-bj.simps resolve-state-eq-compatible*
skip-state-eq-compatible state-eq-ref)
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*
have *cdcl*$_W$*-bj* U U'
by (*meson* ⟨*skip*$^{++}$ T U⟩ *backtrack bt inv-T rtranclp-skip-backtrack-backtrack-end tranclp-into-rtranclp*)
then have *cdcl*$_W$*-bj*$^{++}$ U V
using ⟨*cdcl*$_W$*-bj*** U' V⟩ **by** *auto*
then show *?thesis*
by (*meson 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*

2.2.3 CDCL FW

inductive $cdcl_W$*-merge-restart* :: $'st \Rightarrow 'st \Rightarrow bool$ **where**
fw-r-propagate: *propagate* $S\ S' \Longrightarrow cdcl_W$*-merge-restart* $S\ S'$ |
fw-r-conflict: *conflict* $S\ T \Longrightarrow$ *full* $cdcl_W$*-bj* $T\ U \Longrightarrow cdcl_W$*-merge-restart* $S\ U$ |
fw-r-decide: *decide* $S\ S' \Longrightarrow cdcl_W$*-merge-restart* $S\ S'$|
fw-r-rf: $cdcl_W$*-rf* $S\ S' \Longrightarrow cdcl_W$*-merge-restart* $S\ S'$

lemma *rtranclp-*$cdcl_W$*-bj-rtranclp-*$cdcl_W$:
$cdcl_W$*-bj*** $S\ T \Longrightarrow cdcl_W{}^{**}\ S\ T$
using *mono-rtranclp*[*of* $cdcl_W$*-bj* $cdcl_W$] **by** *blast*

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 $cdcl_W{}^{**}\ T\ U$ **using** *rtranclp-*$cdcl_W$*-bj-rtranclp-*$cdcl_W$ **by** *blast*
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* $\vee$ *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 elim*: *decideE propagateE conflictE forgetE restartE*)
}
then show *?case* **by** (*cases conflicting* U) *fastforce+*
qed (*auto simp add*: $cdcl_W$*-rf.simps elim*: *propagateE decideE restartE forgetE*)

inductive $cdcl_W$*-merge* :: $'st \Rightarrow 'st \Rightarrow bool$ **where**
fw-propagate: *propagate* $S\ S' \Longrightarrow cdcl_W$*-merge* $S\ S'$ |
fw-conflict: *conflict* $S\ T \Longrightarrow$ *full* $cdcl_W$*-bj* $T\ U \Longrightarrow cdcl_W$*-merge* $S\ U$ |
fw-decide: *decide* $S\ S' \Longrightarrow cdcl_W$*-merge* $S\ S'$|
fw-forget: *forget* $S\ S' \Longrightarrow cdcl_W$*-merge* $S\ S'$

lemma $cdcl_W$*-merge-*$cdcl_W$*-merge-restart*:
$cdcl_W$*-merge* $S\ T \Longrightarrow cdcl_W$*-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*** $S\ T \Longrightarrow cdcl_W$*-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 $T \Longrightarrow$ ${cdcl_W}^{**}$ 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 $T \Longrightarrow$ ${cdcl_W}^{**}$ S T
using *rtranclp-mono*[*of* $cdcl_W$*-merge* ${cdcl_W}^{**}$] $cdcl_W$*-merge-rtranclp-*$cdcl_W$ **by** *auto*

lemmas *rulesE* =
skipE resolveE backtrackE propagateE conflictE decideE restartE forgetE

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 (λS T. $cdcl_W$*-all-struct-inv* S $\wedge$ $cdcl_W$*-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 (λS 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 *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
using *bt inv backtrack-no-*$cdcl_W$*-bj* **unfolding** *full1-def* **by** *blast*

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* **and** *conflicting* V = *None*
by (*auto elim*: *propagateE*)
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 and conflicting V ≠ None
    by (auto elim!: conflictE simp del: state-simp simp: state-eq-def)
  ultimately show ?thesis using IH by auto
next
  case other
  then show ?thesis
    proof cases
      case decide
      then show ?thesis using IH cdcl_W-merge-restart.fw-r-decide[of U V] by (auto elim: decideE)
    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
        have conflicting V ≠ None ∧ conflicting U ≠ None
          using ‹skip-or-resolve U V›
          by (auto simp: skip-or-resolve.simps state-eq-def elim!: skipE resolveE
            simp del: state-simp)
        then have ?thesis
          by (metis (full-types) IH f1 rtranclp-trans tranclp-into-rtranclp)
      }
      moreover {
        assume backtrack U V
        then have conflicting U ≠ None by (auto elim: backtrackE)
        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: backtrackE
            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 elim*: *restartE forgetE*)
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 (*auto elim*: *skipE resolveE backtrackE conflictE decideE restartE*)
qed

lemma $cdcl_W$*-merge-restart-no-step-*$cdcl_W$*-bj*:
assumes
$cdcl_W$*-merge-restart S T*
shows *no-step* $cdcl_W$*-bj T*
using *assms*
by (*induction rule*: $cdcl_W$*-merge-restart.induct*)
(*force simp*: $cdcl_W$*-bj.simps* $cdcl_W$*-rf.simps* $cdcl_W$*-merge-restart.simps full-def*
elim!: *rulesE*)+

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* **unfolding** *rtranclp-unfold*
apply (*elim disjE*)
apply (*force simp*: $cdcl_W$*-bj.simps* $cdcl_W$*-rf.simps elim*!: *rulesE*)
by (*auto simp*: *tranclp-unfold-end simp*: $cdcl_W$*-merge-restart-no-step-*$cdcl_W$*-bj*)

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* ⟷ *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'* ⟹ $cdcl_W$*-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'* ⟹ *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 ≠ *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* ⟷ *full* $cdcl_W$*-merge-restart* (*init-state N*) *V*
by (*rule conflicting-true-full-*$cdcl_W$*-iff-full-*$cdcl_W$*-merge*) *auto*

2.2.4 FW with strategy

The intermediate step

```
inductive cdcl_W-s' :: 'st ⇒ 'st ⇒ bool where
conflict': full1 cdcl_W-cp S S' ⟹ cdcl_W-s' S S' |
decide': decide S S' ⟹ no-step cdcl_W-cp S ⟹ full cdcl_W-cp S' S'' ⟹ cdcl_W-s' S S'' |
bj': full1 cdcl_W-bj S S' ⟹ no-step cdcl_W-cp S ⟹ full cdcl_W-cp S' S'' ⟹ 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 cdcl_W-cp.simps elim!: rulesE)
  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 elim: rulesE)
      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''
```

$\wedge$ *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*$_W$*-bj*** T T''
using *local.bj st* **by** *auto*
then have *cdcl*$_W$** T T''
using *rtranclp-cdcl*$_W$*-bj-rtranclp-cdcl*$_W$ **by** *blast*
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
using ‹*cdcl*$_W$*-bj*$^{++}$ T T''› **by** (*blast dest*: *tranclpD*)
{ 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 elim*: *rulesE*)
}
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* ⇒ *'st* **where**
f1: ∀ *x2*. (∃ *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
∨ (∃ U'. *full1* $cdcl_W$*-bj* U U' ∧ (∀ U''. *full* $cdcl_W$*-cp* U' U'' ⟶ *full* $cdcl_W$*-cp* T' 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 local.bj rtranclp.simps rtranclp-*$cdcl_W$*-bj-rtranclp-*$cdcl_W$ *st*)
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
using ‹$cdcl_W$*-bj*$^{++}$ T T''› **by** (*blast dest*: *tranclpD*)
{ **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 elim*: *rulesE*)
}
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* ⇒ *'st* **where**
f1: ∀ *x2*. (∃ *v3*. $cdcl_W$*-cp x2 v3*) = $cdcl_W$*-cp x2* (*ss x2*)
by *moura*
have ¬ $cdcl_W$*-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
    ∨ (∃ U'. full1 cdcl_W-bj U U' ∧ (∀ U''. full cdcl_W-cp U' U'' ⟶ 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 elim: rulesE)
          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
$\vee$ ($\exists$ U' U''. $cdcl_W$*-s′* S U'' $\wedge$ *full1* $cdcl_W$*-bj* U U' $\wedge$ *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$_W$-stgy*** *S ss* $\wedge$ *cdcl$_W$-stgy ss T*
        by (*metis* (*full-types*) *rtranclp.simps st*)
      obtain *ssa* :: *'st* **where**
        ssa: *cdcl$_W$-cp T ssa*
        using *conflict'* **by** (*metis* (*no-types*) *full1-def tranclpD*)
      have $\forall$ *s*. $\neg$ *full cdcl$_W$-cp s T*
        by (*meson ssa full-def*)
      then have *S* = *T*
        by (*metis* (*full-types*) *f3 ssa cdcl$_W$-stgy.cases full1-def*)
      then show *?thesis*
        using *f2* **by** *blast*
    next
      case (*other' U*) **note** *o* = *this*(*1*) **and** *n-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$_W$-s'*** *S T*
            using *IH* **by** (*auto elim*: *rulesE*)
          then show *?thesis*
            by (*meson decide decide' full n-s rtranclp.rtrancl-into-rtrancl*)
        next
          case *backtrack*
          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'*
              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-S'* = *this*(*1*) **and** *bj-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*
elim: *rulesE*)
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 elim*: *rulesE*)
then have *confl-V*: *conflicting* $V \neq$ *None*
using *skip* **by** (*auto elim*!: *rulesE simp del*: *state-simp simp*: *state-eq-def*)
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'
show *?thesis* **using** s' *confl-V skip* **by** *force*
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*)
then show *?thesis* **using** *S-S'* *confl-V* **by** *auto*
qed
next
case *resolve*
then have [*simp*]: $U = V$
using *full* **unfolding** *full-def rtranclp-unfold*
by (*auto elim*!: *rulesE dest*!: *tranclpD*
simp del: *state-simp simp*: *state-eq-def* $cdcl_W$*-cp.simps*)
have *confl-V*: *conflicting* $V \neq$ *None*
using *resolve* **by** (*auto elim*!: *rulesE simp del*: *state-simp simp*: *state-eq-def*)

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*
then show *?thesis* **using** s' *confl-V* **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*)

          then show *?thesis* using *confl-V 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 ¬ *?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 ¬ *?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* ⇒ *'st* ⇒ (*'st* ⇒ *'st* ⇒ *bool*) ⇒ *'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*. ¬ *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* ∧ *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'*
      have *no-step* $cdcl_W$*-s' T*
        using ‹*full* $cdcl_W$*-stgy S T*› **unfolding** *full-def*
        by (*meson* $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-s'E* $cdcl_W$*-stgy.conflict'*
          $cdcl_W$*-then-exists-*$cdcl_W$*-stgy-step inv-T n-step-*$cdcl_W$*-stgy-iff-no-step-*$cdcl_W$*-cl-*$cdcl_W$*-o*)
      then show *?thesis*
        using *s'* **unfolding** *full-def* **by** *blast*
    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
elim: rulesE)
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 ‹full cdcl$_W$-cp T T› ‹full cdcl$_W$-stgy S T› unfolding full-def
by (metis cdcl$_W$-all-struct-inv-def cdcl$_W$-then-exists-cdcl$_W$-stgy-step
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 ∃ 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 ∃ 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 ⟹ conflicting S = Some D ⟹ S = T
using rtranclpD tranclpD by fastforce

inductive cdcl$_W$-merge-cp :: 'st ⇒ 'st ⇒ bool where
conflict': conflict S T ⟹ full cdcl$_W$-bj T U ⟹ cdcl$_W$-merge-cp S U |

propagate': $propagate^{++}$ *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*
unfolding *full1-def* **by** (*metis rtranclp-unfold* $cdcl_W$*-cp-conflicting-not-empty option.exhaust*
rtranclp-$cdcl_W$*-merge-restart-no-step-*$cdcl_W$*-bj tranclpD*)

Full Transformation

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'* **using** *bj* **unfolding** *full-unfold* **by** *blast*
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*
using *local.bj* **unfolding** *full-unfold* **by** *blast*
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 elim*: *rulesE*)
then have $cdcl_W$-*merge-cp*** S U **using** *IH* **by** (*auto elim*: *rulesE*
simp del: *state-simp simp*: *state-eq-def*)
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 intro*: $cdcl_W$-*merge-cp.intros*)
moreover have *conflicting* V = *None*
using *propa* **unfolding** *tranclp-unfold-end* **by** (*auto elim*: *rulesE*)
ultimately show *?thesis* **using** ‹$cdcl_W$-*merge-cp*** S U› **by** (*auto elim*!: *rulesE*
simp del: *state-simp simp*: *state-eq-def*)
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 intro*: $cdcl_W$-*merge-cp.intros*)
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* ≠ *None*
using *full-bj* **unfolding** *full1-def* **by** (*fastforce dest*!: *tranclpD simp*: $cdcl_W$*-bj.simps elim*: *rulesE*)
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** (*blast intro*: $cdcl_W$*-merge-cp.intros*)
moreover have *conflicting V* = *None*
using *propa* **unfolding** *tranclp-unfold-end* **by** (*auto elim*: *rulesE*)
ultimately show *?thesis* **using** *S-U'* **by** (*auto elim*: *rulesE*
simp del: *state-simp simp*: *state-eq-def*)
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** (*blast intro*: $cdcl_W$*-merge-cp.intros*)
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 intro*: $cdcl_W$*-merge-cp.intros*)
      then have *conflict S T*
        using *local.conflict' tranclp-*$cdcl_W$*-cp-propagate-with-conflict-or-not*[*of S T*]
        local.conflict'-without-decide **unfolding** *full1-def rtranclp-unfold*
        by (*metis 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*
          elim: *rulesE*)
    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*
elim: *rulesE*)+

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 elim*: *rulesE*)
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** *simp*
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 *simp*
then show *?thesis* **unfolding** *full-unfold full1-def tranclp-unfold-begin* **by** *blast*
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 elim*: *rulesE*)
then show *?thesis*
using *conflicting-true-full1-cdcl*$_W$*-merge-cp-iff-full1-cdcl*$_W$*-s'-without-decode fw inv* **by** *simp*
qed

inductive *cdcl*$_W$*-merge-stgy* **where**
fw-s-cp[*intro*]: *full1 cdcl*$_W$*-merge-cp S T* ⟹ *cdcl*$_W$*-merge-stgy S T* |
fw-s-decide[*intro*]: *decide S T* ⟹ *no-step cdcl*$_W$*-merge-cp S* ⟹ *full cdcl*$_W$*-merge-cp T U*
⟹ *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* ⟹ *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* ⟹ *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** blast
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 elim: rulesE)
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 intro*: *cdcl*$_W$*-merge-cp.intros*)
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 elim*: *rulesE*)
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*) *cp*
by (*metis* (*full-types*) $cdcl_W$*-merge-cp.propagate' rtranclp.rtrancl-into-rtrancl*
rtranclp-unfold)
qed
next
case *cp-confl*
then show *?thesis* **using** *conflict'* **unfolding** *full1-def* **by** (*fastforce dest*!: *tranclpD*
elim!: *rulesE*)
qed
next
case (*decide' V'*)
then have *conf-V*: *conflicting V = None*
by (*auto elim*: *rulesE*)
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 elim*: *rulesE*)
have *full*: *full1* $cdcl_W$*-cp V' W* ∨ (*V' = W* ∧ *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'*
⟨*full1* $cdcl_W$*-cp V' W* ∨ *V' = W* ∧ *no-step* $cdcl_W$*-cp W*⟩ **unfolding** *full1-def*
by (*metis 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 *auto*
  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 elim: rulesE*)
      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** (*blast intro: cdcl*$_W$*-merge-cp.intros*)
      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** *auto*
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* ∨ (*V′* = *W* ∧ *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 elim*: *rulesE*)
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 (*blast intro*: $cdcl_W$*-merge-cp.intros*)
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 elim*!: *rulesE*
simp del: *state-simp simp*: *state-eq-def*)
next
case *cp-confl*
then show *?thesis* **using** *decide′* **apply** − **by** (*intro HOL.disjI2*) (*fastforce elim*: *rulesE*
simp del: *state-simp simp*: *state-eq-def*)
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* *elim*: *rulesE*)
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* *elim*: *rulesE*)
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 $\neq$ None*
then show *?thesis*
using ‹*cdcl$_W$-merge-stgy** R V'*› ‹*V' = W*› **by** (*metis* ‹*cdcl$_W$-merge-cp U V'*› *conflictE conflicting-not-true-rtranclp-cdcl$_W$-merge-cp-no-step-cdcl$_W$-bj*

dec-confl(*5*) *r-into-rtranclp*)
qed
next
case *propa*
moreover then have $cdcl_W$*-merge-cp* V' W **by** (*blast intro*: $cdcl_W$*-merge-cp.intros*)
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*) ‹¬ *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 elim*: *rulesE*)
have *no-step* $cdcl_W$*-merge-cp* V'
using *confl* **by** (*auto simp*: *full1-def full-def* $cdcl_W$*-merge-cp.simps*
dest!: *tranclpD elim*: *rulesE*)
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 elim*: *rulesE*)
ultimately have $cdcl_W$*-merge-stgy*** R V' **by** *auto*
{ **fix** *ss* :: *'st*
have $cdcl_W$*-merge-cp* U W

using V'-W ‹$cdcl_W$-*merge-cp* U V'› **by** *blast*
then have ¬ $cdcl_W$-*bj* W *ss*
by (*meson conflicting-not-true-rtranclp-*$cdcl_W$*-merge-cp-no-step-*$cdcl_W$*-bj cp-confl*(*1*) *rtranclp.rtrancl-into-rtrancl step.prems*)
then have $cdcl_W$-*merge-stgy*** R W ∧ *conflicting* W = *None* ∨ $cdcl_W$-*merge-stgy*** R W ∧ ¬ $cdcl_W$-*bj* W *ss*
using V'-W ‹$cdcl_W$-*merge-stgy*** R V'› **by** *presburger* **}**
then show *?thesis*
by *presburger*
qed
next
case *propa*
moreover then have $cdcl_W$-*merge-cp* V' W
by (*blast intro*: $cdcl_W$-*merge-cp.intros*)
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**
ss: $cdcl_W$-s'^{**} S T ∨ $cdcl_W$-*cp* T *ss*
using *a1 step.IH* **by** *blast*−
obtain *ssa* :: *'st* ⇒ *'st* **where**
f3: ∀ *s sa sb*. (¬ *decide s sa* ∨ $cdcl_W$-*cp s* (*ssa s*) ∨ ¬ *full* $cdcl_W$-*cp sa sb*) ∨ $cdcl_W$-*s' s sb*
using $cdcl_W$-*s'.decide'* **by** *moura*

have $\forall\, s\ sa.\ \neg\ cdcl_W\text{-}s'\ s\ sa \lor full1\ cdcl_W\text{-}cp\ s\ sa \lor$
$(\exists\, sb.\ decide\ s\ sb \land no\text{-}step\ cdcl_W\text{-}cp\ s \land full\ cdcl_W\text{-}cp\ sb\ sa) \lor$
$(\exists\, sb.\ full1\ cdcl_W\text{-}bj\ s\ sb \land no\text{-}step\ cdcl_W\text{-}cp\ s \land full\ cdcl_W\text{-}cp\ sb\ sa)$
by (*metis* $cdcl_W$*-s′E*)
then have $\exists\, s.\ cdcl_W\text{-}s'^{**}\ S\ s \land cdcl_W\text{-}s'\ s\ V$
using *f3 ss f2* **by** (*metis dec full1-is-full n-s-S rtranclp-unfold*)
then show *?thesis*
by *force*
qed
next
case (s'*-st* T') **note** $s'\text{-}T' = this(1)$ **and** $st = this(2)$
have $cdcl_W\text{-}s'^{**}\ S\ T'$
using $s'\text{-}T'$
proof *cases*
case *conflict′*
then have $cdcl_W\text{-}s'\ S\ T'$
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 $cdcl_W\text{-}s'\ S\ T$
using *dec* $cdcl_W$*-s′.decide′ n-s-S* **by** (*simp add*: *full-unfold*)
then show *?thesis* **using** *decide′* $s'\text{-}T'$ **by** *auto*
next
case *bj′*
then have *False*
using *dec* **unfolding** *full1-def* **by** (*fastforce dest*!: *tranclpD simp*: $cdcl_W$*-bj.simps*
elim: *rulesE*)
then show *?thesis* **by** *fast*
qed
then show *?thesis* **using** *s′ st* **by** *auto*
qed
next
case *base*
then have *full* $cdcl_W\text{-}cp\ T\ T$
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
$cdcl_W\text{-}merge\text{-}stgy^{**}\ R\ V$ **and**
inv: $cdcl_W$*-all-struct-inv R*
shows $cdcl_W\text{-}s'^{**}\ R\ V$
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*
have $\bigwedge s.\ \neg$ *full* $cdcl_W$*-merge-cp s S*
using *fw-s-cp* **unfolding** *full-def full1-def* **by** (*metis tranclp-unfold-begin*)
then have $S = R$
using *fw-s-cp* **unfolding** *full1-def* **by** (*metis* $cdcl_W$*-cp.conflict'* $cdcl_W$*-cp.propagate'* $cdcl_W$*-merge-cp.cases tranclp-unfold-begin inv st rtranclp-*$cdcl_W$*-merge-stgy'-no-step-*$cdcl_W$*-cp-or-eq*)
then have *full1* $cdcl_W$*-s'-without-decide R T*
using *inv local.fw-s-cp*
by (*blast intro*: *conflicting-true-full1-*$cdcl_W$*-merge-cp-imp-full1-*$cdcl_W$*-s'-without-decode*)
then show *?thesis* **unfolding** *full1-def*
by (*metis* (*no-types*) *rtranclp-*$cdcl_W$*-s'-without-decide-rtranclp-*$cdcl_W$*-s' rtranclp-unfold*)
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 elim*: *rulesE*)
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: $\bigwedge s\ sa.\ \neg$ $cdcl_W$*-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: $\bigwedge p\ s\ sa.\ \neg\ p^{++}\ s\ sa \lor p\ s\ (ssb\ p\ s\ sa)$
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*))

$\wedge$ ($\neg$ *cdcl*$_W$*-all-struct-inv s* $\vee$ $\neg$ *cdcl*$_W$*-o s sb* $\vee$ *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.* $\neg$ *cdcl*$_W$*-o R s*
using *s′ inv* **by** *blast*
have *ff5*: $\bigwedge$*s.* $\neg$ *cdcl*$_W$*-cp*$^{++}$ *R s*
using *ff3 ff2 s′* **by** (*metis inv*)
have $\bigwedge$*s.* $\neg$ *cdcl*$_W$*-bj*$^{++}$ *R s*
using *ff4 ff2* **by** (*metis bj*)
then have $\bigwedge$*s.* $\neg$ *cdcl*$_W$*-s′-without-decide R s*
using *ff5* **by** (*simp add*: *cdcl*$_W$*-s′-without-decide.simps full1-def*)
then have $\neg$ *cdcl*$_W$*-s′-without-decide*$^{++}$ *R ss*
using *ff2* **by** *blast*
then have $\neg$ *full1 cdcl*$_W$*-s′-without-decide R ss*
by (*simp add*: *full1-def*)
then have $\neg$ *cdcl*$_W$*-merge-stgy R ss*
using *ff4 ff1 conflicting-true-full1-cdcl*$_W$*-merge-cp-imp-full1-cdcl*$_W$*-s′-without-decode inv*
by *blast* **}**
then show *?thesis*
by *fastforce*
qed
end

Termination and full Equivalence

We will discharge the assumption later using NOT's proof of termination.

locale *conflict-driven-clause-learning*$_W$*-termination* =
conflict-driven-clause-learning$_W$ +
assumes *wf-cdcl*$_W$*-merge-inv*: *wf* {(*T*, *S*). *cdcl*$_W$*-all-struct-inv S* $\wedge$ *cdcl*$_W$*-merge S T*}
begin

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-inv*]
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 *wf-cdcl*$_W$*-merge-cp*:
wf{(*T*, *S*). *cdcl*$_W$*-all-struct-inv S* $\wedge$ *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* $\wedge$ *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* (λ*S T*. *cdcl*$_W$*-all-struct-inv S* $\wedge$ *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: (λ*S T*. *cdcl*$_W$*-all-struct-inv S* $\wedge$ *cdcl*$_W$*-merge-cp S T*)** *R S* **and**
n-s: *no-step* (λ*S T*. *cdcl*$_W$*-all-struct-inv S* $\wedge$ *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'*
proof −
obtain *R'* **where**
cdcl$_W$-merge-cp R R'
using *inv* **unfolding** *cdcl$_W$-all-struct-inv-def* **by** (*meson confl cdcl$_W$-s'-without-decide.simps conflict' conflicting-true-no-step-cdcl$_W$-merge-cp-no-step-s'-without-decide*)
then show *?thesis*
using *that* **by** (*metis cdcl$_W$-merge-cp-obtain-normal-form full-unfold inv*)
qed
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** *auto*
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** *auto*

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 elim*: *rulesE*)
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 elim*: *rulesE*)
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 elim*: *rulesE*)
ultimately show *?thesis*
using *conflicting-not-true-rtranclp-cdcl$_W$-merge-cp-no-step-cdcl$_W$-bj*
unfolding *full-def* **by** *meson*
qed
qed

end

end
theory *CDCL-WNOT*
imports *CDCL-NOT CDCL-W-Termination CDCL-W-Merge*
begin

2.3 Link between Weidenbach's and NOT's CDCL

2.3.1 Inclusion of the states

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

fun *convert-ann-lit-from-W* **where**
convert-ann-lit-from-W (Propagated L -) = Propagated L () |
convert-ann-lit-from-W (Decided L) = Decided L

abbreviation *convert-trail-from-W* ::
('*v*, '*mark*) *ann-lits*
⇒ ('*v*, *unit*) *ann-lits* **where**
convert-trail-from-W ≡ map convert-ann-lit-from-W

lemma *lits-of-l-convert-trail-from-W[simp]*:

lits-of-l (*convert-trail-from-W M*) = *lits-of-l M*
by (*induction rule*: *ann-lit-list-induct*) *simp-all*

lemma *lit-of-convert-trail-from-W*[*simp*]:
lit-of (*convert-ann-lit-from-W L*) = *lit-of L*
by (*cases L*) *auto*

lemma *no-dup-convert-from-W*[*simp*]:
no-dup (*convert-trail-from-W M*) ⟷ *no-dup M*
by (*auto simp*: *comp-def*)

lemma *convert-trail-from-W-true-annots*[*simp*]:
convert-trail-from-W M ⊨*as C* ⟷ *M* ⊨*as C*
by (*auto simp*: *true-annots-true-cls image-image lits-of-def*)

lemma *defined-lit-convert-trail-from-W*[*simp*]:
defined-lit (*convert-trail-from-W S*) *L* ⟷ *defined-lit S L*
by (*auto simp*: *defined-lit-map image-comp*)

The values *0* and {#} are dummy values.

consts *dummy-cls* :: *'cls*
fun *convert-ann-lit-from-NOT*
:: (*'v*, *'mark*) *ann-lit* ⇒ (*'v*, *'cls*) *ann-lit* **where**
convert-ann-lit-from-NOT (*Propagated L -*) = *Propagated L dummy-cls* |
convert-ann-lit-from-NOT (*Decided L*) = *Decided L*

abbreviation *convert-trail-from-NOT* **where**
convert-trail-from-NOT ≡ *map convert-ann-lit-from-NOT*

lemma *undefined-lit-convert-trail-from-NOT*[*simp*]:
undefined-lit (*convert-trail-from-NOT F*) *L* ⟷ *undefined-lit F L*
by (*induction F rule*: *ann-lit-list-induct*) (*auto simp*: *defined-lit-map*)

lemma *lits-of-l-convert-trail-from-NOT*:
lits-of-l (*convert-trail-from-NOT F*) = *lits-of-l F*
by (*induction F rule*: *ann-lit-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-lit-list-induct*) *auto*

lemma *convert-trail-from-W-convert-lit-from-NOT*[*simp*]:
convert-ann-lit-from-W (*convert-ann-lit-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-lit-from-NOT*[*iff*]:
lit-of (*convert-ann-lit-from-NOT L*) = *lit-of L*
by (*cases L*) *auto*

sublocale $state_W \subseteq$ *dpll-state-ops*
$\lambda S.$ *convert-trail-from-W* (*trail* S)
clauses
$\lambda L\ S.$ *cons-trail* (*convert-ann-lit-from-NOT* L) S
$\lambda S.$ *tl-trail* S
$\lambda C\ S.$ *add-learned-cls* $C\ S$
$\lambda C\ S.$ *remove-cls* $C\ S$
by *unfold-locales*

sublocale $state_W \subseteq$ *dpll-state*
$\lambda S.$ *convert-trail-from-W* (*trail* S)
clauses
$\lambda L\ S.$ *cons-trail* (*convert-ann-lit-from-NOT* L) S
$\lambda S.$ *tl-trail* S
$\lambda C\ S.$ *add-learned-cls* $C\ S$
$\lambda C\ S.$ *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 $conflict\text{-}driven\text{-}clause\text{-}learning_W \subseteq cdcl_{NOT}$-*merge-bj-learn-ops*
$\lambda S.$ *convert-trail-from-W* (*trail* S)
clauses
$\lambda L\ S.$ *cons-trail* (*convert-ann-lit-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*
λ- $S.$ *conflicting* S = *None*
$\lambda C\ C'\ L'\ S\ T.$ *backjump-l-cond* $C\ C'\ L'\ S\ T$
$\wedge$ *distinct-mset* $(C' + \{\#L'\#\}) \wedge \neg$*tautology* $(C' + \{\#L'\#\})$
by *unfold-locales*

thm $cdcl_{NOT}$-*merge-bj-learn-proxy.axioms*
sublocale $conflict\text{-}driven\text{-}clause\text{-}learning_W \subseteq cdcl_{NOT}$-*merge-bj-learn-proxy*
$\lambda S.$ *convert-trail-from-W* (*trail* S)
clauses
$\lambda L\ S.$ *cons-trail* (*convert-ann-lit-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*
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* $\notin\#$ *C'*
using ‹*F* $\models$*as CNot C'*› ‹*undefined-lit F L*› *Decided-Propagated-in-iff-in-lits-of-l in-CNot-implies-uminus*(*2*) **by** *fast*
then have *distinct-mset* (*?C'* + {#*L*#})
by (*simp add*: *distinct-mset-single-add*)
moreover
have *no-dup F*
using ‹inv_{NOT} *S*› ‹*convert-trail-from-W* (*trail S*) = *F'* @ *Decided 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-l F*)
using *distinct-consistent-interp* **by** *blast*
then have ¬ *tautology C'*
using ‹*F* $\models$*as CNot C'*› *consistent-CNot-not-tautology true-annots-true-cls* **by** *blast*
then have ¬ *tautology* (*?C'* + {#*L*#})
using ‹*F* $\models$*as CNot C'*› ‹*undefined-lit F L*› **by** (*metis CNot-remdups-mset Decided-Propagated-in-iff-in-lits-of-l 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-mm* (*clauses S*)
∪ *atm-of* ‘ *lits-of-l* (*convert-trail-from-W* (*trail S*))
using ‹*convert-trail-from-W* (*trail S*) = *F'* @ *Decided K* # *F*›
‹*atm-of L* ∈ *atms-of-mm* (*clauses S*) ∪ *atm-of* ‘ *lits-of-l* (*F'* @ *Decided K* # *F*)› **by** *auto*
have *f4*: *clauses S* $\models$*pm remdups-mset C'* + {#*L*#}
by (*metis* (*no-types*) ‹*L* $\notin\#$ *C'*› ‹*clauses S* $\models$*pm C'* + {#*L*#}› *remdups-mset-singleton-sum*(*2*) *true-clss-cls-remdups-mset union-commute*)
have *F* $\models$*as CNot* (*remdups-mset C'*)
by (*simp add*: ‹*F* $\models$*as CNot C'*›)
have *Ex* (*backjump-l S*)
apply *standard*
apply (*rule backjump-l.intros*[*OF - f2*, *of - - -*])
using *f4 f3 f2* ‹¬ *tautology* (*remdups-mset C'* + {#*L*#})›
calculation(*2*−*5*,*9*) ‹*F* $\models$*as CNot* (*remdups-mset C'*)›
state-eq$_{NOT}$*-ref* **unfolding** *backjump-l-cond-def* **by** *blast*+
then show *?thesis*
by *blast*
qed
qed

sublocale *conflict-driven-clause-learning*$_W$ ⊆ *cdcl*$_{NOT}$*-merge-bj-learn-proxy2*
λ*S*. *convert-trail-from-W* (*trail S*)
clauses
λ*L S*. *cons-trail* (*convert-ann-lit-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}
by *unfold-locales*

sublocale *conflict-driven-clause-learning*$_W$ ⊆ *cdcl*$_{NOT}$*-merge-bj-learn*
λ*S*. *convert-trail-from-W* (*trail S*)

clauses
$\lambda L\ S.$ *cons-trail* (*convert-ann-lit-from-NOT L*) *S*
$\lambda S.$ *tl-trail S*
$\lambda C\ S.$ *add-learned-cls C S*
$\lambda C\ S.$ *remove-cls C S*
backjump-l-cond
λ- -. *True*
λ- *S*. *conflicting S* = *None* inv_{NOT}
apply *unfold-locales*
using *dpll-bj-no-dup* **apply** (*simp add*: *comp-def*)
using $cdcl_{NOT}$*.simps* $cdcl_{NOT}$*-no-dup no-dup-convert-from-W* **unfolding** inv_{NOT}*-def* **by** *blast*

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

Notations are lost while proving locale inclusion:

notation *state-eq*$_{NOT}$ (**infix** $\sim_{NOT}$ *50*)

2.3.2 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-map*[*simp*]:
reduce-trail-to (*map f M*) *S* = *reduce-trail-to M S*
by (*rule reduce-trail-to-length*) *simp*

lemma *reduce-trail-to*$_{NOT}$*-map*[*simp*]:
reduce-trail-to$_{NOT}$ (*map f M*) *S* = *reduce-trail-to*$_{NOT}$ *M S*
by (*rule reduce-trail-to*$_{NOT}$*-length*) *simp*

lemma *skip-or-resolve-state-change*:
assumes *skip-or-resolve*** *S T*
shows
$\exists M.$ *trail S* = *M* @ *trail T* $\wedge$ ($\forall m \in$ *set M*. $\neg$*is-decided m*)
clauses S = *clauses T*

backtrack-lvl S = *backtrack-lvl T*
using *assms*
proof (*induction rule*: *rtranclp-induct*)
case *base*
case *1* **show** *?case* **by** *simp*
case *2* **show** *?case* **by** *simp*
case *3* **show** *?case* **by** *simp*
next
case (*step T U*) **note** *st* = *this*(*1*) **and** *s-o-r* = *this*(*2*) **and** *IH* = *this*(*3*) **and** *IH'* = *this*(*3*−*5*)

case *2* **show** *?case* **using** *IH' s-o-r* **by** (*auto elim*!: *rulesE simp*: *skip-or-resolve.simps*)
case *3* **show** *?case* **using** *IH' s-o-r* **by** (*auto elim*!: *rulesE simp*: *skip-or-resolve.simps*)
case *1* **show** *?case*
using *s-o-r*
proof *cases*
case *s-or-r-skip*
then show *?thesis* **using** *IH* **by** (*auto elim*!: *rulesE simp*: *skip-or-resolve.simps*)
next
case *s-or-r-resolve*
then show *?thesis*
using *IH* **by** (*cases trail T*) (*auto elim*!: *rulesE simp*: *skip-or-resolve.simps*)
qed
qed

2.3.3 More lemmas conflict–propagate and backjumping

2.3.4 CDCL FW

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: *state T* = (*Propagated L* (*C* + {#*L*#}) # *M*, *N*, *U*, *k*, *None*)
by (*auto elim*: *propagate-high-levelE*)
have $propagate_{NOT}$ *S T*
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-mm* (*init-clss S*) **and**
T: *T* ∼ *cons-trail* (*Decided L*)
(*update-backtrack-lvl* (*Suc* (*backtrack-lvl S*)) *S*)

by (*auto elim*: *decideE*)
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* $state\text{-}eq_{NOT}$*-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 *C* **where**
S: *conflicting S* = *None* **and**
C-le: *C* ∈# *learned-clss S* **and**
¬(*trail S*) ⊨*asm clauses S* **and**
C ∉ *set* (*get-all-mark-of-propagated* (*trail S*)) **and**
C-init: *C* ∉# *init-clss S* **and**
T: *T* ∼ *remove-cls C S*
by (*auto elim*: *forgetE*)
have *init-clss S* ⊨*pm C*
using *inv C-le* **unfolding** $cdcl_W$*-all-struct-inv-def* $cdcl_W$*-learned-clause-def clauses-def*
by (*meson true-clss-clss-in-imp-true-clss-cls*)
then have *S-C*: *removeAll-mset C* (*clauses S*) ⊨*pm C*
using *C-init C-le* **unfolding** *clauses-def* **by** (*auto simp add*: *Un-Diff ac-simps*)
have $forget_{NOT}$ *S T*
apply (*rule* $forget_{NOT}.forget_{NOT}$)
using *S-C* **apply** *blast*
using *S* **apply** *simp*
using *C-init C-le* **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*
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 *CT* **where**
confl-T: *conflicting T* = *Some CT* **and**
CT: *CT* = C_S **and**
C_S: C_S ∈# *clauses S* **and**
tr-S-C_S: *trail S* ⊨*as CNot* C_S
using *confl* **by** (*elim conflictE*) (*auto simp del*: *state-simp simp*: *state-eq-def*)
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*) $skip\text{-}or\text{-}resolve^{**}$ *T U*
| (*bt*) *T'* **where** $skip\text{-}or\text{-}resolve^{**}$ *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 simp del*: *state-simp simp*: *skip-or-resolve.simps state-eq-def elim*!: *rulesE*)
moreover then have *no-step* $cdcl_W$*-merge U*
by (*auto simp*: $cdcl_W$*-merge.simps elim*: *rulesE*)

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* **and**
    LD: *L* ∈# *D* **and**
    M1-M2:(*Decided K* # *M1*, *M2*) ∈ *set* (*get-all-ann-decomposition* (*trail T′*)) **and**
    get-level (*trail T′*) *K* = *i*+*1*
    get-level (*trail T′*) *L* = *backtrack-lvl T′* **and**
    get-level (*trail T′*) *L* = *get-maximum-level* (*trail T′*) *D* **and**
    get-maximum-level (*trail T′*) (*remove1-mset L D*) = *i* **and**
    U: *U* ∼ *cons-trail* (*Propagated L D*)
            (*reduce-trail-to M1*
                (*add-learned-cls D*
                  (*update-backtrack-lvl i*
                    (*update-conflicting None T′*))))
    using *bt* **by** (*auto elim*: *backtrackE*)
  have [*simp*]: *clauses S* = *clauses T*
    using *confl* **by** (*auto elim*: *rulesE*)
  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 simp*: *skip-or-resolve.simps elim*: *rulesE*)
      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-mm* (*clauses S*)
    using *inv-T′ confl-T′ LD* **unfolding** $cdcl_W$*-all-struct-inv-def no-strange-atm-def*
    clauses-def
    by (*simp add*: *atms-of-def image-subset-iff*)
  obtain *M* **where** *tr-T*: *trail T* = *M* @ *trail T′*
    using *s-or-r skip-or-resolve-state-change* **by** *meson*
  obtain *M′* **where**

tr-T′: *trail T′ = M′ @ Decided K # tl* (*trail U*) **and**
tr-U: *trail U = Propagated L D # tl* (*trail U*)
using *U M1-M2 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′′ @ Decided K # tl* (*trail U*)
using *tr-T tr-T′ confl* **unfolding** *M′′-def* **by** (*auto elim*: *rulesE*)
have *init-clss T′ + learned-clss S ⊨pm D*
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 @* [*Decided K*]])
using *confl M1-M2* ‹*trail T = M @ trail T′*›
apply (*auto dest*!: *get-all-ann-decomposition-exists-prepend*
elim!: *conflictE*)
by (*rule sym*) *auto*
ultimately have [*simp*]: *trail* (*reduce-trail-to*$_{NOT}$ *M1 S*) = *M1*
using *M1-M2 confl* **by** (*subst reduce-trail-to*$_{NOT}$*-reduce-trail-convert*)
(*auto simp*: *comp-def elim*: *rulesE*)
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 *U-D*: *tl* (*trail U*) ⊨*as CNot* (*remove1-mset L D*)
by (*metis append-self-conv2 tr-U*)
have *undef-L*: *undefined-lit* (*tl* (*trail U*)) *L*
using *U M1-M2 inv-U* **unfolding** *cdcl*$_W$*-all-struct-inv-def cdcl*$_W$*-M-level-inv-def*
by (*auto simp*: *lits-of-def defined-lit-map*)
have *backjump-l S U*
apply (*rule backjump-l*[*of - - - - - L D - remove1-mset L D*])
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 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** *auto*[]
using *tr-S-*C_S **apply** *simp*

using *undef-L* **apply** *auto*[]
using *atm-L* **apply** (*simp add*: *trail-reduce-trail-to*$_{NOT}$*-add-learned-cls*)
using ‹*init-clss T′ + learned-clss S ⊨pm D*› *LD* **unfolding** *clauses-def*
apply *simp*
using *LD* **apply** *simp*
apply (*metis U-D convert-trail-from-W-true-annots*)
using *inv-T′ inv-U U confl-T′ undef-L M1-M2 LD* **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 ≡ 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* ∨ (*no-step* $cdcl_W$*-merge T* ∧ *conflicting T* ≠ *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* ∨ (*no-step* $cdcl_W$*-merge T* ∧ *conflicting T* ≠ *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* ⟶ $cdcl_{NOT}$** *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 $cdcl_{NOT}$ ≤ $cdcl_{NOT}$*-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-mm* (*clauses S*) ⊆ *atms-of-mm ?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-l* (*trail S*) ⊆ *atms-of-mm ?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*, *of T*] *atm-trail n-d*
by (*auto split*: *if-split simp*: *comp-def image-image lits-of-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* $\wedge$ $cdcl_W$*-merge S T*$\}$
apply (*rule wfP-if-measure*[*of - -* μ_{FW}])
using $cdcl_W$*-merge-*μ_{FW}*-decreasing* **by** *blast*

sublocale *conflict-driven-clause-learning*$_W$*-termination*
by *unfold-locales* (*simp add*: *wf-*$cdcl_W$*-merge*)

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* $\Rightarrow$ *'st* **where**
ff1: $\forall$ *s*. $\neg$ $cdcl_W$*-all-struct-inv s* $\vee$ $cdcl_W$*-s'-without-decide s* (*ssa s*)
$\vee$ *no-step* $cdcl_W$*-merge-cp s*
using *conflicting-true-no-step-s'-without-decide-no-step-*$cdcl_W$*-merge-cp* **by** *moura*
have ($\forall$ *p s sa*. $\neg$ *full p* (*s*::*'st*) *sa* $\vee$ p^{**} *s sa* $\wedge$ *no-step p sa*) **and**
($\forall$ *p s sa*. ($\neg$ p^{**} (*s*::*'st*) *sa* $\vee$ ($\exists$ *s*. *p sa s*)) $\vee$ *full p s sa*)
by (*meson full-def*)+
then have $\neg$ $cdcl_W$*-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* $\neq$ *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 $R = V \vee$ $cdcl_W$*-merge-stgy*$^{++}$ R V
using *fw-s-cp* **unfolding** *full-unfold fw-s-cp*
by (*metis* (*no-types*) *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* $V \neq$ *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 elim*: *rulesE*)
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* $\longleftrightarrow$ *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* {#} $\wedge$ *unsatisfiable* (*set-mset N*))
$\vee$ (*conflicting* S' = *None* $\wedge$ *trail* S' $\models$*asm N* $\wedge$ *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*
using *full full-cdcl*$_W$*-stgy-final-state-conclusive-from-init-state*
full-cdcl$_W$*-stgy-full-cdcl*$_W$*-merge no-d* **by** *presburger*
qed
end

end
theory *CDCL-W-Incremental*
imports *CDCL-W-Termination*
begin

2.4 Incremental SAT solving

locale *state*$_W$*-adding-init-clause* =
state$_W$
— functions about the state:
— getter:
trail init-clss learned-clss backtrack-lvl conflicting
— setter:
cons-trail tl-trail add-learned-cls remove-cls update-backtrack-lvl
update-conflicting

— Some specific states:
init-state
restart-state
for
trail :: $'st \Rightarrow ('v, 'v\ clause)$ *ann-lits* **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 :: $('v, 'v\ clause)$ *ann-lit* $\Rightarrow 'st \Rightarrow 'st$ **and**
tl-trail :: $'st \Rightarrow 'st$ **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 :: $'st \Rightarrow 'st$ +
fixes
add-init-cls :: $'v$ *clause* $\Rightarrow 'st \Rightarrow 'st$
assumes
trail-add-init-cls[*simp*]:
$\bigwedge st\ C.$ *trail* (*add-init-cls* C *st*) = *trail st* **and**
init-clss-add-init-cls[*simp*]:
$\bigwedge st\ C.$ *init-clss* (*add-init-cls* C *st*) = $\{\#C\#\}$ + *init-clss st*
and
learned-clss-add-init-cls[*simp*]:
$\bigwedge st\ C.$ *learned-clss* (*add-init-cls* C *st*) = *learned-clss 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**
conflicting-add-init-cls[*simp*]:
$\bigwedge st\ C.$ *conflicting* (*add-init-cls* C *st*) = *conflicting st*
begin
lemma *clauses-add-init-cls*[*simp*]:
clauses (*add-init-cls* N S) = $\{\#N\#\}$ + *init-clss* S + *learned-clss* S
unfolding *clauses-def* **by** *auto*

lemma *reduce-trail-to-add-init-cls*[*simp*]:
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 *conflicting-add-init-cls-iff-conflicting*[*simp*]:
conflicting (*add-init-cls* C S) = *None* $\longleftrightarrow$ *conflicting* S = *None*
by *fastforce*+
end

locale *conflict-driven-clause-learning-with-adding-init-clause*$_W$ =
state$_W$*-adding-init-clause*

— functions for the state:
— access functions:
trail init-clss learned-clss backtrack-lvl conflicting
— changing state:
cons-trail tl-trail add-learned-cls remove-cls update-backtrack-lvl

update-conflicting

— get state:
init-state
restart-state
— Adding a clause:
add-init-cls
for
trail :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* **and**
hd-trail :: *'st* ⇒ (*'v*, *'v clause*) *ann-lit* **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*, *'v clause*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒ *'st* **and**
add-learned-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
remove-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
update-backtrack-lvl :: *nat* ⇒ *'st* ⇒ *'st* **and**
update-conflicting :: *'v clause option* ⇒ *'st* ⇒ *'st* **and**

init-state :: *'v clauses* ⇒ *'st* **and**
restart-state :: *'st* ⇒ *'st* **and**
add-init-cls :: *'v clause* ⇒ *'st* ⇒ *'st*
begin

sublocale *conflict-driven-clause-learning*$_W$
by *unfold-locales*

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** (*intro conjI*)
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*]
using *cdcl*$_W$ *cdcl*$_W$*-stgy-not-non-negated-init-clss* **apply** *simp*
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 (*Decided 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 clause* ⇒ *'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*:
∃ *M*. *trail S* = *M* @ *trail* (*cut-trail-wrt-clause C* (*trail S*) *S*)
proof (*induction trail S arbitrary*: *S rule*: *ann-lit-list-induct*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Decided 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 (*Propagated 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-decided*:
assumes
backtrack-lvl *T* = *count-decided* (*trail* *T*)
shows
backtrack-lvl (*cut-trail-wrt-clause* *C* (*trail* *T*) *T*) =
count-decided (*trail* (*cut-trail-wrt-clause* *C* (*trail* *T*) *T*))
using *assms*
proof (*induction* *trail* *T* *arbitrary*:*T* *rule*: *ann-lit-list-induct*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Decided* *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 (*Propagated* *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-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-lit-list-induct*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Decided* *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 clause* **where**
f6: (*mma* ∈ {{#− *l*#} |*l*. *l* ∈# *C*} ⟶ *M* $\models$*a* *mma*) ⟶ *M* $\models$*as* {{#− *l*#} |*l*. *l* ∈# *C*}
using *true-annots-def* **by** *blast*
have *mma* ∈ {{#− *l*#} |*l*. *l* ∈# *C*} ⟶ *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* ∈# *C*}

using *f6 True M bt* **by** (*force simp*: *count-eq-zero-iff*)
then show *?thesis*
using *IH true-annots-true-cls M* **by** (*auto simp*: *CNot-def*)
qed
next
case (*Propagated L l M*) **note** $IH = this(1)[of\ tl\text{-}trail\ T]$ **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 clause* **where**
f6: $(mma \in \{\{\#-\ l\#\}\ |l.\ l \in\#\ C\} \longrightarrow M \models a\ mma) \longrightarrow M \models as\ \{\{\#-\ l\#\}\ |l.\ l \in\#\ C\}$
using *true-annots-def* **by** *blast*
have $mma \in \{\{\#-\ l\#\}\ |l.\ l \in\#\ C\} \longrightarrow trail\ T \models a\ mma$
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 simp*: *count-eq-zero-iff*)
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*:
$((\forall L \in\# C.\ -L \notin lits\text{-}of\text{-}l\ (trail\ T)) \wedge trail\ (cut\text{-}trail\text{-}wrt\text{-}clause\ C\ (trail\ T)\ T) = [])$
$\vee (-lit\text{-}of\ (hd\ (trail\ (cut\text{-}trail\text{-}wrt\text{-}clause\ C\ (trail\ T)\ T))) \in\#\ C$
$\wedge\ length\ (trail\ (cut\text{-}trail\text{-}wrt\text{-}clause\ C\ (trail\ T)\ T)) \geq 1)$
using *assms*
proof (*induction trail T arbitrary*:*T rule*: *ann-lit-list-induct*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Decided L M*) **note** $IH = this(1)[of\ decr\text{-}bt\text{-}lvl\ (tl\text{-}trail\ T)]$ **and** $M = this(2)[symmetric]$
then show *?case* **by** *simp force*
next
case (*Propagated L l M*) **note** $IH = this(1)[of\ tl\text{-}trail\ T]$ **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:
$trail\ S \models asm\ init\text{-}clss\ S \Longrightarrow distinct\text{-}mset\ C \Longrightarrow conflicting\ S = None \Longrightarrow$
$trail\ S \models as\ CNot\ C \Longrightarrow$
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:
$trail\ S \models asm\ init\text{-}clss\ S \Longrightarrow distinct\text{-}mset\ C \Longrightarrow conflicting\ S = None \Longrightarrow$
$\neg trail\ S \models as\ CNot\ C \Longrightarrow$
full $cdcl_W$*-stgy* (*add-init-cls C S*) *T* ⟹

incremental-cdcl$_W$ *S T*

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* $\models$*asm N* **and**
tr-C[*simp*]: *trail T* $\models$*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*]: $\bigwedge$*x*. *x* ∈ *lits-of-l* (*trail* (*cut-trail-wrt-clause C* (*trail T*) *T*)) ⟹
x ∈ *lits-of-l* (*trail T*)
using *inv-T arg-cong*[*OF M*, *of lits-of-l*] **by** *auto*
have *H'*[*dest*]: $\bigwedge$*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*:$\bigwedge$*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 20 1*)
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-l* (*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-l* (*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* **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-decided*)

have [*simp*]: $\bigwedge$*s*. *s* ∈# *learned-clss T* ⟹ ¬*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-ann-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-ann-decomposition* (*trail ?T*))
unfolding *all-decomposition-implies-def*
proof *clarify*
fix *a b*
assume (*a*, *b*) ∈ *set* (*get-all-ann-decomposition* (*trail ?T*))
from *in-get-all-ann-decomposition-in-get-all-ann-decomposition-prepend*[*OF this*, *of M*]
obtain *b'* **where**
(*a*, *b'* @ *b*) ∈ *set* (*get-all-ann-decomposition* (*trail T*))
using *M* **by** *auto*
then have *unmark-l a* ∪ *set-mset* (*init-clss T*) $\models$*ps unmark-l* (*b'* @ *b*)
using *decomp-T* **unfolding** *all-decomposition-implies-def* **by** *fastforce*
then have *unmark-l a* ∪ *set-mset* (*init-clss ?T*) $\models$*ps unmark-l* (*b* @ *b'*)
by (*simp add*: *Un-commute*)
then show *unmark-l a* ∪ *set-mset* (*init-clss ?T*) $\models$*ps unmark-l 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-ann-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) ∀ L∈#C. − L ∉ lits-of-l (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))) ∈# 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 L: get-level (trail (cut-trail-wrt-clause C (trail T) T)) (−?L)
      = count-decided (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: if-split-asm
        simp:rev-swap[symmetric] add-new-clause-and-update-def)

    have L': count-decided(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 M' D)
      then consider
          (DC) D = C
        | (D-T) D ∈# clauses T
        by (auto simp: clauses-def split: if-split-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') @ Decided K # M = trail T
            using MT 1(1) by auto
           then show 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-l M)
```

using *1*(*3*) *C in-CNot-implies-uminus*(*2*) **by** *blast*
moreover
have *lit-of* (*hd* (M' @ *Decided K* # [])) = −*?L*
using *l 1*(*1*)[*symmetric*] *inv*
by (*cases* M', *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-l* (M' @ *Decided K* # []))
by (*cases* (M' @ *Decided K* # [])) *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*] **by** (*auto simp*: *lits-of-def*)
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 rtranclp-*$cdcl_W$*-stgy-rtranclp-*$cdcl_W$ *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-l* (*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 9 1 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 *nc*: $\forall M. (\exists K\ i\ M'.\ trail\ S = M'\ @\ Decided\ K\ \#\ M) \longrightarrow \neg\ M \models as\ CNot\ C$
using ‹$\neg$ *trail S* $\models$*as CNot C*›
by (*auto simp*: *true-annots-true-cls-def-iff-negation-in-model*)

have *cdcl$_W$-stgy-invariant* (*add-init-cls C S*)
using *s-inv* ‹$\neg$ *trail S* $\models$*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*: *clauses-def nc*)
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*
proof *induction*
print-cases
case (*add-confl C T*) **note** *tr = this*(*1*) **and** *dist = this*(*2*) **and** *conf = this*(*3*) **and** *C = this*(*4*) **and** *full = this*(*5*)

have *full cdcl$_W$-stgy T T*
using *full* **unfolding** *full-def* **by** *auto*
then show *?case*
using *full C conf dist tr*
by (*metis full-cdcl$_W$-stgy-inv-normal-form incremental-cdcl$_W$.simps incremental-cdcl$_W$-inv*(*1*)
incremental-cdcl$_W$-inv(*2*) *inv s-inv*)
next
case (*add-no-confl C T*) **note** *tr = this*(*1*) **and** *dist = this*(*2*) **and** *conf = this*(*3*) **and** *C = this*(*4*)
and *full = this*(*5*)

have *full cdcl$_W$-stgy T T*
using *full* **unfolding** *full-def* **by** *auto*
then show *?case*
by (*meson C conf dist full full-cdcl$_W$-stgy-inv-normal-form incremental-cdcl$_W$.add-no-confl*
incremental-cdcl$_W$-inv(*1*) *incremental-cdcl$_W$-inv*(*2*) *inv s-inv tr*)
qed

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)

end

end
theory *CDCL-W-Restart*
imports *CDCL-W-Merge*
begin

2.4.1 Adding Restarts

locale *cdcl$_W$-restart* =

conflict-driven-clause-learning$_W$
— functions for the state:
— access functions:
trail init-clss learned-clss backtrack-lvl conflicting
— changing state:
cons-trail tl-trail add-learned-cls remove-cls update-backtrack-lvl
update-conflicting

— get state:
init-state
restart-state
for
trail :: *'st* ⇒ (*'v*, *'v clause*) *ann-lits* **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*, *'v clause*) *ann-lit* ⇒ *'st* ⇒ *'st* **and**
tl-trail :: *'st* ⇒ *'st* **and**
add-learned-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
remove-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
update-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*$\widehat{}\widehat{}$(*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 $\Longrightarrow$ snd T = 1 + snd S
by (induction rule: $cdcl_W$-merge-with-restart.induct) auto

lemma full1 $cdcl_W$-merge-stgy S T $\Longrightarrow$ $cdcl_W$-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) $\subseteq$ simple-clss (atms-of-mm (init-clss S))
proof
fix C
assume C: $C \in$ 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 $\neg$tautology C
using C inv **unfolding** $cdcl_W$-all-struct-inv-def $cdcl_W$-learned-clause-def **by** auto
moreover
have atms-of C $\subseteq$ atms-of-mm (learned-clss S)
using C **by** auto
then have atms-of C $\subseteq$ atms-of-mm (init-clss S)
using inv **unfolding** $cdcl_W$-all-struct-inv-def no-strange-atm-def **by** force
moreover have finite (atms-of-mm (init-clss S))
using inv **unfolding** $cdcl_W$-all-struct-inv-def **by** auto
ultimately show $C \in$ simple-clss (atms-of-mm (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 $\neg$?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-mm (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* ($\lambda i.$ *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*)

obtain *k* where
f-g-k: *f* (*snd* (*g k*)) > *card* (*simple-clss* (*atms-of-mm* (*init-clss* (*fst ?S*)))) and
k > *card* (*simple-clss* (*atms-of-mm* (*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-mset* (*learned-clss T*)) − *card* (*set-mset* (*learned-clss* (*fst* (*g k*)))) and
m > *f* (*snd* (*g k*)) and
restart T (*fst* (*g* (*k+1*))) and
$cdcl_W$*-merge-stgy*: ($cdcl_W$*-merge-stgy* ^^ *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 $cdcl_W$*-merge-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-mset* (*learned-clss T*)) − *card* (*set-mset* (*learned-clss* (*fst* (*g k*))))
> *card* (*simple-clss* (*atms-of-mm* (*init-clss* (*fst ?S*))))
unfolding *m*[*symmetric*] using ‹*m* > *f* (*snd* (*g k*))› *f-g-k* by *linarith*
then have *card* (*set-mset* (*learned-clss T*))
> *card* (*simple-clss* (*atms-of-mm* (*init-clss* (*fst ?S*))))
by *linarith*
moreover
have *init-clss* (*fst* (*g k*)) = *init-clss T*
using ‹$cdcl_W$*-merge-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-clss* (*fst ?S*) = *init-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-mm* (*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:
($cdcl_W$*-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 ⟹
$cdcl_W$*-with-restart* (*S*, *n*) (*U*, *Suc n*) |
restart-full: *full1* $cdcl_W$*-stgy S T* ⟹ $cdcl_W$*-with-restart* (*S*, *n*) (*T*, *Suc n*)

lemma $cdcl_W$*-with-restart-rtranclp-*$cdcl_W$:
$cdcl_W$*-with-restart S T* ⟹ $cdcl_W{}^{**}$ (*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* ⟹ *snd T* = *1* + *snd S*
by (*induction rule*: $cdcl_W$*-with-restart.induct*) *auto*

lemma *full1* $cdcl_W$*-stgy S T* ⟹ $cdcl_W$*-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* ⟹ $cdcl_W$*-M-level-inv* (*fst S*) ⟹ *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*) ∧ $cdcl_W$*-with-restart S T*}
proof (*rule ccontr*)
assume ¬ *?thesis*
then obtain *g* **where**
g: ⋀*i*. $cdcl_W$*-with-restart* (*g i*) (*g* (*Suc i*)) **and**
inv: ⋀*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-mm* (*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* (λi. *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-mm* (*init-clss* (*fst ?S*)))) **and**
k > *card* (*simple-clss* (*atms-of-mm* (*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-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-mset* (*learned-clss T*)) − *card* (*set-mset* (*learned-clss* (*fst* (*g k*)))) **and**
m > *f* (*snd* (*g k*)) **and**
restart T (*fst* (*g* (*k+1*))) **and**
cdcl$_W$-merge-stgy: (*cdcl$_W$-stgy* ^^ *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$_W$-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-mset* (*learned-clss T*)) − *card* (*set-mset* (*learned-clss* (*fst* (*g k*))))
> *card* (*simple-clss* (*atms-of-mm* (*init-clss* (*fst ?S*))))
unfolding *m*[*symmetric*] **using** ‹*m* > *f* (*snd* (*g k*))› *f-g-k* **by** *linarith*
then have *card* (*set-mset* (*learned-clss T*))
> *card* (*simple-clss* (*atms-of-mm* (*init-clss* (*fst ?S*))))
by *linarith*
moreover
have *init-clss* (*fst* (*g k*)) = *init-clss T*
using ‹*cdcl$_W$-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-clss* (*fst ?S*) = *init-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-mm* (*init-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 ∃ *k*::*nat*. (*2* ^ (*k* − *1*) ≤ *i* ∧ *i* < *2* ^ *k* − *1*) ∨ *i* = *2* ^ *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* ^ (*k* − *1*) ≤ *n* ∧ *n* < *2* ^ *k* − *1* ∨ *n* = *2* ^ *k* − *1*
by *blast*
then consider
(*st-interv*) *2* ^ (*k* − *1*) ≤ *n* **and** *n* ≤ *2* ^ *k* − *2*
| (*end-interv*) *2* ^ (*k* − *1*) ≤ *n* **and** *n* = *2* ^ *k* − *2*
| (*pow2*) *n* = *2* ^ *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* ^ (*k* − *1*) ≤ *n* ∧ *n* < *2* ^ *k* − *1* ∨ *n* = *2* ^ *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* ∃ *k*. *i* = *2^k* − *1*
then 2^((*SOME k*. *i* = *2^k* − *1*) − *1*)
else luby-sequence-core (*i* − 2^((*SOME k*. 2^(*k*−*1*)≤ *i* ∧ *i* < *2^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* ^ (*k* − *1*) ≤ *i* ∧ *i* < *2* ^ *k* − *1*)
have *2* ^ (*?k* − *1*) ≤ *i* ∧ *i* < *2* ^ *?k* − *1*
apply (*rule someI-ex*)
using *2 exists-luby-decomp* **by** *blast*
then show *?case*

proof −
have ∀ *n na*. ¬ (*1*::*nat*) ≤ *n* ∨ *1* ≤ *n* ^ *na*
by (*meson one-le-power*)
then have *f1*: (*1*::*nat*) ≤ *2* ^ (*?k* − *1*)
using *one-le-numeral* **by** *blast*
have *f2*: *i* − *2* ^ (*?k* − *1*) + *2* ^ (*?k* − *1*) = *i*
using ‹*2* ^ (*?k* − *1*) ≤ *i* ∧ *i* < *2* ^ *?k* − *1*› *le-add-diff-inverse2* **by** *blast*
have *f3*: *2* ^ *?k* − *1* ≠ *Suc 0*
using *f1* ‹*2* ^ (*?k* − *1*) ≤ *i* ∧ *i* < *2* ^ *?k* − *1*› **by** *linarith*
have *2* ^ *?k* − (*1*::*nat*) ≠ *0*
using ‹*2* ^ (*?k* − *1*) ≤ *i* ∧ *i* < *2* ^ *?k* − *1*› *gr-implies-not0* **by** *blast*
then have *f4*: *2* ^ *?k* ≠ (*1*::*nat*)
by *linarith*
have *f5*: ∀ *n na*. *if na* = *0 then* (*n*::*nat*) ^ *na* = *1 else n* ^ *na* = *n* ∗ *n* ^ (*na* − *1*)
by (*simp add*: *power-eq-if*)
then have *?k* ≠ *0*
using *f4* **by** *meson*
then have *2* ^ (*?k* − *1*) ≠ *Suc 0*
using *f5 f3* **by** *presburger*
then have *Suc 0* < *2* ^ (*?k* − *1*)

using *f1* **by** *linarith*
then show *?thesis*
using *f2 less-than-iff* **by** *presburger*
qed
qed

function *natlog2* :: *nat* ⇒ *nat* **where**
natlog2 n = (*if n* = *0 then 0 else 1* + *natlog2* (*n div 2*))
using *not0-implies-Suc* **by** *auto*
termination by (*relation measure* (λ*n*. *n*)) *auto*

declare *natlog2.simps*[*simp del*]

declare *luby-sequence-core.simps*[*simp del*]

lemma *two-pover-n-eq-two-power-n′-eq*:
assumes *H*: (*2*::*nat*) ^ (*k*::*nat*) − *1* = *2* ^ *k′* − *1*
shows *k′* = *k*
proof −
have (*2*::*nat*) ^ (*k*::*nat*) = *2* ^ *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*^*k* − *1*) = *2*^(*k*−*1*) (**is** *?L* = *?K*)
proof −
have *decomp*: ∃ *ka*. *2* ^ *k* − *1* = *2* ^ *ka* − *1*
by *auto*
have *?L* = *2*^((*SOME k′*. (*2*::*nat*)^*k* − *1* = *2*^*k′* − *1*) − *1*)
apply (*subst luby-sequence-core.simps*, *subst decomp*)
by *simp*
moreover have (*SOME k′*. (*2*::*nat*)^*k* − *1* = *2*^*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* ^ (*k* − *Suc 0*) ≤ *i* **and**
k′: *i* < *2* ^ *k′* − *Suc 0* **and**
k-k′: *k* > *k′*
shows *False*
proof −
have *2* ^ *k′* − *Suc 0* < *2* ^ (*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* ^ (*k* − *1*) ≤ *i* **and**
i-k: *i* < *2*^ *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 $\neg$ *?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 $\exists\, b.\ \forall\, n.$ *luby-sequence-core* $n \leq b$
  then obtain b **where** b: $\bigwedge n.$ *luby-sequence-core* $n \leq b$
    by *metis*
  have *luby-sequence-core* $(2\hat{}(b+1) - 1) = 2\hat{}b$
    using *luby-sequence-core-two-power-minus-one*[*of* $b+1$] **by** *simp*
  moreover have $(2::nat)\hat{}b > b$
    by (*induction b*) *auto*
  ultimately show *False* **using** b[*of* $2\hat{}(b+1) - 1$] **by** *linarith*
qed

abbreviation *luby-sequence* :: *nat* $\Rightarrow$ *nat* **where**
luby-sequence $n \equiv$ *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: $(0::nat) = 2\hat{}0-1$
    by *auto*
  show *?thesis*

by (*subst 0*, *subst luby-sequence-core-two-power-minus-one*) *simp*
qed

lemma *luby-sequence-core n* $\geq$ *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*) $\leq$ *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 +
conflict-driven-clause-learning$_W$ — functions for clauses:
— functions for the state:
— access functions:
trail init-clss learned-clss backtrack-lvl conflicting
— changing state:
cons-trail tl-trail add-learned-cls remove-cls update-backtrack-lvl update-conflicting

— get state:
init-state
restart-state
for
ur :: *nat* **and**
trail :: *′st* ⇒ (*′v*, *′v clause*) *ann-lits* **and**
hd-trail :: *′st* ⇒ (*′v*, *′v clause*) *ann-lit* **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*, *′v clause*) *ann-lit* ⇒ *′st* ⇒ *′st* **and**

tl-trail :: *'st* ⇒ *'st* **and**
add-learned-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
remove-cls :: *'v clause* ⇒ *'st* ⇒ *'st* **and**
update-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 *DPLL-CDCL-W-Implementation*
imports *Partial-Annotated-Clausal-Logic CDCL-W-Level*
begin

Chapter 3

Implementation of DPLL and CDCL

We then reuse all the theorems to go towards an implementation using 2-watched literals:

- `CDCL_W_Abstract_State.thy` defines a better-suited state: the operation operating on it are more constrained, allowing simpler proofs and less edge cases later.

3.1 Simple Implementation of the DPLL and CDCL

3.1.1 Common Rules

Propagation

The following theorem holds:

lemma *lits-of-l-unfold*[*iff*]:
  $(\forall\, c \in \mathit{set}\ C.\ -c \in \mathit{lits\text{-}of\text{-}l}\ Ms) \longleftrightarrow Ms \models\!\mathit{as}\ \mathit{CNot}\ (\mathit{mset}\ C)$
  unfolding *true-annots-def Ball-def true-annot-def CNot-def* **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* ⇒ (*'a*, *'b*) *ann-lits* ⇒ *'a literal option*
 where
 is-unit-clause l M =
   (*case List.filter* (λ*a*. *atm-of a* ∉ *atm-of* ‘ *lits-of-l M*) *l of*
     a # [] ⇒ *if M* ⊨*as CNot* (*mset l* − {#*a*#}) *then Some a else None*
   | - ⇒ *None*)

definition *is-unit-clause-code* :: *'a literal list* ⇒ (*'a*, *'b*) *ann-lits*
  ⇒ *'a literal option* **where**
 is-unit-clause-code l M =
   (*case List.filter* (λ*a*. *atm-of a* ∉ *atm-of* ‘ *lits-of-l M*) *l of*
     a # [] ⇒ *if* (∀ *c* ∈*set* (*remove1 a l*). −*c* ∈ *lits-of-l M*) *then Some a else None*
   | - ⇒ *None*)

lemma *is-unit-clause-is-unit-clause-code*[*code*]:
  is-unit-clause l M = *is-unit-clause-code l M*
proof −
  have *1*: ⋀*a*. (∀ *c*∈*set* (*remove1 a l*). − *c* ∈ *lits-of-l M*) ⟷ *M* ⊨*as CNot* (*mset l* − {#*a*#})
    using *lits-of-l-unfold*[*of remove1 - l*, *of - M*] **by** *simp*
  then show *?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-l 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* **.**
then have $a \in$ *set* [$a \leftarrow l$. *atm-of a* $\notin$ *atm-of* ‘ *lits-of-l M*]
apply (*cases* [$a \leftarrow l$. *atm-of a* $\notin$ *atm-of* ‘ *lits-of-l M*])
apply *simp*
apply (*rename-tac aa list*; *case-tac list*) **by** (*auto split*: *if-split-asm*)
then have *atm-of a* $\notin$ *atm-of* ‘ *lits-of-l M* **by** *auto*
then show *?thesis*
by (*simp add*: *Decided-Propagated-in-iff-in-lits-of-l*
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-l 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*
then show *?thesis*
apply (*cases* [$a \leftarrow l$. *atm-of a* $\notin$ *atm-of* ‘ *lits-of-l M*], *simp*)
apply *simp*
apply (*rename-tac aa list*, *case-tac list*) **by** (*auto split*: *if-split-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-l 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*
then show $a \in$ *set l*
by (*cases* [$a \leftarrow l$. *atm-of a* $\notin$ *atm-of* ‘ *lits-of-l M*])
(*fastforce dest*: *filter-eq-ConsD split*: *if-split-asm* *split*: *list.splits*)+
qed

lemma *is-unit-clause-Nil*[*simp*]: *is-unit-clause* [] *M = None*
unfolding *is-unit-clause-def* **by** *auto*

Unit propagation for all clauses

Finding the first clause to propagate

fun *find-first-unit-clause* :: *'a literal list list* $\Rightarrow$ (*'a*, *'b*) *ann-lits*
$\Rightarrow$ (*'a literal* × *'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* ∈ *set l* ∧ *M* ⊨*as CNot* (*mset c* − {#*a*#}) ∧ *undefined-lit M a* ∧ *a* ∈ *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* ⊨*as CNot* (*mset c* − {#*a*#}) **and**
  undef: *undefined-lit M a* **and**
  ac: *a* ∈ *set c*
  shows *is-unit-clause c M* ≠ *None*
proof −
  have [*a*←*c* . *atm-of a* ∉ *atm-of* ‘ *lits-of-l M*] = [*a*]
    using *assms*
    proof (*induction c*)
      case *Nil* **then show** *?case* **by** *simp*
    next
      case (*Cons ac c*)
      show *?case*
        proof (*cases a* = *ac*)
          case *True*
          then show *?thesis* **using** *Cons*
            by (*auto simp del*: *lits-of-l-unfold*
                simp add: *lits-of-l-unfold*[*symmetric*] *Decided-Propagated-in-iff-in-lits-of-l*
                  atm-of-eq-atm-of atm-of-in-atm-of-set-iff-in-set-or-uminus-in-set)
        next
          case *False*
          then have *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
  then show *?thesis*
    using *M* **unfolding** *is-unit-clause-def* **by** *auto*
qed

lemma *find-first-unit-clause-none*:
  distinct c $\Longrightarrow$ *c* ∈ *set l* $\Longrightarrow$ *M* ⊨*as CNot* (*mset c* − {#*a*#}) $\Longrightarrow$ *undefined-lit M a* $\Longrightarrow$ *a* ∈ *set c*
  $\Longrightarrow$ *find-first-unit-clause l M* ≠ *None*
  by (*induction l*)
    (*auto split*: *option.split simp add*: *propagate-is-unit-clause-not-None*)

Decide

fun *find-first-unused-var* :: *'a literal list list* ⇒ *'a literal set* ⇒ *'a literal option* **where**
find-first-unused-var (*a* # *l*) *M* =
  (*case List.find* (λ*lit*. *lit* ∉ *M* ∧ −*lit* ∉ *M*) *a of*
    None ⇒ *find-first-unused-var l M*
  | *Some a* ⇒ *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-of* ‘ *set a* $\subseteq$ *atm-of* ‘ *M*)
proof −
have *find-first-unused-var l M* $\neq$ *None*
using *assms* **by** (*cases find-first-unused-var l M*) *auto*
then show $\neg$($\forall$ *a* $\in$ *set l. atm-of* ‘ *set a* $\subseteq$ *atm-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-l Ms*) = *Some a* $\Longrightarrow$ *undefined-lit Ms a*
using *find-first-unused-var-Some*[*of l lits-of-l Ms a*] *Decided-Propagated-in-iff-in-lits-of-l*
by *blast*

3.1.2 CDCL specific functions

Level

fun *maximum-level-code*:: *'a literal list* $\Rightarrow$ (*'a*, *'b*) *ann-lits* $\Rightarrow$ *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*[*simp*]:
maximum-level-code D M = *get-maximum-level M* (*mset D*)
by (*induction D*) (*auto simp add*: *get-maximum-level-plus*)

lemma [*code*]:
fixes *M* :: (*'a*, *'b*) *ann-lits*
shows *get-maximum-level M* (*mset D*) = *maximum-level-code D M*
by *simp*

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*) ⇒ *if i = k ∧ 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 ∈ set Ls ∧ get-maximum-level M* (*mset* (*remove1 L* (*Ls* @ *D*))) = *j ∧ 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*) ⟹
     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*: *if-split-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 add.commute diff-union-single-conv in-multiset-in-set mset.simps*(*2*))
  } 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 *a2 a1*[*of L′ # D*] **unfolding** *find-def* **apply** (*metis add-diff-cancel-left′ mset.simps*(*2*)
        option.inject prod.inject union-commute)
    using *f4 a2 a1*[*of L′ # D*] **unfolding** *find-def* **by** (*metis option.inject prod.inject*)
  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 ¬(*L ∈ set Ls ∧ get-maximum-level M* (*mset D*) < *k ∧ 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′*#}) ⟹ *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*: *if-split-asm*)
qed

fun *bt-cut* **where**
bt-cut i (*Propagated - - # Ls*) = *bt-cut i Ls* |
bt-cut i (*Decided K # Ls*) = (*if count-decided Ls* = *i then Some* (*Decided K # Ls*) *else bt-cut i Ls*) |
bt-cut i [] = *None*

lemma *bt-cut-some-decomp*:
assumes *no-dup M* **and** *bt-cut i M* = *Some M′*
shows ∃ *K M2 M1*. *M* = *M2* @ *M′* ∧ *M′* = *Decided K # M1* ∧ *get-level M K* = (*i*+*1*)
using *assms* **by** (*induction i M rule*: *bt-cut.induct*) (*auto split*: *if-split-asm*)

lemma *bt-cut-not-none*:
assumes *no-dup M* **and** *M* = *M2* @ *Decided K # M′* **and** *get-level M K* = (*i*+*1*)
shows *bt-cut i M* ≠ *None*
using *assms* **by** (*induction M2 arbitrary*: *M rule*: *ann-lit-list-induct*)
(*auto simp*: *atm-lit-of-set-lits-of-l*)

lemma *get-all-ann-decomposition-ex*:
∃ *N*. (*Decided K # M′*, *N*) ∈ *set* (*get-all-ann-decomposition* (*M2*@*Decided K # M′*))
apply (*induction M2 rule*: *ann-lit-list-induct*)
apply *auto*[*2*]
by (*rename-tac L m xs*, *case-tac get-all-ann-decomposition* (*xs* @ *Decided K # M′*))
auto

lemma *bt-cut-in-get-all-ann-decomposition*:
assumes *no-dup M* **and** *bt-cut i M* = *Some M′*
shows ∃ *M2*. (*M′*, *M2*) ∈ *set* (*get-all-ann-decomposition M*)
using *bt-cut-some-decomp*[*OF assms*] **by** (*auto simp add*: *get-all-ann-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 (*Decided - # Ls*) ⇒ (*Propagated L D # Ls*, *N*, *D # U*, *j*, *None*)
| - ⇒ (*M*, *N*, *U*, *k*, *Some D*))
) |
do-backtrack-step S = *S*

end
theory *DPLL-W-Implementation*
imports *DPLL-CDCL-W-Implementation DPLL-W* ~~/*src*/*HOL*/*Library*/*Code-Target-Numeral*
begin

3.1.3 Simple Implementation of DPLL

Combining the propagate and decide: a DPLL step

definition *DPLL-step* :: *int* $dpll_W$*-ann-lits* × *int literal list list*
⇒ *int* $dpll_W$*-ann-lits* × *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-of-l 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-l Ms*) *of*
      Some a ⇒ (*Decided a* # *Ms*, *N*)
    | *None* ⇒ (*Ms*, *N*))))

Example of propagation:

value *DPLL-step* ([*Decided* (*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*) *ann-lits*)
  (*N*:: *int literal list list*). (*Ms*, *mset* (*map mset N*))
abbreviation *toS'* ≡ λ(*Ms*::(*int*, *unit*) *ann-lits*,
  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*)
    then have *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*› **by** (*auto simp add*: *C*)
    then have *?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*
    then have *is-decided L* **using** *backtrack-split-snd-hd-decided*[*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-decided L*›, *of*])

using *C Ms bt* **by** *auto*
moreover have $(Ms', N') = (Propagated\ (-\ (lit\text{-}of\ L))\ ()\ \#\ M,\ N)$
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*: $\neg\ (\exists\ C \in set\ N.\ Ms \models as\ CNot\ (mset\ C))$
obtain *L* **where** *unused*: *find-first-unused-var N* (*lits-of-l Ms*) = *Some L*
using *step exC neq* **unfolding** *DPLL-step-def prod.case unit*
by (*cases find-first-unused-var N* (*lits-of-l Ms*)) *auto*
have $dpll_W$ (*Ms*, *mset* (*map mset N*))
(*Decided 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*: *Decided-Propagated-in-iff-in-lits-of-l atms-of-ms-def*)
moreover have $(Ms', N') = (Decided\ L\ \#\ Ms,\ N)$
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*: $\exists\ C \in set\ N.\ Ms \models as\ CNot\ (mset\ C)$
then have *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*) = []
then show *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*
then have *Ms*: *Ms* = *Propagated* (− *lit-of aa*) () # *list* **using** *Ms* **by** *auto*
have *is-decided aa* **using** *backtrack-split-snd-hd-decided*[*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*
then show *snd* (*backtrack-split Ms*) = [] **by** *blast*
qed

then have *?thesis*
using *n backtrack-snd-empty-not-decided*[*of Ms*] **unfolding** *conclusive-dpll*$_W$*-state-def*
by (*cases backtrack-split Ms*) *auto*

}
moreover {
assume n: $\neg$ ($\exists$ $C \in$ *set* N. *Ms* $\models$*as CNot* (*mset* C))
then have *find-first-unused-var* N (*lits-of-l Ms*) = *None*
using *step* unfolding *DPLL-step-def* by (*simp add*: *unit split*: *option.splits*)
then have a: $\forall$ $a \in$ *set* N. *atm-of* ‘ *set* $a \subseteq$ *atm-of* ‘ (*lits-of-l Ms*) by *auto*
have *fst* (*toS Ms N*) $\models$*asm snd* (*toS Ms N*) unfolding *true-annots-def CNot-def Ball-def*
proof *clarify*
fix x
assume x: $x \in$ *set-mset* (*clauses* (*toS Ms N*))
then have $\neg$*Ms* $\models$*as CNot* x using n unfolding *true-annots-def CNot-def Ball-def* by *auto*
moreover have *total-over-m* (*lits-of-l 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*) $\models$$a$ x
using *total-not-CNot*[*of lits-of-l Ms x*] by (*simp add*: *true-annot-def true-annots-true-cls*)
qed
then have *?thesis* unfolding *conclusive-dpll$_W$-state-def* by *blast*
}
ultimately show *?thesis* by *blast*
qed

Adding invariants

Invariant tested in the function **function** *DPLL-ci* :: *int dpll$_W$-ann-lits* $\Rightarrow$ *int literal list list* $\Rightarrow$ *int dpll$_W$-ann-lits* $\times$ *int literal list list* **where**
DPLL-ci Ms N =
(*if* $\neg$*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*) $\in \{(S', S).$ *dpll$_W$-all-inv S* $\wedge$ *dpll$_W$ S S'*$\}\}$)
show *wf* $\{(S', S).$(*toS' S'*, *toS' S*) $\in \{(S', S).$ *dpll$_W$-all-inv S* $\wedge$ *dpll$_W$ S S'*$\}\}$
using *wf-if-measure-f*[*OF dpll$_W$-wf*, *of toS'*] by *auto*
next
fix *Ms* :: *int dpll$_W$-ann-lits* and *N x xa y*
assume$\neg$ $\neg$ *dpll$_W$-all-inv* (*toS Ms N*)
and *step*: x = *DPLL-step* (*Ms*, *N*)
and x: (*xa*, *y*) = x
and (*xa*, *y*) $\neq$ (*Ms*, *N*)
then show ((*xa*, *N*), *Ms*, *N*) $\in \{(S', S).$ (*toS' S'*, *toS' S*) $\in \{(S', S).$ *dpll$_W$-all-inv S* $\wedge$ *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-lits* $\Rightarrow$ *int literal list list* $\Rightarrow$ *int dpll$_W$-ann-lits* $\times$ *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*: *if-split 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*)
then have *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*
then have *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*)
then have *?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-step* (*Ms*, *N*) **by** (*cases DPLL-step* (*Ms*, *N*)) *auto*

{ assume ¬$dpll_W$*-all-inv* (*toS Ms N*)
then have (*Ms*, *N*) = (*Ms'*, *N*) **using** *step* **by** *auto*
then have *?case* **by** *auto*
}
moreover
{ assume $dpll_W$*-all-inv* (*toS Ms N*)
and (S_1, S_2) = (*Ms*, *N*)
then have *?case* **using** *S step* **by** *auto*
}
moreover
{ assume $dpll_W$*-all-inv* (*toS Ms N*)
and (S_1, S_2) ≠ (*Ms*, *N*)
moreover obtain S_1' S_2' **where** *DPLL-ci* S_1 *N* = (S_1', S_2') **by** (*cases DPLL-ci* S_1 *N*) *auto*
moreover have *DPLL-ci Ms N* = *DPLL-ci* S_1 *N* **using** *DPLL-ci.simps*[*of Ms N*] *calculation*
proof −
have (*case* (S_1, S_2) *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* **by** *presburger*
then have (*if* (S_1, S_2) = (*Ms*, *N*) *then* (*Ms*, *N*) *else DPLL-ci* S_1 *N*) = *DPLL-ci Ms N*

by *fastforce*
then show *?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) ≠ (*Ms*, *N*)› *prod.sel*(*2*) *snd-DPLL-step*)
ultimately have *?case* **by** (*metis* (*mono-tags*, *hide-lams*) *IH S* ‹(S_1, S_2) ≠ (*Ms*, *N*)›
‹*DPLL-ci Ms N* = *DPLL-ci* S_1 *N*› ‹$dpll_W$*-all-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 $dpll_W$*-all-inv* (*Ms*, *N*)
and $dpll_W^{++}$ (*Ms*, *N*) (*Ms*, *N*)
shows *False*
proof −
have *1*: *wf* {(*S′*, *S*). $dpll_W$*-all-inv S* ∧ $dpll_W^{++}$ *S S′*} **using** $dpll_W$*-wf-tranclp* **by** *auto*
have ((*Ms*, *N*), (*Ms*, *N*)) ∈ {(*S′*, *S*). $dpll_W$*-all-inv S* ∧ $dpll_W^{++}$ *S S′*} **using** *assms* **by** *auto*
then show *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*
then have *DPLL-ci Ms′ N* = (*Ms*, *N*) **using** *step inv st Ms′N*[*symmetric*] **by** *fastforce*
then have $dpll_W^{++}$ (*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*)
then show *False* **using** $dpll_W$*-all-inv-*$dpll_W$*-tranclp-irrefl inv* **by** *auto*
qed
then show *?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*)

then have *?case* **using** *that* **by** *auto*
}
moreover {
assume *n*: $(S, N) \neq (Ms, N)$
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*›)
then have *?thesis*
using *IH that* **by** *fastforce*
}
moreover {
assume *n*: (*S*, *N*) = (*Ms*, *N*)
then have *?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*)
then have *?case* **using** *step* **by** *auto*
}
moreover {
assume $dpll_W$*-all-inv* (*toS Ms N*)
and (S_1, *N*) = (*Ms*, *N*)
then have *?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*
then have (*if* (S_1, *N*) = (*Ms*, *N*) *then* (*Ms*, *N*) *else DPLL-ci* S_1 *N*) = *DPLL-ci Ms N*
by *fastforce*
then show *?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*) *ann-lits* **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*
then have *star*: $dpll_W{}^{**}$ (*toS Ms N*) (*toS Ms' N*) **unfolding** *MsN* **using** *DPLL-ci-*$dpll_W$*-rtranclp* **by** *blast*
then have *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*) *ann-lits*, *N*::*int literal list list*).
$dpll_W$*-all-inv* (*toS M N*)}
morphisms *rough-state-of state-of*
proof
show ([],[]) ∈ {(*M*, *N*). $dpll_W$*-all-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-state-of S* = *rough-state-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*) ∈ {(*M*, *N*). $dpll_W$*-all-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*: $x = DPLL\text{-}step'\ S$
 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** ⟨$x \neq S$⟩ *rough-state-of-inject x* **by** *fastforce+*
   then show *?thesis* **unfolding** *Ms*[*symmetric*] *Ms'*[*symmetric*] **by** *auto*
  qed
 ultimately show $(x, S) \in \{(T', T). (rough\text{-}state\text{-}of\ T', rough\text{-}state\text{-}of\ T)$
  $\in \{(S', S). (toS'\ S', toS'\ S) \in \{(S', S). dpll_W\text{-}all\text{-}inv\ S \wedge dpll_W\ S\ S'\}\}\}$
  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*
  then have *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*)
  then show *?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

Code export

A conversion to *DPLL-W-Implementation.dpll*$_W$*-state* **definition** *Con* :: (*int*, *unit*) *ann-lits* × *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-state-of* (*DPLL-tot S*)) *in* (∀ *A* ∈ *set N*. (∃ *a*∈*set A*. *a* ∈ *lits-of-l* (*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